Multiplex: A Formal Model for Multidatabases and Its Implementation [*]

Amihai Motro

Department of Information and Software Engineering
George Mason University
Fairfax, VA 22030-4444
ami@gmu.edu

Technical Report ISSE-TR-95-103

March 1995, revised February 1999

Abstract

The integration of information from multiple databases has been an enduring subject of research for almost 20 years, and many different solutions have been attempted or proposed. Missing from this research has been a uniform framework. Usually, each solution develops its own ad-hoc framework, designed to address the particular aspects of the problem that are being attacked and the particular methodology that is being used. To address this situation, in this paper we define a formal model for multidatabases, which we call Multiplex. Multiplex is a simple extension of the relational model, which may serve as a uniform abstraction for many previous ad-hoc solutions. Multiplex is based on formal assumptions of integrability, which distinguish between scheme and instance reconcilability among independent databases. Multiplex supports database heterogeneity, and it provides several degrees of freedom that allow it to model actual situations encountered in multidatabase applications. In addition, in situations in which a single answer is not obtainable (either because the global query is not answerable, or there are multiple candidate answers), Multiplex defines approximative answers. Finally, Multiplex provides a practical platform for implementation. A prototype of such an implementation is described briefly.

[*]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

The integration of information from multiple databases has been an enduring subject of research for almost 20 years. (Surveys of this area include [5, 7, 27, 36]; collections of articles on this topic include [19, 34, 18, 14]; recent workshops include [35, 13].) Indeed, while the solutions that have been advanced tended to reflect the research approaches prevailing at their time, the overall goal has remained mostly unchanged: to provide flexible and efficient tools for retrieving information from a collection of distributed, heterogeneous and overlapping databases.[1]

A standard approach to this problem has been to integrate the independent databases by means of a comprehensive *global scheme* that models the information contained in the entire collection of databases (for example, [22, 38, 6, 29]). This global scheme is fitted with a *mapping* that defines the elements of the global scheme in terms of elements of the schemes of the member databases. Algorithms are designed to interpret queries on the global scheme. Such *global queries* are translated (using the information captured in the mapping) to queries on the member databases; the individual answers are then combined to an answer to the global query. The global scheme and the scheme mapping constitute a *virtual database*; the main difference between a virtual database and a conventional database is that whereas a conventional database contains data, a virtual database points to other databases that contain the data. An important concern is that this query processing method be *transparent*; i.e., users need not be aware that the database they are accessing is virtual.

An attractive alternative to this architecture is the *federated* architecture [17]. Briefly, in a federated architecture each database system contains an *import scheme* which specifies the information available to it from external sources (and how it is obtained). In essence, the import scheme and its mapping correspond to a *partial* global scheme, and the main distinguishing feature of the federated architecture is that every database system plays the role of both global and local database. Other approaches include multidatabase languages (for example, [26]) and interoperable database systems (for example, [21]).

Much of the work in this area has been on the construction of global schemes (either comprehensive or partial); the main issue here is the resolution of *intensional inconsistencies* (semantic heterogeneity) among the member schemes (for example, [4, 15, 21]). Yet the complementary problem of *extensional inconsistencies* has received much less attention. This problem arises when alternative sources with overlapping information provide mutually inconsistent information, and requires methods for resolving such inconsistencies in global answers [32, 1].

Inconsistencies result in multiple candidate answers; the dual problem also exists, in which a global query might have no answer at all. Such situations often occur when a member database becomes temporarily unavailable. In such cases, rather then reject the query altogether, it is desirable to *approximate* the global answer using whatever information that is available [9, 8].

[1]More generally, the problem may involve other kinds of information sources as well.

Often missing from multidatabase research is a *uniform* framework; each solution formulates its own version of the problem, states its own assumptions, develops its own ad-hoc model, and proposes its own solutions. Often missing as well is a *formal* treatment of the subject, with precise formulation of all assumptions, definitions and algorithms.

In this paper we address these and other issues in a model and a system for multidatabases, which we call Multiplex. Several important features of the Multiplex model are elaborated below.

(1) **Extension of the relational model**. Multiplex extends the definition of relational schemes, constraints, queries and answers to an environment of multiple databases, and it provides a common ground for other definitions as they become necessary. The extension also retains the attractive *simplicity* of the relational model, with relatively few new concepts.

(2) **Formal assumptions of integrability**. Most integration methods have operated under tacit assumptions regarding the mutual consistency of the schemes and/or the instances of the underlying databases. Multiplex defines two kinds of inconsistency (intensional and extensional) and formulates two assumptions of integrability: the Model Consistency Assumption and the Instance Consistency Assumption. These explicit assumptions, which have been absent from previous work, provide an unambiguous framework, and help to classify other integration approaches.

(3) **Abstraction of various ad-hoc solutions**. The Multiplex model can serve as an abstract model behind most of the approaches that rely on a comprehensive global scheme. In addition, the Multiplex model can capture the essential features of federated architectures.

(4) **Full support for heterogeneity.** The simplicity and popularity of the relational model makes it an ideal integration model, and the integrated view that Multiplex provides is indeed relational. Yet, there is no restriction on the underlying data models; the only requirement is that they communicate their results in tabular form. Consequently, the member databases in a Multiplex multidatabase may be relational, object-oriented, or, in general, stored in any software system that can respond to requests with tabulated answers.

(5) **Flexibility to model real-world situations**. The Multiplex model is distinguished by several *degrees of freedom* that allow it to model actual situations encountered in multidatabase applications. Specifically,

1. **Source unavailability**. Multiplex reflects the dynamics of multidatabase environments where some member databases may become temporarily unavailable, and some global queries might therefore be unanswerable in their entirety.

2. **Source inconsistency**. Multiplex accepts that requested information may be found in more than one database, and admits the possibility of inconsistency among these multiple versions.

3. **Ad-hoc integration**. Multiplex permits ad-hoc global schemes of limited scope, that cull from existing databases only the information relevant to a given application.

Intuitively, these degrees of freedom correspond to mappings (from the global scheme to the member schemes) that are *neither total, nor single-valued, nor surjective.* We note that earlier approaches to global scheme integration were based on often unrealistic assumptions that existing database schemes could be integrated completely and perfectly in a single global scheme (i.e., mappings that are total, surjective, and single-valued; sometimes even one-to-one). The complexity of existing databases quite often renders this approach unrealistic. The abovementioned degrees of freedom are therefore important, as they represent a significant departure from earlier approaches.

(6) **Approximative answers**. Because Multiplex mappings are not total, global queries may be *unanswerable*; because the mappings are not single-valued and because there is no assumption of mutual consistency among the member databases, global queries may have *several candidate answers.* In both these situations, Multiplex defines *approximative answers.* The overall concern is that when a single authoritative answer is unavailable, a multidatabase system should approximate the request as best as possible. For example, the best approximation in response to a query on the names, salaries, departments and locations of all employees, could be the names and departments of all employees, the salaries of some employees, and the locations for none of the employees. As another example, the best approximation in response to a query on the employees who earn over 50, could be the employees who earn over 40. Note that the former approximative answer was "less" than what was requested, whereas the latter was "more" than what was requested, corresponding to "below" and "above" approximations.

