A Formal Framework for Integrating Inconsistent Answers from Multiple Information Sources *

Amihai Motro

Department of Information and Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444

Technical Report ISSE-TR-93-106

October 1993

Abstract

In any environment of multiple information sources it is practically unavoidable that information sources would *overlap*; that is, describe the same portion of the real world. And when information sources overlap, it is practically unavoidable that information would be *inconsistent*; that is, describe *differently* the same portion of the real world. In this report we define a formal framework for resolving inconsistencies that are detected in the process of processing queries against a collection of information sources. This framework establishes basic database concepts, such as schemes, instances, views, queries and integrity constraints, as well as new concepts, such as view equivalence, scheme mappings, multi-databases, multi-database queries and inconsistency. Altogether, it provides the formal foundations necessary for addressing issues of information integration and inconsistency resolution. Our approach to the integration of inconsistent answers is based on *information goodness*, a measure for estimating the proximity of stored information to the ideal information. We propose to maintain for each information source a *goodness basis*: a set of basic views whose goodness has been established and are expected to be useful for inferring the goodness of anticipated queries. When a query is answered inconsistently by individual information sources we propose to (1) infer the goodness of each individual answer from the appropriate goodness basis, and (2) integrate the individual answers in an answer of the highest goodness. We sketch the architecture of a software agent, called Harmony, that will implement our approach.

*This work was supported in part by ARPA grant, administered by the Office of Naval Research under Grant No. N0014-92-J-4038.

1 Introduction

In any environment of multiple information sources it is practically unavoidable that information sources would *overlap*; that is, describe the same portion of the real world. And when information sources overlap, it is practically unavoidable that information would be *inconsistent*; that is, describe *differently* the same portion of the real world.

Inconsistency is a most challenging issue in any attempt at information integration and interchange. Inconsistencies fall into two main categories: (1) intensional inconsistencies, and (2) extensional inconsistencies.

Intensional inconsistencies, often referred to as *semantic heterogeneity* occur when overlapping information is represented differently in the different sources. Simple examples include (1) naming conflicts (e.g., *DOB* vs. *Date_of_birth*); (2) use of different units (e.g., inches vs. centimeters); (3) conflicting levels of generalization (e.g., a *Person* entity with a *Sex* attribute vs. separate *Male* and *Female* entities). A more difficult case of intensional inconsistency is when the sources employ altogether different models (e.g., an object-oriented model with frame-based knowledge representation vs. a relational model with rule-based knowledge representation).

The subject of intensional inconsistency has been receiving attention for almost two decades, and researchers have developed different methods for coalescing the semantics of independent database schemes, or for constructing gateways that will allow different database systems to exchange information. For the most part, this research has been either in the context of view integration as part of the database design process (e.g., [4, 9, 12]) or in the context of systems for virtual merging of independent databases (e.g., [13, 14, 7]). For overviews of this area, see [2, 5].

Compared to this focus on intensional inconsistency, the equally challenging problem of extensional inconsistency has received much less attention. 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 two different answers to the same query.

We may assume that individual information sources would be responsible for resolving all their *internal* extensional inconsistencies. The problem here is similar to "simple" database inconsistency, and is usually addressed with a combination of proper design that avoids repetitions, and appropriate integrity constraints to control those repetitions allowed to remain. However, the autonomy assumed in environments of multiple information sources implies that extensional inconsistencies across the environment are entirely possible.

Currently, most architectures for integrating information from multiple sources approach this issue in one of two ways. Some systems (e.g., [1]) assume that the information sources are never inconsistent. Thus, when the same information could be retrieved from multiple sources, the cheapest source (in terms of variables such as communication costs) is used.

Usually, this assumption of perfect consistency is unrealistic. Other approaches (e.g., [10]) detect and report inconsistencies, but do not attempt to resolve them in any way.

In this report we define a formal framework for resolving extensional inconsistencies that are detected in the process of processing queries against a collection of information sources. We make two simplifying assumptions. First, our discussion here is within the framework of relational databases; however, most of our results could be generalized to other kinds of information systems. Second, we take advantage of previous efforts in the area of intensional inconsistency and we assume that all such inconsistencies have been resolved. In its simplest form, the problem we address is formulated as follows.

