WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

A METADATA APPROACH TO RESOLVING SEMANTIC CONFLICTS

Michael Siegel
Stuart Madnick

February 1991

WP # 3252-91-MSA
WP # CIS-91-20

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

A METADATA APPROACH TO RESOLVING SEMANTIC CONFLICTS

Michael Siegel
Stuart Madnick

February 1991

WP # 3252-91-MSA
WP # CIS-91-20

A Metadata Approach to Resolving Semantic Conflicts

Michael Siegel
Stuart E. Madnick
Sloan School of Management, E53-323
Massachusetts Institute of Technology
Cambridge, MA 02139
msiegel@sloan.mit.edu

Abstract

Semantic reconciliation is an important step in determining logical connectivity between a data source (database) and a data receiver (application). Semantic reconciliation is used to determine if the semantics of the data provided by the source is meaningful to the receiver. In this paper we describe a rule-based approach to semantic specification and demonstrate how this specification can be used to establish semantic agreement between a source and receiver. We describe query processing techniques that use these specifications along with conversion routines and query modification to guarantee correct data semantics. In addition, this work examines the effect of changing data semantics. These changes may occur at the source of the data or they may be changes in the specifications of the data semantics for the application. Methods are described for detecting these changes and for determining if the database can continue to supply meaningful data to the application. Though described in terms of the source-receiver model, these techniques can also be used for semantic reconciliation and schema integration for multidatabase systems.

Keywords[data dictionaries, metadata, query modification, schema integration, semantic conflicts]

1 Introduction

With the development of complex information systems, the need for the integration of heterogeneous information systems, and the availability of numerous online computer data sources, it has become increasingly important that methods be developed that consider the meaning of the data used in these systems. For example, it is important that an application requiring financial data in French francs does not receive data from a source that reports in another currency. This problem is further complicated by the fact that the source meaning may change at any time; a source that once supplied financial data in French francs might decide to change to reporting that data in European Currency Units (ECUs).

To deal with this problem, these systems must have the ability to represent data semantics and detect and automatically resolve conflicts in data semantics. At best, present systems permit an application to examine the data type definitions in the database schema, thus allowing for type checking within the application. But this limited capability does not allow a system to represent and examine detailed data semantics nor handle changing semantics.

McCarthy [McC82,McC84] describes the importance of detailed data description or *metadata* systems for use in large statistical databases. Another system, the *Information Resource Dictionary System* (IRDS) [GK88,Law88], allows a user to develop a metadata dictionary using an entity-relationship model. These systems provide the ability to access and manipulate metadata in a manner similar to that used for data. However, these systems do not include a well-defined methodology for utilizing metadata for semantic reconciliation, the identification and resolution of semantic conflicts.

This research examines the specification and use of metadata in a simple *source-receiver* model. The *source* (database) supplies data used by the *receiver* (application). The source and receiver may be at the same physical location, as in a local database management system accessed by an application program, or the source may be in a different location, such as an online data service.

We describe a rule-based representation for both the database semantic specification and the application's *semantic view* of the data. A well-defined model for data semantic representation allows for comparisons between database and application rule sets to determine whether the database can supply meaningful data to the application. Initially, we examine query processing strategies that can be used to determine if the application query can be correctly answered using the semantic definitions of the data at the source. Then, we examine query processing techniques for adding restrictions to an application query to guarantee correct data semantics.

The methods proposed for semantic reconciliation allow for changes in data semantics in the database or changes in the application's data semantic requirements. These methods can be used to track changes automatically and determine, as a result of any changes, if the database can still supply meaningful data. Additionally, we show that in some instances these methods can be used to resolve semantic conflicts between the database and the receiver, thus allowing the database to supply meaningful data.

This paper is organized as follows. In the next section we examine related work in the area of metadata representation. In Section 3 we present examples of the problems that can occur in the source-receiver model when methods for semantic reconciliation are not available. We introduce a model for defining data semantics for use in identifying and resolving semantic conflicts. In Section 4 we describe the use of metadata in semantic reconciliation. We define an architecture for a system that can represent and manipulate data semantics. We then describe the process of comparing metadata between systems (i.e., an application and a database) and present methods for application query processing that can be used to automatically determine if a database can provide an application with meaningful data. Then, in Section 4.3.3 we consider the use and effect of query processing techniques that modify the application query to guarantee semantic correctness. In Section 5 we examine the use of semantic reconciliation in a dynamic system environment where changes occur in the application or database semantics. Finally, in Section 6 we present our conclusions and describe areas of future research including the the use of metadata and semantic reconciliation and schema integration in multidatabase systems.

2 Metadata

Metadata refers to data about the meaning, content, organization, or purpose of data. Metadata may be as simple as a relational schema or as complicated as information describing the source, derivation, units, accuracy, and history of individual data items.

In [McC82], McCarthy describes a metadata representation and manipulation language where the metadata is part of the data files. The representation allows for the inclusion of a wide range of metadata accessible through a set of specially defined operators. In [McC87] he demonstrates the use of metadata in a Material Properties Database. The development of the *Information Resource Dictionary System* (IRDS) for handling metadata is described in [GK88,Law88]. The IRDS allows the user to develop an entity-relationship model description of the metadata. The IRDS includes a set of primitive entities and relationships, and a set of operations to build new entities and relationships for describing metadata. [GSdB88] describes additional knowledge-based representations for metadata. However, these approaches do not include a well-defined methodology for utilizing this metadata for semantic reconciliation. [YSDK90] describes the use of concept hierarchies for comparing attributes from different schemas. However, practical means for defining comparable concept hierarchies are not discussed and these methods deal with attribute comparisons not data comparisons.

It is important to provide a representation that is rich enough to describe the significant data semantics and can be used in methods to identify and reconcile semantic heterogeneities. We are interested in a representation that can be used for defining both the application and database semantics. We intend to use this metadata for semantic reconciliation; more precisely, we intend to use metadata to resolve the following questions in the source-receiver model:

1. Can the database provide data that is semantically meaningful to the application?
2. Is the application affected by a change in the database semantics? (or a change in its own data semantics requirements?)

In the next section we describe a sample database and application and consider the problems that occur when the semantics of the data provided by the database are not the same as those required by the application. Then, we introduce a restricted rule-based representation for data semantics and define a model for semantic representation that will permit the comparison of semantics between systems.

3 Providing Metadata for Semantic Reconciliation

As a more detailed example showing the need for semantic reconciliation, consider a data source that provides the trade price for a variety of financial instruments. The schema of the relation containing this data is shown in Figure 1 along with two sample records. Each record contains the type and name of the instrument being traded, the exchange that the instrument was traded on, and the trade price. It should be noted that although we are using the notion of data source as a database, the concepts apply equally well to a *data stream* such as a *stock ticker*, where records are being continuously transmitted and the application selects those of interest to it.