(7) **Quick adaptation to evolving environments**. Present data environments may be highly dynamic; for example, newly discovered data sources may need to be incorporated, the structure of existing data sources may change, or existing data sources may need to be deleted altogether. In Multiplex such changes are easy to effect. As we shall see, the integration consists of providing pairs of equivalent views: a view on the global database ("the information needed"), and a view on a member database ("how it is materialized"). The complexity of these views can vary greatly: they could range from a complex calculation, to a statement that simply denotes the equivalence of two attribute names.

(8) **A practical platform for implementation**. Finally and most importantly, the Multiplex model is a practical platform for implementation. Hence, Multiplex is not only a formal model, but a practical system as well. We note that the software architecture to implement Multiplex is a relatively simple *generalization* of a relational database system.

With respect to limitations, we note that Multiplex queries and mappings are based on conjunctive views (queries may also use aggregate functions). Although a language based on conjunctive views and aggregate functions does not have the full power of the relational algebra or calculus, it is a powerful language nonetheless. The Multiplex model involves several major computations: the translation of a global query to a set of queries on the member databases, the reduction of a set of database constraints to constraints that are applicable to a given query, and the calculation of lower and upper approximations of the requested answer either when this answer is not available or when multiple candidate answers are available. Most of these problems were investigated and algorithms are known; efficient algorithms for the other problems still need to be developed.

The recent explosion of on-line information sources on the Internet has increased the interest in this area significantly, with several systems that are roughly in the same class as Multiplex. These include SIMS [3], TSIMMIS [16], UniSQL [20], and the Information Manifold [2]. These systems are discussed later in this paper.

Section 2 defines the database concepts that will be used later. Section 3 discusses integrability, and defines multidatabases and multidatabase queries and answers. Section 4 extends the model to approximative answers. Section 5 describes the Multiplex prototype, Section 6 compares Multiplex to several other systems that share similar goals, and Section 7 concludes this paper with a brief summary and a discussion of further research issues.

2 Relational Databases

In this section we define the database concepts that will be used throughout this work. Our formalization of relational databases is mostly conventional, where it might differ from the standard definitions (e.g., the various view relationships and the treatment of constraints as views) the new definitions are nevertheless strictly within the conventional axiomatization of the relational model.

2.1 Schemes and Instances

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 $(i = 1, \ldots, n)$.

As an example, consider the attribute set $T = (Name, Level, Salary)$ and the domains $dom(Name) = \{smith, jones, brown\}$, $dom(Level) = \{junior, senior\}$, and $dom(Salary) = \{20, 30, 40\}$. Example of tuples of the relation scheme $Emp = (Name, Level, Salary)$ are $(smith, junior, 20)$, $(brown, senior,\ 40)$, and $(jones, senior, -)$. A relation instance is usually written in tabular form; for example, *Emp*

Name	*Level*	*Salary*
smith	*junior*	*20*
brown	*senior*	*40*
jones	*senior*	–

is the relation instance that consists of the three aforementioned tuples,

Although the definition of a relation scheme maintains an order among its attributes, and the definition of a tuple maintains an order among its values, we shall consider a relation scheme R with m attributes and an instance r of R *interchangeable* with a relation scheme R' and an instance r' of R', if there exists a permutation of the numbers $1, \ldots, m$ that maps R to R' and the tuples of r to the tuples of r'.

2.2 Views and Queries

Let D be a database scheme. A *view* V of D is an expression of the form:

$$V = \{(a_1, \ldots, a_k) \mid (\exists b_1, \ldots, b_p)\ \psi_1 \wedge \ldots \wedge \psi_q\}$$

Where the ψ's may be of two kinds:

1. **membership:** $(c_1, \ldots, c_l) \in R$, where R is a relation scheme in D (of arity l), and the cs are either as or bs or constants,
2. **comparative:** $d_1\, \theta\, d_2$, where d_1 is either an a or a b, d_2 is either an a or a b or a constant, and θ is a comparator (e.g., $<, \leq, >, \geq, =, \neq$),

and each a must appear exactly once among the cs, and each b must appear at least once among the cs.

Since each a appears in exactly one membership formula, it is associated with a unique attribute. Consider the membership formula $(c_1, \ldots, c_l) \in R$, where $R = (A_1, \ldots, A_l)$. If $c_j = a_i$, then a_i is associated with the attribute A_j. The tuple of the attributes associated with $(a_1, \ldots, a_k)$ is the *scheme* of the view V.[2]

Given a database instance d of the database scheme D, the view V defines the following relation v:

$$v = \{(a_1, \ldots, a_k) \mid (\exists b_1, \ldots, b_p)\ \psi_1 \wedge \ldots \wedge \psi_q\}$$

where each occurrence of a relation scheme R in a formula ψ is replaced by the corresponding relation instance r. v is also called the *extension* of the view V in the database instance d.

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.

[2]We shall use V to denote both the view and its *scheme*.

The views and queries defined in this way are known as *conjunctive* views [39]. 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). We shall also use the relational algebra notation in definitions of views.

As an example, assume the relation schemes $Emp = (Name, Salary, Dname)$ and $Dept = (Dname, Supervisor)$, and consider the view

$$Emp\_sup = \{(a_1, a_2) \mid (\exists b_1, b_2)\ (a_1, b_1, b_2) \in Emp\ \wedge\ (b_2, a_2) \in Dept\}$$

The scheme of Emp_sup is $(Name, Supervisor)$. Alternatively, this view can be expressed with this relational algebra expression:

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

Assume a view V of a database scheme D. It is possible to add V to the database scheme D, and to add to every database instance d of D the corresponding relation instance v (the extension of V in d). It is then possible to define views of the new database scheme. In particular, it is possible to define views of the view V, and to compute the extensions of these views in every database instance.

Given views V' and V of a database scheme D, V' is a *subview* of V, denoted $V' \sqsubseteq V$, if there exists a selection-projection view W of V, such that the schemes of V' and W are identical, and in every database instance d of D, the extensions of V' and W are identical. V is then a *superview* of V'

Note the difference between this definition of subview and the common definition of *contained view*, denoted $V' \subseteq V$, which is based on the containment of two sets of tuples. By restricting W to selection only, the concept of subview is reduced to contained view. As an example, assuming the same relation schemes, consider these views:

$$\begin{aligned}
V &= project_{Name,Salary}\, select_{(Emp.Dname=Dept.Dname)\wedge(Supervisor=jones)} Emp \times Dept \\
V' &= project_{Name}\, select_{(Emp.Dname=Dept.Dname)\wedge(Supervisor=jones)\wedge(Salary<30)} Emp \times Dept \\
W &= project_{Name}\, select_{Salary<30} V
\end{aligned}$$

The view V' (names of employees who are supervised by Jones and who earn less than 30) is a subview of the view V (names and salaries of employees who are supervised by Jones). The view W establishes this relationship.