A query Q is presented to n different instances $d_1, \ldots, d_n$ of the same database scheme D, and is answered by n different answers $q_1, \ldots, q_n$. What is the true answer to Q?

This formulation assumed that the individual databases have identical schemes. A more general framework allows for intensional inconsistencies among the individual schemes, but assumes that they have been resolved by means of a global databases scheme, which consolidates the individual database schemes. This general problem is formulated as follows.

A query Q is presented to a database scheme D that integrates multiple database schemes. In the process of evaluating this query, a related query T is found to be answerable in n individual databases, and is translated to n "equivalent" queries $T_1, \ldots, T_n$. These queries are submitted to the individual databases, and are answered by n different answers $t_1, \ldots, t_n$. What is the true answer to T (and hence Q)?

Our general approach is as follows. We assume that every database D has a hypothetical instance d_0 that correctly represents the real world. A given database instance d is therefore an *approximation* of the real world instance d_0. We then adopt a measure of *goodness* of information. This measure attempts to quantify the *proximity* of database information to real world information. Goodness is assigned to database *views*: for a given database view V, the goodness of V measures the proximity of its extension v in a database instance d to its extension v_0 in the real world instance d_0. As queries are also views, goodness also measures the quality of database answers.

Of course, since the real world instance d_0 is unavailable, the goodness of a given database view v cannot be readily computed. Nevertheless, the goodness of such views can be *estimated* either by external methods such as sampling, or by internal methods such as knowledge discovery.

Our first goal is to estimate the goodness of an arbitrary view (i.e., any answer provided by one of the available database instances). Our approach is to select for each database a set of basic views, to be termed a *goodness basis*, whose goodness will be established by external or internal methods. The goodness of arbitrary views will be *inferred* from the goodness of these basic views. This goal is phrased formally as follows.

Given a set of m views $V_1, \ldots, V_m$ and their respective extensions $v_1, \ldots, v_m$ in a given database instance, and given the respective goodness estimates of these extensions $g_1, \ldots, g_m$,

estimate the goodness g of the extension v in the same database instance of an arbitrary view V.

It may now be assumed that each of the n answers to the query Q has a goodness estimate associated with it. Our final goal is to conclude from these answers a single, possibly new, answer which is the best answer to Q. Formally, this goal is phrased as follows.

Given a set of n answers $q_1, \ldots q_n$ to query Q,[1] obtained from n different database instances, and given their respective goodness estimates $g_1, \ldots, g_n$, find an answer q that has the highest goodness.

In summary, our approach to the integration of inconsistent information can be summarized as follows.

Preparatory Procedure:

P1 Establish a measure of information quality (*goodness measure*).

P2 For each database determine a set of basic views whose goodness will be useful for inferring the goodness of anticipated queries (*goodness basis*).

P3 Estimate the goodness of each view in the goodness basis (and maintain these estimates continuously).

Integration Procedure:

I1 Given a query, decompose it into related queries against the individual databases, and obtain all answers.

I2 For every group of overlapping queries, check whether the individual answers are consistent. If not, then provide the Harmonization Procedure with the intersection query and its individual answers, and obtain an authoritative answer.

I3 Consolidate the individual answers in an answer to the original query.

Harmonization Procedure

H1 Estimate the goodness of each answer from the inconsistent set.

H2 Combine the inconsistent answers in an answer of the highest goodness.

As mentioned earlier, we assume a multi-database system that conforms to the Integration Procedure (i.e., steps I1 and I3), and the focus of this project is the Preparatory Procedure, the detection of inconsistencies (step I2), and the Harmonization Procedure.

[1] In the more general case, $q_1, \ldots, q_n$ are answers to "equivalent" queries $Q_1, \ldots, Q_n$.

Our work is distinguished by several fundamental aspects. First, we address an important issue of information integration that so far been largely neglected: the inconsistency of multiple information sources.

Second, we acknowledge that information sources are rarely perfect. This state of affairs may be conveniently ignored when there is a single source of information, but it must be acknowledged when there are multiple and overlapping sources. Consequently, we treat all information sources as *estimates*.