A query that requests the trade price of Telecom SP will return the value 1107.25. Even in this simple relation, the *natural* interpretation of this value might not provide a complete understanding

Instrument_Type	Instrument_Name	Exchange	Trade_Price
Equity	IBM	nyse	115.25
Equity	Telecom SP	madrid	1107.25

Figure 1: The FINANCE Relation

of the data. For example, this relation does not report all trade prices in US currency. Rather, prices are given in the currency of the exchange. The trade price for most equities represents the latest trade price except for equities traded on the Madrid Stock Exchange where trade price represents the latest nominal price. Because of these semantic complications, there should be a means for representation of and access to both the trade price value and its associated metadata. Then, given an *application's semantic view* (i.e., data semantics specification) methods can be provided to determine if the semantics associated with the data are those expected by the application.

One way to represent this metadata is to extend the traditional database schema definition to include virtual fields. For example, the relation in Figure 1 could be extended to include attributes such as *Trade Price Status* and *Currency*. However, it is at this point our intention to make the representation and methods for semantic reconciliation transparent to the user. We have chosen a more flexible representation of data semantics that is data model independent. Allowing users access to the metadata will simply involve mapping the representation onto the appropriate data model. Separating the metadata from the data also has the advantage that the metadata system is non-intrusive in the sense that it does not require the data source to be changed which may not be feasible especially if the data source is a separate organization.

In the next section we describe a rule-based representation to associate (i.e., tag data with) metadata with a given attribute. Through an examination of a number of applications we have determined that this representation can be used to describe much of the data semantics in existing databases while also being useful in defining the application's *semantic view* of the data.

3.1 A Model for Data Semantics

In this section we present a model that provides both a representation of data semantics and the range of applicability of our methods for semantic reconciliation. We begin by defining the *semantic domain* of an attribute T as the set of attributes used to define the semantics of T and note this as

$\text{sem}(T) = <Y_1, Y_2, Y_3, ..., Y_n>$ where each Y_i is an attribute.

For each value t in the domain of T the semantics of that value can be defined in terms of the semantic domain as

$\text{sem}(t) = <y_1, y_2, y_3, ...y_n>$ where $y_i \epsilon \text{domain}(Y_i)$.

As an example, we may think of the semantic domain of the Trade_Price attribute in terms of the status and currency of the trade price. The semantic domain is then defined as

sem(Trade_Price) = <Trade_Price_Status, Currency>

and the semantics of a particular trade price may be defined as

sem(115.25) = < latest_trade_price, US_dollars>

domain(Instrument_Type) = <equity, future>.
domain(Exchange) = <nyse, madrid>.
domain(Currency) = <US dollars, French francs, pesetas>.
domain(Trade_Price_Status) = <latest_trade_price, latest_nominal_price>.

Figure 2: Examples of Primitive Attribute Domains

where the value 115.25 represents the latest trade price in US dollars.

The basis for our model of data semantics is the assignability of values to the semantic domain. An attribute is *semantically assignable* if there is some function that can determine sem(t) for each $t \epsilon$domain(T). The *assignment domain* of attribute T is defined as

assign(T) = $<X_1, X_2, X_3, ..., X_n>$ where each X_i is an attribute.

The assignment domain for a particular value t in the domain of T is defined as

assign(t) = $<x_1, x_2, x_3, ...x_n>$ and $x_i \epsilon$domain(X_i).

As an example of semantic assignability consider the following assignment and semantic domains:

sem(Trade_Price) = <Trade_Price_Status, Currency>

assign(Trade_Price) = <Instrument_Type, Exchange>

We want some function F that maps values in the assignment domain to values in the semantic domain:

F<Instrument_Type, Exchange> $\Rightarrow$ <Trade_Price_Status, Currency>

Then, for a given trade price, the instrument type being traded and the exchange that it is traded on determine the status and currency of the trade price.

Different classes of semantic assignability exist. An attribute T is *trivially assignable* if for each $t \epsilon$domain(T), sem(t) is constant. That is, the assignment domain is not material in determining the semantic domain, only the value t has any effect on the semantic domain. We say that the attribute T is *primitive* if T is trivially assignable and for all $Y_i \epsilon$sem(T) Y_i is primitive. for example, primitive attributes might include Instrument_Type, Exchange, Currency and Trade_Price_Status as shown in Figure 2. Then the semantics of a value for Currency, say, US dollars, is well-defined among all systems that share this primitive concept. In Section 4.1 we describe the establishment of primitive concepts for use in these systems.

The existence of primitive attributes provides a *common language* by which the semantics of other attributes can be defined. We say that attribute T is *semantically definable* either it is primitive or it is semantically assignable and for all $X_i \epsilon$assign(T), X_i is semantically definable and for all $Y_i \epsilon$sem(T), Y_i is semantically definable.

In this paper we use sets of rules as procedures for assigning semantics to each semantically definable attribute. A semantic assignment rule for attribute T has the following form:

$$C_1(X_1), C_2(X_2), \ldots, C_i(X_i) \Rightarrow C_{i+1}(Y_1), C_{i+2}(Y_2), \ldots, C_n(Y_n)$$

$C_1, C_2, \ldots, C_i$ are constraints on the attributes $X_1, X_2, \ldots, X_i \epsilon \text{assign}(T)$ and $C_{i+1}, C_{i+2}, \ldots, C_n$ are constraints on the attributes $Y_1, Y_2, \ldots, Y_n \epsilon \text{sem}(T)$.

Examples of rules that might be used to define the database semantic specification for the Trade_Price attribute are shown in Figure 3. The first rule says that if an instrument is an equity traded on the Madrid Stock Exchange then the trade price is reported as the latest trade price in pesetas. In the next section we show how this representation can be used in semantic reconciliation.

assign(Trade_Price) = <Instrument_Type, Exchange>
sem(Trade_Price) = < Trade_Price_Status, Currency>

1. Instrument_Type = "equity" and Exchange = "madrid" ⇒
   Trade_Price_Status := "latest_nominal_price" and Currency := "pesetas"
2. Instrument_Type = "equity" and Exchange = "nyse" ⇒
   Trade_Price_Status := "latest_trade_price" and Currency := "US dollars"
3. Instrument_Type = "future" ⇒
   Trade_Price_Status := "latest_closing_price" and Currency := "US dollars"

Figure 3: Database Semantic Rules for Trade_Price

4 Using Metadata for Semantic Reconciliation

Figure 4 shows the proposed architecture for a system that uses metadata for semantic reconciliation. The *database metadata dictionary* (DMD) defines the semantic and assignment domains for each attribute and the set of rules that define the semantic assignments for each of these attributes. The *application semantic view* (ASV) contains the application's definition of the semantic and assignment domain and the set of rules defining the semantic requirements for the data. While a conventional database view definition defines the application's structural view of the database, the ASV contains the complete specification of the semantic requirements for the application. The ASV acts as the semantic import schema for the application and the DMD as the semantic export schema for the database [HM85]. The *metadata manager* creates and maintains data on the results from comparisons between the semantic specifications in the ASV and the DMD. It also defines the location of available conversion routines for resolving semantic conflicts (Section 4.2.1).