Assume that V' is a subview of V and now let W denote a relation scheme consisting of the attributes that are in V but not in V'. Without loss of generality, we may assume that W is the "suffix" of V (i.e., V' concatenated with W yields V). Let w be the instance of W which has a single tuple composed entirely of null values. The *enlargement* of V' to V is a view whose scheme is V, and for every database instance d of D, its extension is $v' \times w$, where v' is the extension of V' in d. Intuitively, the enlargement of V' to V involves extending the tuples of every instance of V' with null values.

Assume that V_1 and V_2 are subviews of V. The *subview union* of V_1 and V_2 *over* V, denoted $V_1 \sqcup V_2$, is the union of their enlargements to V. Assume that V_1 and V_2 are superviews of V. The *superview intersection* of V_1 and V_2 *over* V, denoted $V_1 \sqcap V_2$, is the intersection of their projections on V.[3]

When the definition of V may be assumed from the context, we shall call these operations simply the *subview union* and *superview intersection* of V_1 and V_2. Note that these operations generalize the union and intersection operations on views, which are commonly defined only for views that have the same scheme, because when V_1, V_2 and V all have the same scheme, these new operations are reduced to the common view operations. As we shall see, the subview union and the superview intersection will be used to provide *lower and upper bounds* of the view V.

Views V_1 and V_2 of a database scheme D are *overlapping*, if there exists a view V of D, such that $V \sqsubseteq V_1$ and $V \sqsubseteq V_2$, and there exists some instance d of D in which the extension of V is non-empty.

As an example, assuming the same relation schemes, consider these views:

$$\begin{aligned}
V_1 &= project_{Name,Salary} select_{(Emp.Dname=Dept.Dname)\wedge(Supervisor=jones)} Emp \times Dept \\
V_2 &= project_{Name,Depart} Emp \\
V &= project_{Name} select_{(Emp.Dname=Dept.Dname)\wedge(Supervisor=jones)} Emp \times Dept
\end{aligned}$$

The views V_1 (names and salaries of employees who are supervised by Jones) and V_2 (names and departments of employees) are overlapping. The view V (names of employees who are supervised by Jones) establishes this relationship.

2.3 Integrity Constraints

Quite often the information stored in a database must satisfy specific relationships. These relationships, called integrity constraints, restrict the allowable instances of a database. Our definition of integrity constraints follows the one in [30].

An *integrity constraint* I on a database scheme D is a view of D. A database instance d of D *satisfies* an integrity constraint I, if the extension of I in d is the empty set.

As an example, consider the relation scheme *Emp = (Name,Level,Title,Salary,Supervisor)* and the integrity constraints

$$\begin{aligned}
I_1 &= project_{Name} select_{(Level=junior)\wedge(Title=manager)} Emp \\
I_2 &= project_{Name.1,Name.2} select_{(Level.1=Level.2)\wedge(Salary.1\neq Salary.2)} Emp \times Emp
\end{aligned}$$

The first integrity constraint is satisfied in any database instance that does not have a tuple in which $Level = junior$ and $Title = manager$. Intuitively, it models a real world restriction

[3]Note that it may be possible to *identify* separate tuples by using additional information that may be available, such as functional dependencies [28].

that junior employees may not be managers. The second integrity constraint[4] is satisfied in any database instance that does not have two tuples with the same $Level$ but different $Salary$. It models the real world restriction that employees at the same level earn the same salary; i.e., the functional dependency $Level \longrightarrow Salary$.

Of course, the expressive power of integrity constraints corresponds to the expressive power of queries. For example, if a query can be formulated to retrieve the employees who are paid more than their supervisors, then the constraint could be formulated that all employees may not earn more than their supervisors.

Assume a constraint I and a view V of a database scheme D. I is *applicable* to V, if $I \sqsubseteq V$.

For example, with the previous relation scheme consider the view

$$V_1 = project_{Name,Level} select_{Title=manager} Emp$$

The constraint I_1 (there are no junior managers) is applicable to the view V_1 (the names and levels of managers).

Assume a constraint I and a query Q on a database scheme D, and assume that I and Q are overlapping views. Let $I' = I \sqcap Q$. I', called the *reduction* of I to Q, is a constraint applicable to Q.

Given a database scheme D, a set C of integrity constraints on D, and a query Q on D, the answer q to Q in every instance d of D satisfies the reduction to Q of every constraint in C.

For example, with the previous scheme consider the view

$$V_2 = project_{Name} select_{Supervisor=jones}(Emp)$$

The reduction of the constraint I_1 to the view V_2 is given by

$$I_3 = project_{Name} select_{(Level=junior)\wedge(Title=manager)\wedge(Supervisor=jones)}(Emp)$$

The constraint I_3 (there are no junior managers working for Jones) is applicable to the view V_2 (the employees supervised by Jones). The transformation of database constraints to constraints that are applicable to a given view is similar to *constraints residues* [11] or *intensional answers* [31].

2.4 Database

Finally, a *database* (D, C, d) is a combination of a database scheme D, a set C of integrity constraints on the scheme D, and a database instance d of the scheme D that satisfies all the

[4]The definition uses a simple notational device to distinguish between multiple occurrences of an attribute in the same view.

integrity constraints in C. A database (D, C, d) acts as a *function* from queries to answers: given a query Q on scheme D, it computes its answer q in the instance d. We shall use the term *database model* to refer collectively to the scheme and the constraints.

3 Multidatabases

We begin by defining derivative databases; i.e., databases derived from other databases. This notion is necessary to define equivalence between views and constraints from different databases, which in turn provides a method for expressing the commonality of two database models using model mappings. Model mappings are the basis for our definition of multidatabases. We then formulate the Model Consistency Assumption and the Instance Consistency Assumption. These assumptions lead to fundamental observations concerning the integrability of independent databases. We complete the description of the Multiplex multidatabase model by defining multidatabase queries.

3.1 Derivative Databases

Consider a database (D, C, d). Let D' be a database scheme whose relation schemes are defined as views of the relation schemes of D.[5] Intuitively, the views that transform D to D' also imply a set of integrity constraints C' and a database instance d'. Altogether, these views determine a derivative database (D', C', d'). Formal definitions follow.

Consider a database scheme (D, C, d). Let D' be a database scheme whose relations schemes are 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. Let C' be a set of integrity constraints on the scheme D'. The integrity constraints C' are *derived* from the integrity constraints C, if for every database instance d of the scheme D that satisfies the integrity constraints C, the derived database instance d' satisfies the integrity constraints C'.

Altogether, a database (D', C', d') is a *derivative* of a database (D, C, d), if its scheme D' is derived from the scheme D; its constraints C' are derived from the constraints C, and its instance d' is derived from the instance d. When extensions are ignored, we shall also refer to the database model (D', C') as a derivative of the database model (D, C).

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.

[5]Our present definition of views permits Cartesian products, selections, and projections, but could be extended to views that involve additional operations, such as aggregations or attribute renaming.