Third, we require providers of information (or independent "certifiers") to assess the goodness of their information. Such declarations may be regarded as formalization of everyday practices. For example, a library describing its collection as "excellent" in the area of French Classicism and "poor" in the area of Italian Romanticism; or a film guide claiming to include "over 85% of English speaking films, and over 75% of European films"; or a mailing list supplier that boasts of less that 10% error rate.

Finally, our model has attractive uniformity: available databases are assumed to have associated levels of goodness, answers from individual databases are similarly associated with levels of goodness, and multi-database answers are constructed to optimize their level of goodness.

In the next section we provide definitions for essential database concepts, such as a scheme, an instance, a constraint, a view, a query, and an answer. In Section 3 we formalize the concept of a multi-database, and define a multi-database query, a multi-database answer, and answer inconsistency. In Section 4 we define goodness of information and we discuss specific goodness measures. Our treatment is entirely formal, yet our ultimate goal is an operational software tool for resolving answer inconsistencies. In Section 5 we sketch an architecture of a software agent, which we call Harmony, for resolving inconsistent answers in an environment of multiple databases, and we summarize the major research issues currently under investigation.

2 Database

As indicated earlier, our formalization is within the framework of relational databases. Initially, we consider databases that are a single relation, queries that are selection-projection expressions, and a simple type of integrity constraints. A more general treatment will be attempted at later stages of this research.

2.1 Schemes and Instances

Assume a finite set of attributes $D = \{A_1, \ldots A_n\}$, and for each attribute A_i $(i = 1, \ldots, n)$ assume a finite *domain* $dom(A_i)$.

A *relation scheme* R is a non-empty set of attributes $\emptyset \neq R \subseteq D$. In particular, the relation scheme D is called the *database scheme*.

A *tuple* t of a relation scheme R is a partial function that assigns every attribute in R a value from the domain of that attribute. Formally, let $att(t)$ denote the set of attributes on which the tuple t is defined. Then

$$\forall A \in att(t): \quad t(A) \in dom(A)$$

This definition allows tuples that are only partially specified, a feature which is useful when some values are unavailable (unknown or inapplicable).

As an example, consider the attribute set $D = \{A, B, C\}$ and the domains $dom(A) = \{a_1, a_2, a_3, a_4\}$, $dom(B) = \{b_1, b_2\}$, and $dom(C) = \{c_1, c_2, c_3\}$. An example of a tuple on the relation scheme $\{A, C\}$ is the function that maps A to a_1 and C to c_1; it will be denoted $\{A = a_1, C = c_1\}$. Another example is the function that only maps A to a_2; it will be denoted $\{A = a_2\}$.

t' is a *subtuple* of t, denoted $t' \subseteq t$, if $att(t') \subseteq att(t)$ and t and t' agree on the values of the attributes in $att(t')$; i.e., $\forall A \in att(t') : \quad t'(A) = t(A)$. If $att(t') \subset att(t)$, then t' is a *strict* subtuple of t, denoted $t' \subset t$.

A *relation* r on relation scheme R is a finite set of tuples of R, such that no tuple is a strict subtuple of another. A *database instance* d is a relation on the database scheme D.

For example, the following set of tuples constitutes a database instance: $\{ \{A = a_1, B = b_1\}, \{A = a_2, C = c_3\}, \{A = a_3, B = b_2, C = c_3\} \}$. We shall also use the following tabular representation

A	B	C
a_1	b_1	
a_2		c_3
a_3	b_2	c_3

If the tuple $\{A = a_3, C = c_3\}$ were added to this instance, the result would not be an instance, because the new tuple is a subtuple of $\{A = a_3, B = b_2, C = c_3\}$.

2.2 Views and Queries

Assume a set of *variables* $X = \{v_1, \ldots, v_m\}$.