The rules shown in Figure 3 will act as an example DMD. Other attributes in the example relation (Figure 1) are primitive and thus do not require semantic assignment rules. An example of an ASV is shown in Figure 5. The specification contains two rules. The antecedents of these rules define the domain for values of the Trade_Price attribute based on values of the assignment domain. The first rule limits the domain of interest to equities traded on the Madrid Stock Exchange. Trade_Price values with this assignment domain are to be reported (i.e., values of the semantic domain) as the latest nominal price in pesetas. The second rule limits the domain of interest to instruments traded on the *nyse* where Trade_Price values are to be reported as the latest trade price in US dollars. Thus the total domain of interest of the application is limited to any instrument traded on the *nyse* or any *equity* traded on the *madrid* exchange.

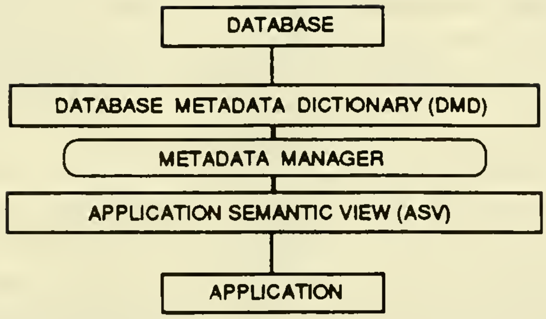

Figure 4: Systems Architecture Using Metadata

assign(Trade_Price) = <Instrument, Instrument_Type, Exchange>
sem(Trade_Price) = < Trade_Price_Status, Currency>

1. Instrument_Type = "equity" and Exchange = "madrid" ⇒
   Trade_Price_Status := 'latest_nominal_price' and Currency := 'pesetas'

2. Exchange = "nyse" ⇒
   Trade_Price_Status := 'latest_trade_price' and Currency := 'US dollars'

Figure 5: Application Semantic View (ASV) for Trade_Price

To decide whether a database can supply meaningful data to an application we must determine if the rules in the DMD guarantee the data semantics specified by the rules in the ASV. In Section 4.2 we describe methods for comparing these rule sets. The results of these comparisons are used in query processing to guarantee semantically meaningful solutions. Before we present these methods we describe restrictions on the DMD and ASV that allow for comparison of these rule sets.

4.1 Restrictions on the Semantic Representation

So that data semantics can be compared between systems (e.g., an application and a database) they must share some common language [ML90]. Data standardization is one method of imposing common language requirements but this method is intrusive on the individual systems. Rather than impose standards on the data we require that these systems share primitive attributes which define a base vocabulary, terminology limited to a unique interpretation inside of the enterprise domain of discourse. Any system in the enterprise can use this base vocabulary to develop rules describing the semantics of semantically definable attributes. Terminology outside of this common language must either be converted to the common language or remain non-comparable, making semantic reconciliation undecidable.

The question remains how practical is it to define such a language and to require that metadata definitions adhere to specifications of the language. A first reaction to this question might be that this is no different than data standardization. Many of the problems of implementing data standards over autonomous organization also exist in establishing and enforcing common language constraints. However, it is intrusive to expect a data source to make data comply to a specific external organization's standards especially when that data may be used by any number of different external organizations while it is non-intrusive on the data operations to require that the source supply metadata based on a shared vocabulary. Methods can be established that permit the evolution of the shared vocabulary as required by changes in data semantics.

In addition to sharing primitive attributes, we require that the assignment and semantic domain of an attribute defined in the ASV be a subset of the assignment and semantic domain for that attribute in the DMD. In the case of our examples (Figures 3 and 5), the assignment and semantic domains of the ASV must be subsets of the assignment and semantic domains of the Trade_Price attribute as defined in DMD. As described in the next section, this requirement facilitates the comparison of the procedures for semantic assignment in the ASV and the DMD. Present research efforts are considering less restricted relationships between the semantic and assignment domains of the ASV and the DMD.

4.2 Comparing Application and Database Semantic Specifications

Prior to the application requesting data from the database the metadata management system must compare the rules in the ASV to those for the same attribute in the DMD. The purpose of these comparisons is to determine for each attribute requested by the application whether the database can deliver semantically meaningful data. Later, in Section 4.3.3 we examine how these comparisons can be used to determine additional constraints that might be used to guarantee correctness.

The rule set comparison begins by selecting a single attribute whose semantics are specified in the ASV. For each rule in the ASV that restricts the semantic domain of that attribute we need to determine those rules in the DMD with *matching* antecedents. The basic types of comparisons

ASV Rule Number	1	2	
DMD Rule Number	1	2	3
Comparison Type	2	1	1
Application Constraint	Instrument_Type = "equity" Exchange = "madrid"	Exchange = "nyse"	Exchange = "nyse"
Database Constraint	Instrument_Type = "equity" Exchange = "madrid"	Instrument_Type = "equity" Exchange = "nyse"	Instrument_Type = "future" Exchange = "nyse"
ASV Semantic Assignment	Currency = "pesetas" Trade_Price_Status = "latest_trade_price"	Currency = "US dollars" Trade_Price_Status = "latest_nominal_price"	Currency = "US dollars" Trade_Price_Status = "latest_trade_price"
DMD Semantic Assignment	Currency = "pesetas" Trade_Price_Status = "latest_trade_price"	Currency = "US dollars" Trade_Price_Status = "latest_nominal_price"	Currency = "US dollars" Trade_Price_Status = "latest_closing_price"
Semantic Equivalence	Yes	Yes	No

Table 1: Comparisons of ASV and DMD for the Trade Price Attribute

between rule antecedents are defined in Figure 6. The type of comparison is determined by the relationship between constraints in the antecedent of the rules. There are only four possible comparisons types based on this relationship: *subset, superset, overlaps, and disjoint.* As an example, two rules are said to *overlap* if there is at least one common attribute in the antecedents of the rules and there are other attributes that are unique to each of the rules. A match occurs whenever constraints for the overlapping attributes are related through implication. There is a match if the constraint for the ASV rule *implies* the constraint for the DMD rule (e.g., salary > 50K *implies* salary >30K). In this case the DMD rule is more general but still applies to the application semantic view of the data. Alternatively, the constraint in the DMD rule may *imply* the constraint in the ASV rule. There is still a match because the DMD rule specifies the semantic assignment for a portion of the assignment domain specified in the ASV rule.

These methods for comparing the rule sets assume that the rules in the DMD may have incomplete antecedent restrictions. For example if a rule in the DMD is:

Instrument_Type = "equity" ⇒
Trade_Price_Status := 'latest_nominal_price' **and** Currency := 'pesetas'

it would match the first and second rule in the ASV in Figure 5. It matches the first rule because the rule contains the a constraint on the Instrument_Type attribute. It matches the second rule matches even though there are no common attributes in the antecedent. This is because the constraint in the DMD does not exclude constraints on other attributes in the assignment domain. In this example, the database would provide data for *equities* traded on the *nyse* with the semantic assignment defined in this rule even though this DMD rule only restricts the Instrument_ Type attribute.