3.2 View Equivalence

Let (D_1, C_1, d_1) and (D_2, C_2, d_2) be two derivatives of a database (D, C, d). The derivative databases are mutually "consistent" in the sense that "equivalent" views are extended identically in the databases in which they apply, and "equivalent" constraints are satisfied (or unsatisfied) simultaneously in the databases in which they apply. These notions of view and constraint equivalence are 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.

A constraint I_1 on D_1 and a constraint I_2 on D_2 are *equivalent*, if for every instance d of D I_1 is satisfied in d_1 if and only if I_2 is satisfied in d_2. Intuitively, two constraints are equivalent if they model the same real world restriction.

Assume that views V_1 of D_1 and V_2 of D_2 are equivalent, and denote their view schemes $(A_1, \ldots, A_k)$ and $(B_1, \ldots, B_k)$, respectively. Let U_1 be a subview of V_1 and let U_2 be a subview of V_2, and assume that the definition of U_2 is identical to the definition of U_1, except for consistent replacement of every attribute A_i in U_1 with the corresponding attribute B_i in U_2. It is easy to verify that U_1 and U_2 are equivalent as well.

Recalling that constraints are views, a similar result can be stated for two equivalent constraints I_1 and I_2 and two identical (up to variable renaming) constraints J_1 and J_2 which are subviews of I_1 and I_2.

3.3 Model Mapping

Given two different database models, which are both derivatives of the same data model (the "reference model"), we express their commonality by means of model mappings.

Assume two database models (D_1, C_1) and (D_2, C_2), which are both derivatives of a database model (D, C). 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 $V_{i,2}$. A *constraint mapping* (C_1, C_2) is a collection of constraint pairs $(I_{i,1}, I_{i,2})$ $(i = 1, \ldots, k)$, where each $I_{i,1}$ is derived from C_1, each $I_{i,2}$ is derived from C_2, and $I_{i,1}$ is equivalent to $I_{i,2}$.

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

$$(\, project_{Salary} Emp, \; project_{Sal} Employee \,)$$

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

$$(\, project_{Name,Salary} select_{Title=manager} Emp, \; project_{Ename,Sal} Manager \,)$$

To illustrate a constraint mapping, assume that C_1 includes the constraint $Bonus \leq 0.2 \cdot Year\_Sal$, whereas C_2 includes the constraint $Bonus \geq 200 \cdot Hour\_Wage$. The commonality of these constraints is possibly expressed with the pair (for simplicity, we express these constraints in logic formulas instead of views):

$$(\, 0.1 \cdot Year\_Sal \leq Bonus \leq 0.2 \cdot Year\_Sal, \; 200 \cdot Hour\_Wage \leq Bonus \leq 400 \cdot Hour\_Wage \,)$$

As with view equivalences, the statement of constraint equivalences reflects additional knowledge; in this case, that a work year is equivalent to 2,000 hours.

3.4 Multidatabase

A *multidatabase* is

1. A scheme D and a set C of integrity constraints on scheme D.
2. A collection $(D_1, C_1, d_1), \ldots, (D_n, C_n, d_n)$ of databases.
3. A collection $(D, D_1), \ldots, (D, D_n)$ of scheme mappings.
4. A collection $(C, C_1), \ldots, (C, C_n)$ of constraint mappings.

The first item defines the model 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 fourth item defines a mapping from the global constraints to the constraints of the member databases.

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 in Section 3.3, models (scheme and constraints) mappings allow to substitute certain views (or satisfaction of certain constraints) in one database with equivalent views (or satisfaction of equivalent constraints) in another database. In a multidatabase, the former database is the global database, and the latter is a member database.

This definition may be considered a formalization of *virtual databases* defined in [29]. Scheme mapping may be considered an abstraction of different solutions that have been advanced to the task of relating global schemes to schemes of member databases (e.g., [22, 38, 6, 29]).

3.5 Integrability Assumptions

The purpose of multidatabases is to integrate information from several, independent databases. Of course, the problem of integration is trivial, unless the information sources are *inconsistent*: i.e., a portion of the real world is described differently by more than one source. It has been observed recently [32] that such inconsistencies fall into two categories: (1) *intensional inconsistencies*, and (2) *extensional inconsistencies*.

Intensional inconsistencies, often referred to as *semantic heterogeneity*, are defined as differences in *modeling*. For example, differences in relation schemes, or in the semantics of individual attributes (e.g., measurement units). Extensional inconsistencies surface only *after* all intensional inconsistencies have been resolved, at a point where the systems participating in a specific transaction may be assumed to have identical intensional representation for all overlapping information. At that point it is possible that two information sources would provide different answers to the same query.

We shall assume that there exists a single (hypothetical) database that represents the real world. This ideal database includes the usual components of scheme, constraints, and instance. Its scheme and constraints constitute the perfect model, and its instance constitutes the perfect data. We now formulate two assumptions. These assumptions are similar to the Universal Scheme Assumption and the Universal Instance Assumption [28], although their purpose here is quite different. These two assumptions are statements of the *integrability* of the given databases. They use the definition of derived databases in Section 3.1.

The Model Consistency Assumption (MCA). All database models (schemes and constraints) are *derivatives* of the real world model. That is, in each database model, every relation scheme is a view of the real world scheme, and every integrity constraint is implied by the real world constraints. 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 models 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.

Although these assumptions have not been articulated before in the context of database integration, tacit assumptions have often been made. Most previous work on scheme integration has tacitly subscribed to the Model Consistency Assumption, and the different approaches to scheme integration are therefore implementations of specific techniques for reconciling modeling inconsistencies. With few exceptions in the areas of logic databases [37] and data fusion [1], most previous work on database integration has tacitly subscribed to the Instance Consistency Assumption as well, thus avoiding any possibility of data inconsistency.

The Multiplex model assumes that the Model Consistency Assumption *holds*, meaning that all differences among database models (schemes and constraints) 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 models (schemes and constraints) that are derivatives of a hypothetical real world database model; these models are related through the multidatabase model, which is yet another derivative of this perfect database model. But the member database instances are not assumed to be derivatives of the real world instance.

Clearly, without subscribing to the MCA, 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.

3.6 Discussion

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.

The results stated at the end of Section 3.2 reduce substantially the number of view pairs that need to be specified in scheme mappings. For example, when the following view pair is part of a mapping

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

many other view pairs are implied as well. For example, the equivalence of the attributes *Name* and *Ename*, the equivalence of the views that describe in each database the names of managers who earn less than 30, and so on. Hence, it is more economical to use the "largest" views possible in mappings. A similar conclusion holds for constraints mappings.

3.7 Multidatabase Queries

An essential part of the definition of a database model is its query language. The definition of a language must provide for syntax, as well as semantics; that is, one must define not only how queries are written, but also their extension in any database instance. In this section we consider multidatabase queries.