A *view* V of database scheme D is a combination of a function s_V on D and a subset p_V of D. The function assigns every attribute either a value from its domain or a variable from X, with the restriction that two attributes may be assigned the same variable only if they have the same domain. Formally,

$$\forall A \in D: \quad s_V(A) \in dom(A) \cup X$$
$$\forall A_i, A_j \in D: \quad s_V(A_i), s_V(A_j) \in X, \; s_V(A_i) = s_V(A_j) \implies dom(A_i) = dom(Aj)$$

Intuitively, the function s_V specifies selection constraints on d of two kinds: a value a for attribute A is a constraint that is satisfied by tuples that have value a in attribute A; an identical variable for two different attributes A_i and A_j is a constraint that is satisfied by tuples that have the same value in attributes A_i and A_j. The subset p_V specifies the attributes on which the selected tuples are projected.

The *extension* v of view $V = (s_V, p_V)$ in a database instance d is a relation on the database scheme p_V as follows: $t \in v$ if and only if there exists $t' \in d$ such that

$$\begin{array}{l} \forall A \in p_V : \;\; t'(A) = t(A) \\ \forall A \in D : \;\; s_V(A) \in X \implies \exists a' \in dom(A) : \;\; t'(A) = a' \\ \forall A \in D : \;\; s_V(A) \in dom(A) \implies t'(A) = s_V(A) \\ \forall A_i, A_j \in D : \;\; s_V(A_i) = s_V(A_j) \implies t'(A_i) = t'(A_j) \end{array}$$

Consider again the database scheme $D = \{A, B, C\}$ and the variables $\{x, y, z\}$. Together, the function that maps A to x, B to y, and C to c_3, and the subset of attributes $\{A, B\}$ describe a view. This view selects the tuples for which $C = c_3$ and projects them on attributes A and B. We shall also use the notation $project_{A,B}select_{C=c_3}(D)$. Its extension in the previous database instance (using the tabular representation) is

A	B
a_2	
a_3	b_2

Given views V' and V of database scheme D, V' is a *subview* of V, denoted $V' \subseteq V$, if

1. $p_{V'} \subseteq p_V$
2. $\forall A \in D, \;\; \forall c \in dom(A) : \;\; s_V(A) = c \implies s_{V'}(A) = c$
3. $\forall A_i, A_j \in D : \;\; s_V(A_i) = s_V(A_j) \implies s_{V'}(A_i) = s_{V'}(A_j)$

Intuitively, one view is a subview of another, if the former's selection constraint is more restrictive than the latter's, and the former's projected attributes are contained in the latter's. Assume $V' \subseteq V$ and let v' and v be their respective extensions in some database instance d. Then every tuple of v' is a subtuple of some tuple in v.

Views V_1 and V_2 of database scheme D are *overlapping*, if

1. $p_{V_1} \cap p_{V_2} \neq \emptyset$
2. $\forall A \in D \;\; \forall c_1, c_2 \in dom(A) : \;\; s_{V_1}(A) = c_1 \;\wedge\; s_{V_2}(A) = c_2 \implies c_1 = c_2$

Intuitively, two views are overlapping, if their selection constraints are not contradictory, and their projected attributes have a non-empty intersection.

Assume V_1 and V_2 are two overlapping views of a database scheme D. The *intersection* of V_1 and V_2, denoted $V_1 \wedge V_2$, is the view obtained by conjoining their selection constraints and intersecting their projection attributes. The intersection of two views is a subview of each of the original views: $V_1 \wedge V_2 \subseteq V_1$ and $V_1 \wedge V_2 \subseteq V_2$. It can be verified that the extension of the intersection view $V_1 \wedge V_2$ is equal to the intersection of the extensions of V_1 and V_2.

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

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 [8].

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

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

$$I_1: \quad project_{Name} select_{(Level=junior) \wedge (Title=manager)}(Emp)$$

This 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 that junior employees may not be managers.[2]

Assume a constraint I and a view V on scheme D. I is *applicable* to V, if I is a subview of V. Let v be the extension of V in some database instance. Then v satisfies the constraint I.