If a rule in the DMD matches the rule in the ASV then the semantic restrictions in the consequent of these rules must be compared to determine if they are *semantically equivalent*; where semantic equivalence for each attribute is defined by the application. Procedures for defining semantic equivalence will be described in Section 4.2.1.

Table 1 contains the results from the comparison of the ASV shown in Figure 5 and the DMD shown in Figure 3. As an example from this table, the first rule in the ASV matches the first rule in the DMD according to the subset type of comparison shown in Figure 6. The antecedent constraints from the ASV and the DMD are shown along with the assignments to the semantic domains. The methods used to determine semantic equivalence values for Table 1 are described in the next section.

For attribute T with $X_i \epsilon assign(T)$ and $Y_i \epsilon sem(T)$

1. Antecedent(APP) is a *subset* of Antecedent(DB)

   APP : $C_1(X_1) \Rightarrow C_4(Y_1)$
   DB : $C_2(X_1) \wedge C_3(X_2) \Rightarrow C_4(Y_1)$

   (a) if $C_1(X_1) \Rightarrow C_2(X_1)$ then there is a **match**
   (b) if $C_2(X_1) \Rightarrow C_1(X_1)$ then there is a **match**
   (c) otherwise **no match**

2. Antecedent(APP) is a *superset* of Antecedent(DB)

   APP : $C_1(X_1) \wedge C_3(X_2) \Rightarrow C_4(Y_1)$
   DB : $C_2(X_1) \Rightarrow C_4(Y_1)$

   (a) if $C_1(X_1) \Rightarrow C_2(X_1)$ then there is a **match**
   (b) if $C_2(X_1) \Rightarrow C_1(X_1)$ then there is a **match**
   (c) otherwise **no match**

3. Antecedent(APP) *overlaps* Antecedent(DB)

   APP : $C_1(X_1) \wedge C_3(X_2) \Rightarrow C_5(Y_1)$
   DB : $C_2(X_1) \wedge C_4(X_3) \Rightarrow C_5(Y_1)$

   (a) if $C_1(X_1) \Rightarrow C_2(X_1)$ then there is a **match**
   (b) if $C_2(X_1) \Rightarrow C_1(X_1)$ then there is a **match**
   (c) otherwise **no match**

4. Antecedent(APP) *disjoint* Antecedent(DB)

   APP : $C_1(X_1) \wedge C_2(X_2) \Rightarrow C_3(Y_1)$
   DB : $C_4(X_4) \Rightarrow C_3(Y_1)$
   then there is a **match**

Figure 6: Four Types of Comparisons

$\mathbf{sem}_D$ = < Trade_Price_Status$_D$, Currency$_D$ >
$\mathbf{sem}_A$ = < Trade_Price_Status$_A$, Currency$_A$ >

$\mathbf{sem}$(Trade_Price$_D$) ≡ $\mathbf{sem}$(Trade_Price$_A$) if
string-equivalent(Trade_Price_Status$_D$, Trade_Price_Status$_A$)
string-equivalent(Currency$_D$, Currency$_A$)

Figure 7: Semantic Equivalence for Trade_Price

$\mathbf{sem}_D$ = < Trade_Price_Status$_D$, Currency$_D$ >
$\mathbf{sem}_A$ = < Trade_Price_Status$_A$, Currency$_A$ >

$\mathbf{sem}$(Trade_Price$_D$) ≡ $\mathbf{sem}$(Trade_Price$_A$) if
string-equivalent(Trade_Price_Status$_D$, Trade_Price_Status$_A$)
convert-currency(Currency$_D$, Currency$_A$)

Figure 8: Semantic Equivalence Using Currency Conversion

4.2.1 Semantic Equivalence

The definition of semantic equivalence is left to the application developer and is included as part of the ASV. For each non-primitive attribute the application developer must define the qualifications for semantic equivalence over assignments to the semantic domain. A simple example is shown in Figure 7 where the application requires that, for the Trade_Price attribute, assignments to the semantic domain are equivalent only if the values for the database (i.e., subscript D) and those in the application (i.e., subscript A) are identical strings. According to this definition the first and second comparisons in Table 1 are equivalent while the last is not because *latest_trade_price* is not the same string as the *latest_closing_price.*

There are a number of advantages in allowing the application to define semantic equality. First, not all applications will have the same requirements for data semantics. For example, an application may require trade prices whose semantics are *string-equivalent* for both Currency and Trade_Price_Status while another application may have less strict requirements that allow the latest closing price in lieu of the latest trade price. Secondly, an application specification for semantic equivalence may reference routines to convert data semantics. For example, if the metadata manager has access to a method, that can be used to convert one currency to another then the application can define the semantic equivalence of values of currency as shown in Figure 8. In this case, there is equivalence between currencies if they are string equivalent or the currency defined for Trade_Price in the DMD can be converted into the currency defined for Trade_Price in the ASV.

The function *convert-currency* is a boolean function that determines if two currencies are comparable. However, there must be a corresponding implementation that provides the exchange rate for converting a trade price in one currency to a trade price in another currency. Knowing that there is such a conversion function may not assure that at query execution time the conversion can be performed (e.g., at specific times currency conversion rates may not be available for certain currencies). The evaluation of semantic equivalence may have to be delayed if conversion routines need to be executed at query run-time. In the remainder of the examples we assume that semantic equivalence can be evaluated when comparing the rule sets. In Section 4.3.5 we consider the changes in query processing methods when the evaluation of semantic equivalence must be done at query execution time.

4.2.2 Results from Comparisons of Application and Database Metadata

Prior to query execution time, we can use the results from the comparison of the ASV and DMD rule sets along with the definition of semantic equivalence to determine, for a given attribute, whether the database can supply data with the correct semantics. As a result of the comparisons the metadata manager can determine the *semantic status* for each non-primitive attribute, i.e., whether data for that attribute attribute will **never, always or may** be meaningful to an application. In this section we present an example for each of the three possible results.

First, consider an ASV with the following single rule for the semantics of Trade_Price:

> Instrument_Type = "equity" **and** Exchange = "madrid" ⇒
> Trade_Price_Status := "latest_trade_price" **and** Currency := "pesetas"

and the same semantic and assignment domains defined in Figure 5 and the definition of semantic equivalence shown in Figure 7. Under these specifications the database can **never** supply a *non-null* solution[1] because there is only a single rule in the DMD (Figure 3) that matches this rule and the assignment to the semantic domain is in conflict. In this example, the database provides the *latest nominal price* while the application requires the *latest trade price*. For this example, any application query containing a reference to the Trade_Price attribute would have no meaningful non-null solution. In addition, if there are no matching rules for a given attribute then the database can **never** provided meaningful data.