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 $V_1 = project_{A,B} R$ and $V_2 = project_{A,C} R$, and consider the query $Q = project_A R$. 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 $Q = select_{A=a} R$. 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 [23, 24], by Levy et al. [25] and by Brodsky and Motro [8]. We shall assume that a translation algorithm exists which is sound and complete; i.e., it computes all the correct translations that exist.

3.8 Multidatabase Constraints

Recall that a multidatabase query is simply a query Q of the scheme D. To answer this query it must be translated to a query of the mapped views M (i.e., the views in the first position of the view mappings; the views that can be materialized). In analogy, a multidatabase constraint is simply a constraint I on the scheme D. This constraint is satisfied if it can be translated to a constraint derivable from the mapped constraints (i.e., the constraints in the first position of the constraint mappings; the constraints that are known to be satisfied). Formally, the required transformation of the constraints is stated as follows.

Let D denote a database scheme, and let C denote a set of constraints on this scheme. Let P denote another set of constraints on D. Given a constraint I in C, transform it to an equivalent constraint defined by means of the constraints P.

However, such a transformation might not exist, or there could be multiple transformations. Hence, in general, it might not be possible to check whether a global constraint is satisfied in the multidatabase environment.

Note that under the model Consistency Assumption, the multidatabase and the member database constraints are all implied by the real world constraints and are therefore mutually consistent. If the Instance Consistency Assumption holds as well, then every multidatabase query would have at most one answer, and that answer would satisfy the global constraints. However, if the ICA does not hold, then it is possible that a multidatabase query would have several candidate answers. Each of these answers was put together from "answer components" that were retrieved from member databases, where they satisfied the constraints. Yet it is possible that some of the answers would fail the global constraints.

As an example, consider this simple multidatabase. The database scheme includes the relation scheme *Emp = (Name,Salary,Bonus)*, and the constraint that bonus is smaller than salary. The two member databases include, respectively, the relation schemes *Sal = (Name,Salary)* and *Bon = (Name,Bonus)*, without any constraints. The view mapping simply matches $project_{Name,Salary}Emp$ with *Sal*, and $project_{Name,Bonus}$ with *Bon*. Consider now a multidatabase query to list the *Emp* relation. Its answer will be formed by joining the two views in the mapping. While each view is consistent with the member constraints (there are none), the result may not be consistent with the global constraint.

Hence, the global integrity constraints should be used to prune the set of candidate answers. Indeed, other than stating the semantics of the database, this is their sole purpose. To test whether an answer satisfies the global constraints, it is necessary first to *reduce* the global constraints to constraints that apply to the query (Section 2.3). We are now ready to define multidatabase answers.

3.9 Multidatabase Answers

Assume a multidatabase with scheme D, constraints C, 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, that satisfy the constraints C.

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.[6]
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.*[7]

Of course, if we were to subscribe to the Instance Consistency Assumption, then all the solutions generated in the first case would be guaranteed to evaluate to the same answer, and the answer to Q would be this unique answer. In theory, this answer may be evaluated from an arbitrary solution to the translation problem. In practice, however, some realistic model of *cost* should be adopted, and the *cheapest* solution should be chosen. Note that under the Instance Consistency Assumption, it is possible that the translation algorithm will have no solutions. Hence, the possibilities under the ICA are one answer or no answer.

4 Approximative Answers

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 formal Multiplex model presented in Section 3, to handle these situations.

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". The most common reason for this is that some member databases are not responding. But a similar situation would occur if some of the requested information cannot be delivered due to insufficient permissions, or due to some resource having been exhausted before the entire answer could

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

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

be obtained (e.g., time, or some other cost measure). 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?

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 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.

Our discussion of approximative answers is divided into two. First we assume that the ICA holds. Next we assume that the ICA does not hold. We begin with concepts that will be used in both situations.

4.1 Sound and Complete Answers

Recall our assumption of a (hypothetical) database that represents the real world perfectly. Under the Model Consistency Assumption (which is adopted in Multiplex) all database models (schemes and constraints) are derivatives of this database, and without loss of generality we assume now that the model of the available database is *identical* to the real world model. Let (D, C, d_0) denote the real world database, and let (D, C, d) denote the actual database. Therefore, the database instance d is an *estimate* of the real database instance d_0.

Consider a query Q on the database scheme D. Let q denote its answer in the database instance d, and let q_0 denote its answer in the database instance d_0; i.e., q_0 is the perfect answer, and q is its estimate from the actual database instance. Following [30], we say that q is a *sound* answer if $q \subseteq q_0$, and q is a *complete* answer if $q \supseteq q_0$. If q is both sound and complete then q has *integrity.*

Clearly, the perfect answer q_0 lies "between" any sound answer q_s and any complete answer q_c: $q_s \subseteq q_0 \subseteq q_c$. When the perfect answer cannot be computed from the available database, sound and complete answers serve as "below" and "above" *approximations.* Clearly, it is desirable to obtain the "largest" sound answer and the "smallest" complete answer. Together, these provide the tightest approximation for the perfect answer.

Consider a *subanswer* of Q (a subview *enlarged* with nulls to the scheme of Q). The elements of the subanswer are elements of q_0; hence the subanswer is a sound answer.[8] Consider a *superanswer* of Q (a superview *projected* on the scheme of Q). The elements of q_0 are elements of the supernaswer; hence the superanswer is a complete answer. Intuitively, combining subanswers and superanswers (through union and intersection) will provide us with the aforementioned "largest' sound answer and "smallest" complete answer.

However, when the ICA does not hold, inconsistencies among subanswers and superanswers are possible. For example, we may encounter a subanswer and a superanswer where the former is not contained in the latter (i.e., some elements of the subanswer are not in the superanswer).

It is possible to define models where such inconsistencies are resolved by preferring certain answers over others (based on known external properties of the answers). Multiplex assumes that no such additional information is available, in effect accepting all answers, subanswers and superanswers as "equally good". Inconsistencies are then resolved on the basis of *voting.*

Sound answers establish that certain data are *included* in q_0; complete answers establish that certain data are *excluded* from q_0. Therefore, the soundness or completeness of an answer may be interpreted as a claim (a *vote*) on each element of the domain of the answer: a sound answer is a *yes* vote for its members, and a *maybe* vote for all its non-members. A complete answer is a *maybe* vote for its members and a *no* vote for all its non-members.

The assumption that all information is "equally good" is interpreted that each subanswer *claims* to be sound, each superanswer *claims* to be complete, and each answer *claims* to be both sound and complete. Our assumption that no additional information is available implies that all claims have the same likelihood of being correct; i.e., their individual votes have equal weights.