For example, with the previous scheme consider the view

$$V_1: \quad project_{Name,Level} select_{Title=manager}(Emp)$$

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 scheme D, and assume that I and Q are overlapping views. Let $I' = I \wedge Q$. Then I' is a constraint applicable to Q. Given a database scheme D, a set C of integrity constraints on scheme D, and a query Q on scheme D, the set of constraints obtained by intersecting Q with every overlapping constraint in C is called the *reduction* of C to Q, and is denoted C_Q. Intuitively, the answer q to Q in every instance d satisfies the constraints in C_Q.

[2] 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.

For example, with the previous scheme consider the view

$$V_2: \quad project_{Name}\, select_{Supervisor=jones}(Emp)$$

The intersection of the constraint I_1 with the view V_2 is given by

$$I_2: \quad project_{Name}\, select_{(Level=junior)\wedge(Title=manager)\wedge(Supervisor=jones)}(Emp)$$

The constraint I_2 (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* discussed in [3].

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 on the scheme D that satisfies all the 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 instance d. We shall use the term *model* to refer collectively to the scheme and the constraints.

3 Multi-database

There have been many individual approaches to the problem of integrating multiple databases. In this section we provide a formal framework that encompasses many of these individual approaches. Our framework extends the commonly accepted definitions for a single database environment (a version of which we gave in the previous section) to an environment of multiple databases. Specifically, we formalize the concept a multi-database, and define a multi-database query, a multi-database answer, and answer inconsistency.

3.1 View and Constraint Equivalence

Consider a database (D, C, d). Let D' be a database scheme that is a view of D. [3] The view that transforms D to D' also determines a set of integrity constraints C' and a database instance d'. Altogether, this view determines a *derivative* database (D', C', d').

Consider a database (D, C, d), and let (D_1, C_1, d_1) and (D_2, C_2, d_2) be two such derivative databases. These three databases are all 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

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

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.[4]

3.2 Model and Instance Assumptions

To define equivalence among views and constraints in different databases, we assumed that the databases were all derived from a "universal" database. In general, 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 [6], although their purpose here is quite different.

The Model Consistency Assumption. All database models (schemes and constraints) are derivatives of the scheme and constraints of the real world model. 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 individual database models are reconcilable. We *adopt* this assumption.

The Instance Consistency Assumption. All database instances are derivatives of 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 individual database instances are reconcilable. We *do not adopt* this assumption.

This approach allows us to ignore the possibility of irreconcilable intensional inconsistencies, and to assume that all manifestations of semantic heterogeneity can be reconciled with techniques established by previous research. On the other hand, we acknowledge that, in reality, database instances are only approximations of the true information, and information stored in different databases is not necessarily consistent.

Consequently, we treat all statements of view and constraint equivalence as *constraints*: conditions that should be satisfied in any database that is consistent with reality, but are not necessarily satisfied by the database instances at hand. In other words, if view V_1 of one database and view V_2 of another database are declared to be equivalent, but their extensions are different, then at least one of the database instances is in error.

[4] Again, the expressivity of view or constraint equivalence is limited only by the expressive power of the query language.

3.3 Model Mapping

We assumed that all differences among models (schemes and constraints) are reconcilable. The reconciliation between two different models is achieved by means of mappings.

Assume two database schemes D_1 and D_2. 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 $V_{i,2}$. Let C_1 be a set of constraints on scheme D_1 and let C_2 be a set of constraints on scheme D_2. A *constraint mapping* (C_1, C_2) is a collection of constraint pairs $(I_{i,1}, I_{i,2})$ $(i = 1, \ldots, m)$, where each $I_{i,1}$ is a constraint on D_1, each $I_{i,2}$ is a constraint on D_2, and $I_{i,1}$ is equivalent to $I_{i,2}$.

As an example, the equivalence of attribute A in scheme D_1 and attribute B in scheme D_2 is indicated by the view pair

$$(\ project_A(D_1), \ project_B(D_2) \ )$$

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

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

The constraint that Jones does not supervise any managers is expressed differently in each scheme, as indicated by the constraint pair

$$(\ project_{Name} select_{(Title=manager) \wedge (Supervisor=jones)}(Emp), \ project_{Ename} select_{Sup=jones}(Manager) \ )$$

Finally, a multi-database is

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

This definition may be considered a formalization of *virtual databases* defined in [7]. Scheme mapping may be considered an abstraction of the different solutions that have been advanced to the task of relating global schemes to individual schemes (e.g., [7, 1, 13]).