Secondly, consider an ASV with the following rule:

> Instrument_Type = "equity" **and** Exchange = "madrid" ⇒
> Trade_Price_Status := "latest_nominal_price" **and** Currency := "pesetas"

and the definition of semantic equivalence in Figure 7. In this example the database can **always** supply meaningful data for the Trade_Price attribute. There is only a single matching rule in the DMD and the semantic assignment for that rule is semantically equivalent to the semantic assignment defined in the ASV (i.e., for this example the table of comparison would be only the first column in Table 1). The correct semantics are **always** provided because any query from the application will refer to data with the meaning defined in the ASV and this meaning is guaranteed by the database.

Finally, for the ASV shown in Figure 5 and the definition of semantic equivalence in Figure 7 the database **may** be able to provide data with the correct semantics. In this example the first rule

[1] For simplicity, we will only consider meaningful non-null solutions. In Section 4.3.2 we describe the conditions where a meaningful null solution can be determined for queries where there are semantic conflicts.

in the ASV does not conflict (i.e., semantic equivalence holds) with the matching rule in the DMD. The second rule in the ASV matches two rules in the DMD and conflicts with the second of these rules. The conflict occurs because for *futures* traded on the *nyse* the application expects the trade price to be reported as the *latest_trade_price* while the database provides the *latest_closing_price*. Because of this semantic conflict, any application query that refers to Trade_Price data on futures might return semantically incorrect data.

In the case where the database **may** deliver the correct data, an application query could be modified to eliminate any possible conflict. However, additional query constraints may limit the application beyond the restrictions specified in the query or the ASV. In this example, the application query would have to be modified so that the Trade_Price for futures could not be included in solution. This change in the application query might require that the application be notified of a change to the original query. In Section 4.3.3 we describe the query processing strategies for restricting application queries to guarantee semantic correctness.

The metadata management systems must create and maintain Table 1 which describes the results of comparisons between rules in the ASV and DMD. These tables are created prior to the submission of application queries. As shown in Section 4.3 these tables may be modified by the introduction of constraints in an application query. As described in Section 4.3.3, the metadata manager will have to reevaluate these comparisons as changes are made in either the application or the database semantics. In the next section we examine query processing strategies, based on the ASV and DMD comparisons, for determining when the application will receive meaningful data.

4.3 Query Processing and Semantic Reconciliation

In this section we examine the use of metadata in semantic reconciliation for application query processing. Initially, we examine the stages of query processing where the results of comparisons between the ASV and the DMD are used to assess the potential of the database to supply a meaningful solution to an application query. Following this we describe a different approach to query processing which uses the results of comparisons between the ASV and the DMD to define modifications to the application query such that the application is guaranteed to receive a meaningful but possibly partial solution to a query.

4.3.1 Query Processing: Stages for Detecting Semantic Conflicts

Prior to the submission of an application query the metadata manager has created tables similar to Table 1 for each non-primitive attribute in the ASV. During the *compile-time* stage, query processor must consider each attribute in the query (i.e., any part of the projection list of attributes and any attribute constrained in the query) and determine if the database might (i.e., may or always) supply the correct semantics. For example, there may be attributes in the database that will never provide meaningful data (i.e., either all matching rules result in semantic conflict or there are no matching rules). For a query that contains such an attribute, the outcome from query processing with semantic reconciliation is:

> Query Resolution by Semantic Conflict at Compile-time - there is a semantic conflict between the database and the application for at least one attribute in the query. The conflict can be determined prior to query execution based on the results of comparisons between the ASV and the DMD as determined prior to query submission.

and as a result the query is aborted. The application can be notified that an unresolvable semantic conflict was identified prior to execution (i.e., users could actually receive detailed descriptions of the conflict so as to permit the user to work towards a resolution).

Still prior to query execution time the constraints in the query can be used to remove comparisons that are no longer applicable because the constraints in the query invalidate the comparison. Determining applicable rules is equivalent to adding the constraints in the query to the antecedent of each rule in the ASV. If a contradiction occurs between these added constraints and the constraints in the antecedent of a rule in the ASV then the rule no longer applies. The contradiction signifies that the query will not require data from that portion of the application view. The remaining modified rules are matched against the DMD according to the methods for comparison defined in Figure 6. As an example consider the impact of query Q_1:

select Trade_Price (Q_1)
where Instrument_Name = "future"

on the comparisons in Table 1. The constraint on Instrument_Type is in contradiction with the first rule in the ASV (i.e., "future" $\neq$ "equity"). The database will not be required to supply any Trade_Price data on equities and this test for semantic equivalence is irrelevant. The second rule is still applicable but only for Instrument_Type = "future". With this restriction the only matching rule is the last one in the DMD. There is a semantic conflict in this portion of the application view. As a result the database can **never** provide data to this query. For such a query the outcome from query processing with semantic reconciliation is:

> Query Resolution by Semantic Conflict at Compile-time through Reduction - after reducing the number of applicable comparisons there is at least one attribute that can **never** provide data with the correct semantics. Again, this conflict can be determined prior to query execution time.

Also prior to query execution time it can be determined that a query whose attributes **always** provide the correct semantics can be processed correctly. Query modification must be used to include the constraints specified in the applicable rules in the ASV. Again, the comparisons between the ASV and DMD may change with the consideration of constraints in the query. For example, consider query Q_2:

select Trade_Price (Q_2)
where Instrument_Name = "equity"

and the comparisons in Table 1. The constraint on Instrument_Type is in contradiction with the conditions of the match between the second rule in the ASV and the third rule in the DMD (i.e., "future" $\neq$ "equity"). The database will not be required to supply any Trade_Price data on futures. This eliminates any possible semantic conflicts for Trade_Price. For this query, there are no conflicts and the database can provide the correct semantics for the Trade_Price attribute.

Finally, there is the case where no conflicts occur at compile-time but there is at least one attribute in the query that **may** provide the correct semantics. Again, the number of qualifying comparisons is reduced to account for constraints in the query. There may still remain at least one attribute for which there is a comparison between the DMD and the ASV where there is both semantic agreement and semantic conflict. For example, consider query Q_3,

select Trade_Price (Q_3)
where Instrument_Name = "IBM"

and the comparisons in Table 1. All of the comparisons in Table 1 are still valid because there is no conflict (i.e., known prior to run-time) between the constraint in the query and those in the antecedent of the rules in the ASV or DMD. However, the solution to this query may not be meaningful because there will be a semantic conflict if the data retrieved is a *future* traded on the *nyse* (i.e., "latest_trade_price" $\neq$ "latest_closing_price"). Query execution must be followed by a process that checks for conflicting data. In this example, any data where the instrument is a *future* would be in conflict. Query modification is used to add constraints and to add any attributes to the projection list that are required for checking for semantic agreement. The modified query is as follows:

select Trade_Price, Instrument_Type, Exchange (Q_4)
where Instrument_Name = "IBM'
and ((Instrument_Type = "equity" and Exchange = "madrid")
or (Exchange = "nyse")

At this stage of processing constraints are added to the query in a way similar to conventional query processing using view definitions. Rather than constraints begin provided by the conventional view definition they are provided by the ASV based on the results of comparisons with the DMD. The query processor must (1) identify which constraints can be added to the query without changing the semantics of the query (2) determine which additional constraints must be met to guarantee semantic correctness, and (3) determine which attributes must be added to the projection list to facilitate checking for semantic correctness. The procedures for identifying the correct constraints are determine by the comparison type. For example, in Figure 9 we show the requirements for the *subset* comparison type.