We now propose the following three-valued operation to combine conflicting votes:

	yes	no	maybe
yes	yes	maybe	maybe
no	maybe	no	maybe
maybe	maybe	maybe	maybe

Briefly, this operation reflects the attitude that the final verdict should be definite if and only if all votes are consistent. It is easy to verify that this operation is associative.[9]

Consider an example with two answers q_1 and q_2. There are four subsets of the answer space that are treated homogeneously: $q_1 \cap q_2$, $q_1 - q_2$, $q_2 - q_1$, and $\overline{q_1 \cup q_2}$, . The subsets for which the vote is *yes* suggest a sound answer; the subsets for which the vote is not *no* suggest a complete answer. It is easy to verify that the elements in $q_1 \cap q_2$ have a *yes* vote, elements in $q_1 - q_2$ and in $q_2 - q_1$ have a *maybe* vote, and all other elements have a *no* vote. Hence, $q_1 \cap q_2$ is voted as a sound answer, and $q_1 \cup q_2$ is voted as a complete answer.

[8] We consider a tuple with null values to be sound if its non-null values match those of a tuple of q_0.

[9] Associativity assures that the order in which the candidate answers are considered is immaterial.

4.2 Approximation under the ICA

If we assume that the Instance Consistency Assumption holds, then the only possible anomaly are queries for which "full answers" (answers that respond to the entire query) are unavailable.[10] We define "partial answers" (which the ICA would guarantee to be mutually consistent), and we show how they should be combined into *approximative answers*, consisting of the largest sound and the smallest complete answers.

Because the ICA holds, then every *subview* of Q that can be expressed with the available views would evaluate to a sound approximation of the answer to Q, and any *superview* of Q that can be expressed with the available views would evaluate to a complete approximation of the answer to Q.

As an example, assume the global scheme $Emp = (Ename, Salary, Department, Location)$ and the views

$$\begin{aligned} V_1 &= project_{Ename,Salary} Emp \\ V_2 &= project_{Ename,Salary} select_{Department=design} Emp \\ V_3 &= project_{Ename,Department} Emp \\ V_4 &= project_{Ename,Department} select_{Salary>40} Emp \\ V_5 &= project_{Department,Location} Emp \end{aligned}$$

and consider these two queries:

$$\begin{aligned} Q_1 &= project_{Ename,Salary,Department,Location} Emp \\ Q_2 &= project_{Ename,Department} select_{Salary>50 \wedge Location=midtown} Emp \end{aligned}$$

Consider first Q_1. Normally, it would be answered by joining V_1, V_3 and V_5. Suppose that V_1 and V_5 are not available. By substituting V_2 for V_1, the system can provide the names and departments of *all* the employees, and the salaries of *some* of the employees, but locations for *none* of the employees. In effect, this information is the *union* of two *subviews* of Q_1.

Consider now Q_2. Normally, it would be answered by joining V_1, V_3 and V_5 and selecting $Location = midtown$ and $Salary > 50$. Suppose that V_1 is not available. By joining V_3 and V_5 and selecting $location = midtown$ the system can provide the names and departments of the midtown employees. By itself, V_4 provides the names and departments of the employees who earn over 40. Each of these answers *contains* the requested answer. The *intersection* of these two *superviews* of Q_2 provides the requested information for the midtown employees who earn over 40, a set which is "close" to the answer.

Let $D = \{R_1, \ldots, R_n\}$ denote a database scheme, let $M = \{V_1, \ldots, V_m\}$ denote a set of views of D, and let Q be a query of D. A view V of M is a *maximal sound approximation* of Q using M, if $V \sqsubseteq Q$, and for every view V' of M: $V' \sqsubseteq Q \Rightarrow V' \sqsubseteq V$. A view V of M is a *minimal complete approximation* of Q using M, if $V \sqsupseteq Q$, and for every view V' of M: $V' \sqsupseteq Q \Rightarrow V' \sqsupseteq V$.

[10] If one or more full answers are available, then because they are mutually consistent, any answer may be chosen at random.

The following results follow from mathematical set theory. The maximal sound approximation of Q using M is the *union* of all the subviews of Q that are expressible with views of M: $\bigcup\{W_M \mid W_M \sqsubseteq Q\}$. The minimal complete approximation of Q using M is the *intersection* of all the superviews of Q that are expressible with views of M: $\bigcap\{W_M \mid W_M \sqsupseteq Q\}$.

Assume a multidatabase with scheme D, constraints C, and mapped views M. When a query Q on this multidatabase has no answer, the *approximate answer* to Q is the pair of maximal sound approximation and minimal complete approximation:

$$< \bigcup\{W_M \mid W_M \sqsubseteq Q\}, \bigcap\{W_M \mid W_M \sqsupseteq Q\} > \tag{1}$$

Observe that, because the ICA holds, when Q is answerable in its entirety, this approximation converges to a single answer, as described in Section 3.9. Note that a minimal complete approximation is not guaranteed, because a query might have no superviews that can be expressed with the available views. Assume a database scheme with two relations R_1 and R_2 and one view $V_1 = R_1$, and consider the query $Q = R_2$. Clearly, Q cannot be expressed with the available views; moreover, there is no view of the available views that is a superview of Q. On the other hand, a maximal sound approximation is guaranteed because a subview (possibly empty) of a query always exists.

4.3 Approximation when the ICA Does Not Hold

Assume now that the Instance Consistency Assumption does not hold. In constructing an approximation for a query, we must consider both full answers and partial answers (subviews and superviews of the answer).

Let $D = \{R_1, \ldots, R_n\}$ denote a database scheme, let $M = \{V_1, \ldots, V_m\}$ denote a set of views of D, and let Q be a query of D. The result of the query translation algorithm is three sets of views:

1. $q_1, \ldots, q_k$ are the available answers,
2. $s_1, \ldots, s_m$ are the available subviews of the answer, and
3. $c_1, \ldots, c_n$ are the available superviews of the answer.

We assume that all these answers are *independent* of each other; that is, none of the translations are subviews of each other.

After the appropriate enlargements and projections, we have a total of $k+m+n$ answers that *vote* on the entire answer space. Every q_i votes *yes* on its members and *no* on all other elements; every s_i votes *yes* on its members, and *maybe* on all other elements; and every c_i votes *maybe* on its members and *no* on all other elements. Observe, however, that tuples may now be only partially specified (i.e., with null values). The question whether a tuple is a member of an answer is indeed whether a tuple *could* be a member of an answer.

The set of tuples for which the vote was unanimously *yes* is adopted as a sound estimate. The tuples the vote was not unanimously *no* is a adopted as a complete estimate:

$$< \ \{t \mid t \text{ has only } \textit{yes} \text{ votes}\}, \ \{t \mid t \text{ does not have all } \textit{no} \text{ votes}\} \ > \tag{2}$$

Observe that these are now *estimates* of a sound lower bound and a complete upper bound: since the candidate answers and partial answers are not assumed to be sound or complete, the lower approximation is not guaranteed to be sound, and the upper approximation is not guaranteed to be complete.

Our assumption here is the most general: that the ICA does not necessarily hold, and that the translation algorithm delivers both full answers and partial answers (i.e., subviews and superviews). When more specific assumptions are made, this solution is simplified. First, if only full answers are considered, then these estimates are reduced, respectively, to the intersection and union of the full answers: $< \ \bigcap_{i=1}^{k} q_i, \ \bigcup_{i=1}^{k} q_i \ >$. Second, if the ICA holds, but only partial answers are available, then these estimates are reduced, respectively, to the subview union and the superview intersection (Formula (1)). Finally, if the ICA holds, and a full answer is available, then these estimates converge to the simple answer defined at the end of Section 3.9.