Note that the mappings from D and C to the individual databases are not necessarily "total"; i.e., not all views and constraints on D are expressible in every individual database

(and even if they are expressible, there is no guarantee that they are mapped). Similarly, these mappings are not necessarily "onto"; i.e., the individual databases may include views or constraints that are not expressible in D (and even if they are expressible, there is no guarantee that they are mapped).

Recall that we do not assume that the individual 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 multi-database does not imply that the extensions of V in the individual databases are identical. Rather, it implies that they *should be* identical. Similarly, the inclusion of constraint pairs (I, I_1) and (I, I_2) in two constraint mappings does not imply that they are satisfied simultaneously.

3.4 Multi-database Queries

When a multi-database receives a query Q, the scheme mappings are used to locate related queries (called *decomposition queries*) that can be processed in the individual databases. Several, possibly overlapping, such queries $Q_1, \ldots, Q_m$ are submitted to the individual databases (more precisely, the *mappings* of these queries are submitted). The answers $q_1, \ldots, q_m$ returned from these databases are then assembled into an extension q of the target query Q. This extension is delivered to the user.[5] Hence, like a single database, a multi-database is a function from queries to answers: given a query Q on scheme D, it computes its answer q in the databases $(D_1, C_1, d_1), \ldots, (D_n, C_n, d_n)$.

As a simple example, assume databases $D = (A, B, C, D, E)$, $D_1 = (A, B, C)$ and $D_2 = (A, D, E)$, and assume the obvious scheme mappings between D and D_1 and between D and D_2. A multi-database query such as $Q = (B, D)$ will be decomposed into $Q_1 = (A, B)$ which will be submitted to the first database and $Q_2 = (A, D)$ which will be submitted to the second database. Their answers q_1 and q_2 will be integrated (by a natural join) in an answer q to the original query Q. Note that the decomposition queries Q_1 and Q_2 are "related" to Q, but are not strictly subqueries of Q.

Each individual answer satisfies the integrity constraints applicable to it in its individual database. The integrated answer should be defined so it satisfies the constraints applicable to it in the multi-database. Consider, for example, two databases with the same scheme *Emp = (Name, Level, Salary)* and the constraint that employees at the same level must earn the same salary. Each individual answer will satisfy this constraint, yet two employees, one included in one answer and the other included in the second answer, could have identical levels but different salaries.

Assume two decomposition queries of Q: Q_1 is mapped to S_1 which is submitted to (D_1, C_1, d_1), and Q_2 is mapped to S_2, which is submitted to (D_2, C_2, d_2). Let q_1 and q_2 denote their respective answers. Q_1 and Q_2 are both queries on the scheme D. When Q_1

[5] The decomposition of Q into $Q_1, \ldots, Q_m$ and the composition of $q_1, \ldots, q_m$ into q is the central part of any system that integrates information from multiple sources. We assume that such a method exists.

and Q_2 overlap we construct the intersection view $T = Q_1 \wedge Q_2$. T is a view of the scheme D and a subview of both Q_1 and Q_2, and therefore can be answered from both q_1 and q_2. Let t_1 and t_2 denote the respective answers.

In summary, a query T on the scheme D (a query related to the original query Q) has two answers t_1 and t_2. In general, a query T on the scheme D may have n answers $t_1, \ldots, t_n$. If $t_1, \ldots, t_n$ are are not identical, then we must ask: *what is the true answer to T (and hence Q)?*

Note that different answers are not necessarily inconsistent. We shall assume that each answer is accompanied by a *claim of goodness* that estimates its relationship to the true answer. It is then possible that different answers may be harmonized into a single answer that satisfies the goodness claims of the individual answers.