An example of this modification is demonstrated by the application query Q_3 and the modified query Q_4. The first rule in the ASV (Figure 5) matches the first rule in the DMD (Figure 3) through the *subset* comparison type. This adds the constraint on Instrument_Type and Exchange. The second rule in the ASV matches the second and third rules in the DMD. The first comparisons is an equivalence and adds to the query the constraint, Exchange = "nyse". The second comparison results in a conflict so the **new restriction** defined in Figure 9 must be satisfied by any acceptable solution. This new restriction:

not(Exchange = "nyse" **and** Instrument_Type = "future")

must be added to the list of constraints that are used to test for semantic conflicts at run-time. So that the new restriction can be tested at run-time the Instrument_Type and Exchange attributes must be added to the query's projection list.[2] Should any of these constraints be violated then semantic reconciliation leads to:

Query Resolution by Semantic Conflict at Run-time - at query execution time the data retrieved from the database is used to determine that there is a semantic conflict.

[2]Optimizations to these query modification procedures exist but are not considered in this paper.

For attribute T with $X_i \epsilon assign(T)$ and $Y_i \epsilon sem(T)$

- Antecedent(APP) is a *subset* of Antecedent(DB)

  APP : $C_1(X_1) \Rightarrow C_4(Y_1)$

  DB : $C_2(X_1) \wedge C_3(X_2) \Rightarrow C_5(Y_1)$

  – if semantic equivalence holds then

  1. if $C_1(X_1) \Rightarrow C_2(X_1)$ then add $C_1(X_1)$
  2. if $C_2(X_1) \Rightarrow C_1(X_1)$ then add $C_2(X_1)$

  – if a semantic conflict occurs then

  1. if $C_1(X_1) \Rightarrow C_2(X_1)$ then add the **new restriction** $not(C_1(X_1) \wedge C_3(X_2))$
  2. if $C_2(X_1) \Rightarrow C_1(X_1)$ then add the **new restriction** $not(C_2(X_1) \wedge C_3(X_2))$

Figure 9: Constraints for Subset Comparison Type

Otherwise, the values for the additional attributes on the projection list are removed before the solution is sent to the application.

The logic for query modification is as follows. Each comparisons between a rule in the ASV and a rule in the DMD can contribute at most one constraint (i.e., may be the conjunction of restrictions on different attributes) to the query. For a rule in the ASV with multiple matching rules in the DMD, each one that is an equivalence forms a *disjunction* of conditions for that ASV rule. For each non-primitive attribute, the conditions determined by each rule in the ASV form a *disjunction* of conditions (i.e., each rule represents an acceptable semantic interpretation). Finally, the conditions for each non-primitive attribute form the *conjunction* of semantic restrictions that must be added to the query. At the same time, for all semantic conflicts, the negation of the conjunction of the constraint defined in Figure 9 are placed on the list of conditions that must be satisfied by any acceptable query solution.

For query Q_4, a conflict would occur if the solution included data on a *futures* instrument traded on the *nyse.* In actual execution query Q_4 would locate a single record in sample relation (Figure 1). Query processing using semantic reconciliation will correctly determine that *IBM* is an *equity* traded on the *nyse* and return to the application the trade price reported as the *latest trade price* in *US dollars.*

A query that is not resolved by one of the stages of semantic conflict will have a solution that is semantically meaningful to the application. This method of query processing assumes that an application (i.e., user) does not permit query modification that changes the meaning of the query. In Section 4.3.3 we describe query processing techniques using semantic reconciliation that can be used to modify the user query to permit only semantically meaningful results.

4.3.2 Query Processing: Null Solutions

Thus far we have described a number of methods that can be used prior to query execution time to determine that a query will not have a semantically meaningful non-null solution. Under certain conditions such a query can actually be executed to determine whether a null solution is correct. For example, consider the query Q_5:

$$\begin{aligned}&\textbf{select } \text{Trade\_Price} \\ &\quad \textbf{where } \text{Instrument\_Type} = \text{"future"} \\ &\quad \textbf{and } \text{Instrument\_Name} = \text{"XYZ"}\end{aligned} \qquad (Q_5)$$

and the comparisons in Table 1. The first two comparisons are removed by contradiction with the constraint on Instrument_Type. The query is then resolved by semantic conflict through reduction. However, the query can be executed because the constraints in the query are meaningful (i.e., they are restrictions on primitive attributes). Execution of the query may return a null solution if constraints are not matched in the database, any other solution would be semantically meaningless.

Queries resolved through semantic conflict can be executed if the following is true:

> *Any attribute that is in semantic conflict either appears only in the projection list or it appears in a join condition with another attribute that is comparable to it.*

Attributes are comparable if they are semantically equivalent. Testing for comparability of two attributes is identical to testing for semantic agreement between an attribute in the application and the database. Selection conditions are not included because the data cannot be correctly compared to the restriction in the constraint because the restriction is based on the application semantics which are known to be in conflict with those of the database.

4.3.3 Query Processing: Adding Restrictions to Guarantee Correctness

Query processing, as described in Section 4.3.1, used compile-time information used to test the semantic correctness of the application query. Then there were query modifications that did not effect the meaning of the query. The query was then executed and the results were examined to determine if there were semantic conflicts. If a conflict occurred the query was resolved by semantic conflict at run-time. The approach to query processing described in this section is identical except that constraints can be added so as to guarantee semantically meaningful partial solutions to the application query. An application (e.g., user) may be designed to accept partial solutions to queries in exchange for semantic correctness. If permitted by the application, queries that would normally be resolved through semantic conflict at run-time are candidates for this type of query modification.

As an example of the use of constraints to provide a correct partial solution, consider query Q_3 from above. Under normal operations any violation of the constraint

not(Exchange = "nyse" **and** Instrument_Type = "future")

would lead to query resolution by semantic conflict at run-time. This would be the case even though there may be a partial solution that does not include conflicts. Rather than reporting that the results are not meaningful the query processor can simply remove any incorrect solutions. For

For attribute T with $X_i \epsilon assign(T)$ and $Y_i \epsilon sem(T)$

- Antecedent(APP) is a *subset* of Antecedent(DB)

  APP : $C_1(X_1) \Rightarrow C_4(Y_1)$
  DB : $C_2(X_1) \wedge C_3(X_2) \Rightarrow C_5(Y_1)$

  – if semantic equivalence holds then

  1. if $C_1(X_1) \Rightarrow C_2(X_1)$ then add $C_1(X_1) \wedge C_3(X_2)$
  2. if $C_2(X_1) \Rightarrow C_1(X_1)$ then add $C_2(X_1) \wedge C_3(X_2)$