The above estimates are aimed at maximizing soundness (the lower estimate) and completeness (the upper estimate). It is also possible to define answers "in between" these two. For example, the set of tuples for which at most one vote was not *yes*, the set of tuples for which at most two votes were not *yes*, and so on. This creates a sequence of containing answers that increase overall completeness while reducing overall soundness. It is then possible to provide the most sound answer which meets a user requirement on *minimal answer size*, or the largest answer which meets a user requirement on *minimal soundness*, and so on.

As mentioned earlier, when additional assumptions on the candidate answers can be made, more elaborate consolidation techniques may be developed. Elsewhere, we describe how the quality of databases can be measured using the dual measures of soundness and completeness, which estimate the discrepancy between the given database and the perfect database [33]. For example, a bibliographic source might estimate its collection of citations on Italian Renaissance to be 85% sound 65% complete, an airline guide might estimate its listings of transatlantic flights on non-American carriers to be 98% sound and 80% complete, and so on. An essential part of that work is to accurately infer the quality (soundness and completeness ratios) of the answers to arbitrary queries. In the context of Multiplex, this means that candidate answers could have different quality ratings. The aforementioned voting schemes could then be adapted to consider this information.

5 Implementation

As explained in the introduction, the Multiplex model is intended to provide both a formal foundation for research in multidatabases, and a practical architecture for a multidatabase system. Of course, actual implementations must consider additional details; yet the general principles of the Multiplex model would be upheld. In this respect it is similar to the relational model itself, whose formal definitions must be augmented with practical considerations in any implementation (e.g., optimization).

A prototype of the multidatabase model described in this paper has been implemented. This software system is still evolving, and presently it does not yet include integrity constraints. A simple diagram of the system is shown in Figure 1.

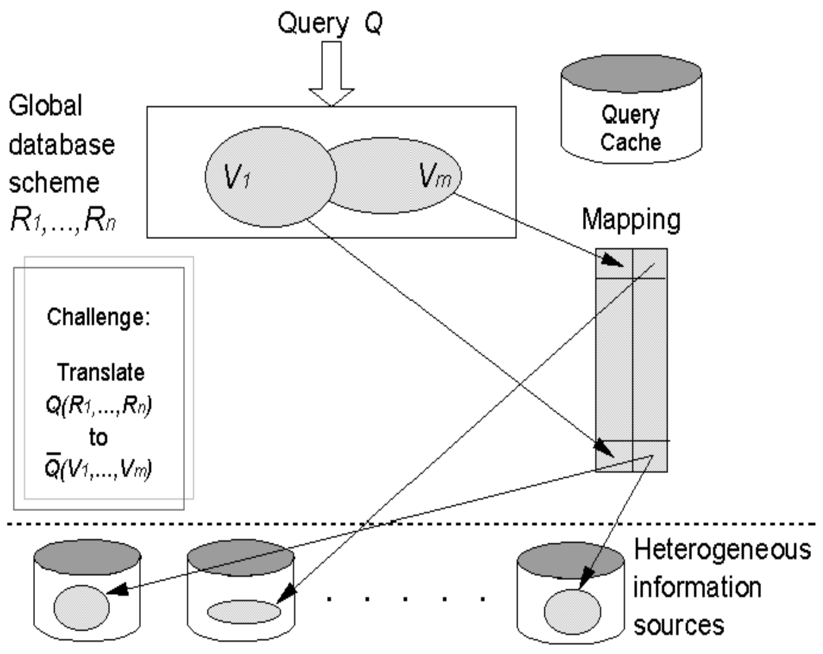
Query Q
Global database scheme $R_1,\ldots,R_n$
V_1
V_m
Query Cache
Mapping
Challenge:
Translate $Q(R_1,\ldots,R_n)$ to $\bar{Q}(V_1,\ldots,V_m)$
Heterogeneous information sources

Figure 1: A diagrammatic view of the Multiplex system.

The architecture and features of the Multiplex DBMS has six functional components: (1) user interface, (2) query parser, (3) query translator, (4) view retriever, (5) query optimizer, and (6) query processor, and it uses two sources of metadata: (1) a database scheme file, and (2) a database mapping file.

The user interface, the query parser, the query optimizer, the query processor, and the database scheme file are functionally similar to those of generic DBMS. There are two significant differences between a generic DBMS and Multiplex: (1) Multiplex does not have any relations; "instead" it has a scheme mapping file that matches views of the global scheme with views of the member databases. (2) Between the parsing and optimizing phases, which in a generic DBMS follow each other, the Multiplex query translator translates the global query to a query of the “available views”, and the Multiplex view retriever obtains the necessary views from the member databases. The translated query is then optimized and processed in the retrieved views in very much the same way as in a generic DBMS. Indeed, Multiplex retrieves views into a commercial DBMS (Orcale), and then submits the translated query to be optimized and processed by this DBMS.

The software architecture of Multiplex is described in a separate document. Here we mention briefly several notable aspects of the system:

1. Multiplex uses the HTTP protocol for communicating with the member databases. In other words, it is a World Wide Web application. The user interface is accessible via WWW browsers, and client databases are invoked by so-called “cgi scripts”.
2. For its query translation, Multiplex uses the DRP software package [12], which has been enhanced with many modifications and improvements. For example, the translated query is modified further in an attempt to reduce transmission costs, using techniques known from distributed query optimization [10].
3. With respect to heterogeneity, presently, Multiplex can retrieve information from six kinds of sources: (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) (5) spreadsheets (Microsoft Excell, HTML output), and (6) menu-based (the XLibris library retrieval system).
4. Multiplex answers (refer to Formula (2)) are presented, using color, as two relations: one contains the *sound estimate*, the other contains the tuples that augment the sound estimate to a *complete estimate* (i.e., the union of both relations is the complete estimate), and users are advised that the answer to their query is estimated to contain the first relation *plus* a subset of the second relation.
5. 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.

Overall, the linkage between the global database and the contributing sources is rather quick, requiring only entering pairs of equivalent queries in the mapping file. The result is that new information sources may be “plugged-in” very quickly.

6 Comparison with Other Approaches

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. In this section we compare our model and system to four differnt works, representing fairly different approaches.

UniSQL [20] 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 pre-defined, 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 [16] is an example of a system that is based on mediators and wrappers. *Mediators* [40] 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* [41] are simpler software interfaces that allow a 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.

Finally, it should also be noted that none of these systems considers extensional inconsistencies ("too much data") and their handling of partial ansewrs ("too little data") is quite limited.

7 Conclusion

The Multiplex model that was described in this paper is both *formal* and *pragmatic*. It is a formal extension of the relational database model to multidatabases. This formalization reflects the important pragmatic issues encountered in actual multidatabase environments, and it can serve as the formal model behind many previous ad-hoc integration models that have already been designed.