As an intuitive example, assume a query Q is answered by $q_1 = \{1, 2\}$ and $q_2 = \{1, 3, 4\}$, and assume that q_1 is "half sound and half complete" and q_2 is "complete and two-thirds sound". Any two element answer that contains either 1 or 2 will satisfy the goodness claim of q_1. Either $\{1, 3\}$, $\{1, 4\}$ or $\{3, 4\}$ will satisfy the goodness claim of q_2. Therefore, two answers, $\{1, 3\}$ and $\{1, 4\}$, are within the goodness claims of both q_1 and q_2.

Different answers are *inconsistent*, if there is no single answer that meets their goodness claims. As a trivial example, different answers that claim to be perfect are inconsistent. Goodness of answers is the subject of the next section.

4 Goodness of Answers

4.1 Soundness, Completeness and Perfectness of Answers

As explained in the introduction, we assume the existence of a hypothetical database instance d_0 that captures perfectly that portion of the real world which is modeled by the database scheme D (the *perfect* or *true* database). In addition, we assume one or more actual database instances d_i ($i \geq 1$) that are *approximations* of the perfect database d_0.

Given a query Q, we denote by q_0 its answer in the perfect database d_0 (the *perfect* or *true* answer to Q), and we denote by q_i its answer in the actual database d_i. Thus, the answers q_i are *approximations* of the perfect answer q_0.

Consider query Q, its perfect answer q_0, and an approximation q. If $q \supseteq q_0$, then q is a *complete* answer. If $q \subseteq q_0$, then q is a *sound* answer. Obviously, an answer which is sound and complete answer is the perfect answer.

The concepts of a perfect instance and soundness and completeness of database answers were first introduced in [8].

4.2 Goodness of Answers

We wish to assign each answer a value that denotes how well it approximates the perfect answer. We shall term this value the *goodness* of the answer. We require that the goodness of each answer be a value between 0 and 1, that the goodness of the perfect answer be 1, and that the goodness of answers that are entirely disjoint from the perfect answer be 0. Formally, a *goodness measure* is a function g on the set of all possible answers

$$\begin{aligned} &\forall q: \quad g(q) \in [0,1] \\ &\forall q: \quad q \cap q_0 = \emptyset \implies g(q) = 0 \\ &g(q_0) = 1 \end{aligned}$$

A simple approach to goodness is to compare the *number of tuples* in q and q_0. Let $|q|$ denote the number of tuples in q. Then

$$\frac{|q \cap q_0|}{|q|}$$

expresses the proportion of the database answer that appears in the true answer. Hence, it is a measure of the *soundness* of q. Similarly,

$$\frac{|q \cap q_0|}{|q_0|}$$

expresses the proportion of the true answer that appears in the database answer. Hence, it is a measure of the *completeness* of q.

It is easy to verify that soundness and completeness satisfy all the requirements of a goodness measure. [6] Soundness and completeness are very similar to *precision* and *recall* in information retrieval [11].

While these appear to be the only natural measures of goodness that correspond to the degree of soundness and to the degree of completeness (in both cases: the degree to which one set is contained in another), there appear to be several possible goodness measures that correspond to the degree of *perfectness* (the degree to which two sets are identical).

One measure that expresses the degree of "agreement" or "overlap" between two answers is the Jaccard measure [11]

$$\frac{|q \cap q_0|}{|q \cup q_0|}$$

Another possible measure that satisfies all the requirements of a goodness measure is the *dice* measure [11]

$$\frac{2|q \cap q_0|}{|q| + |q_0|}$$

[6] When q is empty, soundness is $0/0$. If q_0 is also empty then soundness is defined to be 1; otherwise it is defined to be 0. Similarly for completeness, when q_0 is empty.

4.2.1 More Precise Estimation

As defined, a tuple $t \in q$ is either correct (identical to a tuple in q_0), or incorrect. Partial correctness is not defined. To consider partial correctness we decompose the information encapsulated in a tuple into more elementary components.

Let q be an answer to query Q. The *decomposition* of q, denoted $dec(q)$, is the set of all subtuples of every tuple of q:

$$dec(q) = \{t' \mid \exists t \in q : \ t' \subseteq t\}$$

The measures of soundness, completeness and perfectness can now be modified to calculate the number of tuples in the decomposition of answers. For example, a measure of the perfectness of q is given by

$$\frac{|dec(q) \cap dec(q_0)|}{|dec(q) \cup dec(q_0)|}$$

5 Conclusion

In this final section we sketch an architecture of a software agent for resolving inconsistent answers in an environment of multiple databases, and we summarize the major research issues currently under investigation.