  – if a semantic conflict occurs then

  1. if $C_1(X_1) \Rightarrow C_2(X_1)$ then add the **new restriction** $not(C_1(X_1) \wedge C_3(X_2))$
  2. if $C_2(X_1) \Rightarrow C_1(X_1)$ then add the **new restriction** $not(C_2(X_1) \wedge C_3(X_2))$

Figure 10: Constraints for Subset Comparison Type - Partial Solutions

query Q_3 and the comparisons in Table 1 the modified query Q_6 would include restrictions that remove any tuples that are in conflict.

select Trade_Price (Q_6)
where Instrument_Name = "IBM"
and (Instrument_Type = "equity" **and** Exchange = "madrid")
or ((Exchange = "nyse" **and** Instrument_Type = "equity")
and *not*(Exchange = "nyse" **and** (Instrument_Type = "future"))

As in Section 4.3.1, the methods for adding constraints to the query are determined by the comparison type. The constraints used in query modification for the *subset* comparison type are shown in Figure 10. For example, consider query Q_3 and the modified query Q_6. The second rule in the ASV (Figure 5) matches with two rules in the DMD (Figure 3). The first is an equivalence so according to the Figure 10 the constraint:

(Exchange = "nyse" **and** Instrument_Type = "equity")

is added to the query. The second match results in a semantic conflict and the constraint:

not(Exchange = "nyse" **and** Instrument_Type = "future")

must be added to the query. The negation of the constraint found in the DMD is added to the query to limit the result to correct data. The addition of these constraints to the query changes the meaning of the application query by reducing the scope of the original query. The result may be a partial solution to the original query but it is guaranteed to be a semantically meaningful solution. As for changes to the original query, the user can be informed of the added restrictions, the reasons for the added restrictions, and a list of the records that were eliminated as a result of these restrictions.

The logic for query modification is as follows. Each comparisons between a rule in the ASV and a rule in the DMD can contribute at most one constraint (i.e., may be the conjunction of restrictions on different attributes to the query). For an ASV rule with multiple matching rules in the DMD the constraints from equivalence matches form a *disjunction* of conditions. The matches that result in conflict are in *conjunction* with these conditions and in *conjunction* with the constraints from other conflicting matches for the same rule in the ASV. Again, for a given attribute the conditions resulting from each rule in the ASV form the *disjunction* of the possible interpretations for that attribute. Finally, the conditions for each attribute form the *conjunction* of semantic restrictions that must be added to the query.

4.3.4 Query Resolution by Semantic Restriction

During the process of query modification constraints are added to the query and the query statement may be reduced to the point where the only acceptable solution to the query appears from logical reduction of the constraint list. As an example, consider query Q_7:

select Instrument_Type (Q_7)
 where Trade_Price > 50.00
 and Exchange = "madrid"

and the results of comparisons in Table 1. The constraints in the table are added to the query to produce the modified query Q_8.

select Instrument_Type (Q_8)
 where Trade_Price > 50.00
 and Exchange = "madrid"
 and (Exchange = "madrid" **and** Instrument_Type = "equity")

The query can be logically reduced to:

select Instrument_Type (Q_9)
 where Trade_Price > 50.00
 and Exchange = "madrid"
 and Instrument_Type = "equity"

and further reduction leads to the only possible non-null solution:

Instrument_Type = "equity".

Unfortunately, this methods of query resolution may not produce the same answer as executing the query. Because there may be **no** data for equities with a trade price greater than 50.00 on the Madrid Stock Exchange the query could return a null result. A similar problem was found in query reduction using semantic query optimization [CFM84,HZ80,Kin81,SSS91]. During semantic query optimization integrity constraints may be added or removed from a query. Contradictions may occur as they do in query processing using semantic reconciliation. In addition a logical reduction of the query may lead to the only possible non-null solution to the query. But is was shown in [SSS91] that the null solution was feasible and therefore some query execution is required. Execution of these queries can be simplified because as soon as a single solution is found in the database then the result determined by semantic restriction will be correct.

4.3.5 Query Processing: Semantic Equivalence

An application's definitions for semantic equivalence may include references to conversion functions that must be executed at run-time to convert the semantics of the database data to those required by the application. Checking for semantic equivalence includes the evaluation of boolean functions that define the conversion capabilities. For example, the definition for semantic equivalence in Figure 8 contains a reference to a currency conversion function. The definition of semantic equivalence must determine if the currency provided by the database is semantically equivalent to that required by the application. This may depend on the ability, at run-time, to convert the database to those acceptable by the application. In this example, this may involve a look-up for the exchange rates between two currencies for a specified date and time. Should that exchange rate not be available at run-time then the test for semantic equivalence would fail and the conflict would have to be considered during query processing.

For run-time semantic equivalence testing, the metadata manager must reevaluate the comparisons between rules in the ASV and the DMD based on this run-time information and the query processor must consider this new information during semantic reconciliation. Modifications to the query processing routines to include run-time semantic equivalence must define a correct execution order for semantic equivalence testing and methods for semantic reconciliation.

5 Semantic Reconciliation and Changing Database Semantics

It is important that the methods for determining semantic agreement among systems allow for changes in the semantics of those systems. Rules defining the semantics of the database and the application are likely to change many times during the life-cycle of the source-receiver relationship. Most databases are not static and just as the structure may change so may the meaning of the data. In fact, our experience leads us to believe that changes in the semantics of data are more common occurrences than changes in structure.

The methods presented for query processing and semantic reconciliation can be used in such a dynamic environment. As changes are made in the ASV or DMD (i.e., corresponding to changes in the semantics of the database or application) the metadata manager must reevaluate any comparisons that are effected by these changes. A comparisons is effected if the ASV or DMD rule used in the comparison is modified. Additionally, rules added to the ASV or DMD must be evaluated according to the methods described in Section 4.2. The metadata manager can then determine any changes in semantic status of the attributes in the ASV. For example, an attribute that **may** provide meaningful data might be changed to one that **always** provides the meaningful data when the rules in the DMD defining the semantics of that attribute are modified. For the comparisons in Table 1, should the database decide to report *latest_trade_price* for futures rather than *latest_closing_price* then the semantic status of the Trade_Price attribute would change from **may** to **always**. The methods for semantic reconciliation permit changes to the semantics at the database or application as long as those changes remain inside of restrictions for the semantic representation model.

6 Conclusions and Future Research

In this paper we described methods for using metadata to automatically identify and resolve semantic conflicts between a data source and a receiver. These methods apply to dynamic system environments. When data semantics change at the source or data semantic requirements change at the receiver these methods can be used to determine if the source can continue to supply meaningful data.

We described a model for representing information on data semantics and provide an architecture for a system that uses this representation for semantic reconciliation. Our representation model is general enough to define metadata for a variety of domains. Using metadata, we show how an application can specify its requirements for data semantics and application specific definitions for semantic equivalence. Allowing the application to define semantic equivalence has the advantage that different applications can express different requirements for data semantics. Additionally, applications can reference functions, defined in the ASV or DMD, in these definitions. These function can be used to automatically convert data semantics, making it possible for the application to receive meaningful data from the source when such data could not normally be provided.