The Multiplex model is also simple to implement, and when various pragmatic issues are addressed properly, it should prove to be highly practical. Towards this goal, we review here its present limitations and discuss several open issues.

Multiplex assumes that both the multidatabase queries and the available views are conjunctive (queries can also use aggregate functions). Queries to the member databases (to materialize the available views) may be arbitrary. The assumption of conjunctive queries and views follows from current translation algorithms. Multiplex also assumes that the multidatabase constraints are conjunctive. Again, the constraints on the member databases may be arbitrary. The assumption on conjunctive constraints follows from current algorithms for inferring the constraints that apply to a specific query from the global constraints. In addition to the algorithms just mentioned, we note the still open problem of providing minimal complete estimations to unavailable answers.

Recall that the only role of multidatabase constraints in Multiplex is to possibly invalidate candidate answers. A possible direction for research is to use multidatabase constraints also to "clean-up" global answers that do not satisfy the constraints.

Multidatabase design may be described as a mediation between the information needed (as expressed in the global database scheme) and the information available. The mediation process generates mappings that match information available (a view of some member database) with information needed (a view of the global scheme). An interesting issue that was not addressed in this paper is the *design* of the mapping. For example, a problem that might concern the designer is whether a present set of mapped views "covers" the global scheme. Formally, given a database scheme $D = \{R_1, \ldots, R_n\}$, does a set of views $M = \{V_1, \ldots, V_m\}$ guarantee that every query of D is expressible with M. Moreover, is it possible to characterize the queries that are not expressible, thus suggesting a view that would complement M?

One of the unique features of Multiplex is that it resolves cross-source inconsistencies, essentially using the approximations in Formula (2). In this approach, every candidate tuple t is voted upon by each of the candidate answers. Because there are no assumptions on tuple identifiers (keys), there is no attempt to *coalesce* tuples. For example, when two employee-salary tuples (Jones, 33) and (Jones, 35) are contributed by two different sources, each would get only a single "yes" vote, implying that both would only be included in the upper bound. The present direction in the development of Multiplex is to use information about keys to coalesce tuples that share the same key. In this process, users would be given complete control over the resolution of non-key conflicts. In the above example, assuming that the first attribute is a key, the two tuples would be coalesced: the user could resolve the conflict in the second attribute in variety of ways; e.g., by designating one source as preferred, by using the median value (the value most cited), or by averaging the different values. The strategies for resolving conflicts would be stated as part of the multidatabase design.

Acknowledgment: The author is grateful to Philipp Anokhin who developed the prototype described in Section 5, and to Igor Rakov, who implemented an earlier version of the system. Both have also provided other important comments on this paper. The author is also indebted to Alex Brodsky, for his many insightful comments and suggestions.

References

[1] M.A. Abidi and R.C. Gonzalez, editors. *Data Fusion in Robotics and Machine Intelligence.* Academic Press, San Diego, CA, 1992.

[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] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *Computing Surveys*, 18(4):323–364, December 1986.

[5] Y. Breitbart. Multidatabase interoperability. *SIGMOD Record*, 19(3):53–60, September 1990.

[6] Y. Breitbart, P. L. Olson, and G. R. Thompson. Database integration in a distributed heterogeneous database system. In *Proceedings of the IEEE Computer Society Second International Conference on Data Engineering* (Los Angeles, California, February 5–7), pages 301–310, 1986.

[7] M. W. Bright, A. R. Hurson, and S. H. Pakzad. A taxonomy and current issues in multidatabase systems. *Computer*, 25(3):50–60, March 1992.

[8] 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.

[9] O. P. Buneman, S. Davidson, and A. Watters. Federated approximations for heterogeneous databases. *Data Engineering*, 3(2):27–34, August 1989.

[10] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill, New York, New York, 1984.

[11] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[12] N. Coburn. *Derived Relation Prototype: User Guide.* Department of Computer Science, University of Waterloo, 1988.

[13] P. Drew, R. King, D. McLeod, M. Rusinkiewicz, and A. Silberschatz. Report of the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, 22(3):47–56, September 1993.

[14] G. Wiederhold (Editor). Special issue: Intelligent integration of information. *Journal of Intelligent Information Systems*, 6(2/3), June 1996.

[15] D. Fang, J. Hammer, and D. McLeod. The identification and resolution of semantic heterogeneity in multidatabase systems. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 136–143, 1991.

[16] 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.

[17] D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[18] A. R. Hurson, M. W. Bright, and S. H. Pakzad, editors. *Multidatabase Systems: An Advanced Solution for Global Information Sharing.* IEEE Computer Society Press, Los Alamitos, California, 1994.

[19] Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors. *Proceedings of the First International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems (*Kyoto, Japan, April 7–9), 1991.

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

[21] R. Krishnamurthy, W. Litwin, and W. Kent. Interoperability of heterogeneous databases with semantic discrepancies. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 144–151, 1991.

[22] T. A. Landers and R. L. Rosenberg. An overview of Multibase. In H.J. Schneider, editor, *Distributed Databases*, page ?? North-Holland, Amsterdam, The Netherlands, 1982.

[23] 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.

[24] 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.

[25] 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.

[26] W. Litwin. MALPHA: A relational multidatabase manipulation language. In *Proceedings of the IEEE Computer Society First International Conference on Data Engineering* (Los Angeles, California, April 24–27), pages 86–93. IEEE Computer Society, Washington, DC, 1984.

[27] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *Computing Surveys*, 22(3):267–293, September 1990.

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

[29] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, SE–13(7):785–798, July 1987.

[30] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.

[31] A. Motro. Using integrity constraints to provide intensional responses to relational queries. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases* (Amsterdam, The Netherlands, August 22–25), pages 237–246, 1989.

[32] A. Motro. A formal framework for integrating inconsistent answers from multiple information sources. Technical Report ISSE-TR-93-106, Department of Information and Software Systems Engineering, George Mason University, October 1993.

[33] A. Motro and I Rakov. Not all answers are equally good: Estimating the quality of database answers. In *Flexible Query-Answering Systems*, pages 1–21. Kluwer Academic Publishers, 1997.

[34] H.-J. Schek, A. Sheth, and B.D. Czejdo, editors. *Proceedings of the Third International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems (*Vienna, Austria, April 19–20), 1993.

[35] P. Scheuermann, C. Yu, A. Elmagarmid, H. Garcia-Molina, F. Manola, D. McLeod, A. Rosenthal, and M. Templeton. Report on the workshop on heterogeneous database systems. *SIGMOD Record*, 19(4):23–31, December 1990.

[36] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous and autonomous databases. *Computing Surveys*, 22(3):183–236, September 1990.

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

[38] M. Templeton, D. Brill, S. K. Dao, E. Lund, P. Ward, A. L. P. Chen, and R. McGregor. Mermaid — a front-end to distributed heterogeneous databases. In *Proceedings of IEEE*, volume 75, number 5, pages 695–708, May 1987.

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

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

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