5.1 Harmony: An Inconsistency Resolver

A possible implementation of the framework that we described in this paper is by using a software agent that would be engaged by intelligent information integrating systems (I^3 systems) whenever they encounter inconsistencies. So that it is deployable by different I^3 systems, this agent, which we call Harmony, will assume that the behavior of I^3 systems is abstracted by the concept of multi-database that was defined formally in Section 3.

To resolve an inconsistency Harmony must be provided with the following information.

1. The definition of the problem query T and the constraints C_T that its answer must satisfy.
2. The individual answers t_i and their respective goodness estimates g_i.

If a goodness estimate g_i cannot be provided, then Harmony must estimate it. In such a case it must be provided with

1. The scheme D_i and the constraints C_i.

2. The goodness basis of database (D_i, C_i, d_i); i.e., a set of view definitions and their goodness estimates.

3. The ability to submit queries against (D_i, C_i, d_i); e.g., to materialize any of the goodness views.

Eventually, Harmony will provide the I^3 system with a single answer t and an associated goodness value g.

5.2 Research Issues

In this report we explored the problem of extensional inconsistencies in an environment of multiple information sources. We formalized a framework in which the problem can be investigated, and we illustrated a procedure for resolving extensional inconsistencies, using a "free agent" called Harmony. This agent would be engaged by any system whose abstract behavior subscribes to that of a multi-database.

Three core research issues are currently under investigation. (1) How to select and maintain a goodness basis for each individual database. (2) How to infer the goodness of an arbitrary database view (query) from the goodness basis of that database. (3) Given different answers to the same query (obtained from different databases) and their respective goodness estimates, how to integrate these answers in an answer of the highest goodness.

Other issues that need to be investigated are concerned with generalizations of our assumptions; in particular, databases with multiple relations, more powerful queries and constraints, and information systems other than relational databases.

As explained earlier, we assumed a method exists for processing multi-database queries, and our research focus is the resolution of inconsistencies encountered during this process. Thus, we only provided the formal framework for representing the semantic information needed to decompose queries (scheme mappings), but did not describe any method for performing the actual decomposition. Yet, it is possible that knowledge of information goodness can be used to guide the process of query decomposition. This possibility requires further research.

Acknowledgement: The following individuals made important contributions to this report: Alessandro D'Atri, Igor Rakov and Alex Brodsky.

References

[1] Y. Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. Technical Report ISI-RR-93-308, USC Information Sciences Institute, March 1993.

[2] 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.

[3] 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.

[4] R. ElMasri and G. Wiederhold. Data model integration using the structural model. In *Proceedings of ACM-SIGMOD International Conference on Management of Data* (Boston, Massachusetts, May 29–June 1), pages 319–326. ACM, New York, New York, 1979.

[5] 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.

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

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

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

[9] S. Navathe and S. Gadgil. A methodology for view integration in logical database design. In *Proceedings of the Eighth International Conference on Very Large Data Bases* (Mexico City, Mexico, September 8–10), pages 142–152. Morgan Kaufmann, Los Altos, California, 1982.

[10] X. Qian. Semantic interopration via intelligent mediation. In *Proceedings of the Third International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems*, pages 228–231, April 1993.

[11] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, New York, 1983.

[12] A. Sheth and J. A. Larson. A tool for integrating conceptual schemes and user views. In *Proceedings of the IEEE Computer Society Fourth International Conference on Data Engineering* (Los Angeles, California, February 1–5), pages 176–183. IEEE Computer Society, Washington, DC, 1988.

[13] J. M. Smith, P. A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. W. T. Lin, and E. Wong. Multibase—integrating heterogeneous distributed database systems. In *Proceedings of AFIPS National Computer Conference* (Chicago, Illinois, May 4–7), pages 487–499. AFIPS Press, Arlington, Virginia, 1981.

[14] 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.