We presented methods for comparing rules that describe the *application's semantic* view and the *database metadata definition*. The metadata manager maintains the results of these comparisons for use in query processing. Prior to query presentation the metadata manager can determine the semantic status of each non-primitive attribute. The constraints in a query then provide for refinement of the comparisons of the attributes semantic rule sets in the ASV and DMD. Semantic reconciliation may result in query resolution by semantic conflict prior to query execution. If no conflicts occur at compile-time then the query can be executed and the solution tested for semantic conflicts. At any stage of this process the user may obtain information describing any conflict that has occurred. Alternatively, query modification can be used to guarantee semantically meaningful partial solutions.

Future research will examine a more general representation [SM89b] for data semantics that will permit the application and the database to more freely define data semantics. This will include a better understanding of common language requirements and the relationship between the semantic requirements for applications and database semantic specifications. The present representation model and methods for semantic reconciliation address simple data semantics, complex data semantics (e.g., derivation formula) will require additional data structures and algorithms if they are to be considered in semantic reconciliation.

Methods for semantic reconciliation using metadata have been presented for systems that conform to the source-receiver model. These results can be extended so that they are useful in multidatabase systems. The need to represent and manipulate data semantics or metadata is particularly important in tightly-coupled federated database systems where data is taken from multiple disparate sources. Integration of multiple systems may require the definition of a global schema representing the composition of the component database schemas [DK86,LR82,She87,Te87]. Typically, schema integration algorithms have been developed for component databases with static structure and semantics [BLN86,CRE87,SG89]. However, to allow for greater local database autonomy, schema integration must be considered a dynamic problem. The global schema must be able to evolve to reflect changes in the structure [BMW86,McL88] and meaning of the underlying databases. If an application is affected by these changes, it must be alerted. Semantic reconciliation will be required between and application and a global schema and between the component schemas and the global schema [SM89a]. Similarly, in federated systems [HM85,SL90] metadata can be used to describe the import and export semantics and methods defined in this paper can be used to determine the semantic relationship between components in the federation.

Acknowledgments

The authors would like to thank Sandra Heiler and Arnie Rosenthal for their helpful reviews of this paper. This work was supported, in part, by Reuters and the International Financial Services Research Center at the Massachusetts Institute of Technology.

References

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[BMW86] A. Borgida, T.M. Mitchell, and K. Williamson. Learning improved integrity constraints and schemas from exceptions in databases and knowledge bases. In Michael Brodie and John Mylopoulos, editors, *On Knowledge Based Management Systems*, pages 259–286, Springer-Verlag, 1986.

[CFM84] U. Chakravarthy, D. Fishman, and J. Minker. Semantic query optimization in expert systems and database systems. In *Proceedings of the First Intl. Conference on Expert Database Systems*, pages 326–340, 1984.

[CRE87] B. Czejdo, M. Rusinkiewicz, and D. Embley. An approach to schema integration and query formulation in federated database systems. In *Proceedings of the Third International Conference on Data Engineering*, February, 1987.

[DK86] P. Dwyer and K. Kasravi. A heterogeneous distributed database management system (DDTS/RAM). In *Honeywell Report CSC-86-7:8216*, 1986.

[GK88] A. Goldfine and P. Konig. *A Technical Overview of the Information Resource Dictionary System (Second Edition). NBSIR 88-3700*, National Bureau of Standards, 1988.

[GSdB88] P. Gray, G. Storrs, and J. du Boulay. Knowledge representations for database metadata. *Artificial Intelligence Review*, 2:3–29, 1988.

[HM85] D Heimbigner and D. McLeod. A federated architecture for information management. *ACM Transactions on Office Information Systems*, 3(3), 1985.

[HZ80] M. Hammer and S. Zdonik. Knowledge-based query processing. In *Proceedings 6th VLDB*, pages 137–146, 1980.

[Kin81] J. King. QUIST : A system for semantic query optimization in relational databases. In *Proceedings 7th VLDB*, pages 510–517, 1981.

[Law88] M. H. Law. *Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning. 500-152*, National Bureau of Standards, 1988.

[LR82] T. Landers and R. Rosenberg. An overview of multibase. In *Distributed Data Bases*, pages 153–183, North Holland, 1982.

[McC82] J. McCarthy. Metadata management for large statistical database. In *Proceedings of the Eight International Conference on Very Large Database Systems*, pages 470–502, Mexico City, 1982.

[McC84] J. McCarthy. Scientific information = data + meta-data. In *Database Management: Proceedings of the Workshop November 1-2, U.S. Navy Postgraduate School, Monterey, California*, Department of Statistics Technical Report, Stanford University, 1984.

[McC87] J. McCarthy. Information systems design for material properties data. In *Proceedings of the First International Symposium on Computerization and Networking of Material Property Databases*, American Society for Testing and Materials, Philadelphia, 1987.

[McL88] D. McLeod. A learning-based approach to meta-data evolution in object-oriented databases. In *Advances in Object-Oriented Database Systems*, Springer-Verlag Lecture Notes In Computer Science, 1988.

[ML90] T. Malone and J. Lee. Partially shared views: a scheme for communicating among groups that use different type hierarchies. *ACM Transactions on Information Systems*, January 1990.

[SG89] A. Sheth and S. Gala. Attribute relationships: an impediment in automating schema integration. In *Position Papers: NSF Workshop on Heterogeneous Databases*, December 11-13, 1989.

[She87] A. Sheth. Heterogeneous distributed databases: Issues in integration, Tutorial on heterogeneous databases. In *Proceedings of the Conference on Data Engineering*, 1987.

[SL90] A. Sheth and J. Larson. Federated databases: architectures and integration. *ACM Computing Surveys*, September 1990.

[SM89a] M. Siegel and S. Madnick. *Identification and Resolution of Semantic Conflicts Using Meta-data*. Technical Report #3102-89-MSA, Sloan School of Management, Massachusetts Institute of Technology, 1989.

[SM89b] M. Siegel and S. Madnick. *Schema Integration Using Metadata*. Technical Report #3092-89-MS, Sloan School of Management, Massachusetts Institute of Technology, (Also NSF Workshop on Heterogeneous Database Systems, 1989), 1989.

[SSS91] M. Siegel, S. Salveter, and E. Sciore. Automatic rule derivation for semantic query optimization. *Accepted for publication to Transactions on Database Systems*, ((Also Boston University PhD Thesis 1989 - Siegel)), 1991.

[Te87] M. Templeton and et al. Mermaid - a front-end to distributed heterogeneous databases. In *Proceedings of the IEEE*, pages 695–708, 1987.

[YSDK90] C. Yu, W. Sun, S. Dao, and D. Keirsey. Determining relationships among attributes for interoperability of multi-database systems. In *Position Papers: Workshop on Multidatabases and Semantic Interoperability*, November 2-4, 1990.