IDENTIFICATION AND RECONCILIATION OF SEMANTIC CONFLICTS USING METADATA

MICHAEL SIEGEL
STUART E. MADNICK

November 1989

WP# 3102-89 MSA
CIS-89-09

Revised Version, June 1990

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

Identification and Reconciliation of Semantic Conflicts Using Metadata

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 an automatic method to establish such semantic agreement through the use of metadata. 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 data semantic specifications of the application. Methods are described for detecting these changes and for determining if the database can supply meaningful data to the application. These methods also can be used for semantic reconciliation where an application depends on multiple data sources and for schema integration in heterogeneous database systems.

1 Introduction

With the development of more complex 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 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 francs might decide to change to reporting that data in European Currency Units (ECU).

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 Dic-*

tionary 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.

This paper begins by examining 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. Later, we describe how metadata can be used to simplify schema integration and semantic reconciliation among multiple, possibly disparate, databases systems.

The methods proposed for semantic reconciliation allow for changes in data semantics in the database or for changes in the application's data semantic specifications. These methods can be used to automatically track these changes and determine, as a result of any changes, if the database can still supply meaningful data. Additionally, we show that in some instances metadata can be used to resolve data 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. Then 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 describe a general model for the representation of data semantics and provide a sample specification for a financial application. 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 and describe an algorithm that can be used to automatically determine if a database can provide an application data with meaningful data. Additionally, we describe how the application or the database can provide methods for resolving conflicts over data semantics. Then in Section 5 we examine how these methods can be applied to a more general representation. In Section 6 we show how methods used for semantic reconciliation in the source-receiver model can be used in multidatabase systems. Finally, we present our conclusions and a discussion of future research directions.

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 a National Bureau of Standard's *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.

Both the IRDS and the system described in McCarthy's work provide the flexibility to represent

most types of metadata. However these approaches do not include a well-defined methodology for utilizing this metadata for semantic reconciliation between two systems.

Providing the representation and functionality for all forms of metadata is beyond the scope of this paper. We are interested in handling those forms of metadata that can be specified for both the application and the database and can readily be used to simplify semantic reconciliation between them. For simplicity, we begin by describing a restricted view of data semantics. Then, in Section 5 we examine more general approaches to semantic reconciliation.

In the next section we examine an example 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.

3 Providing Metadata for Semantic Reconciliation

Generally, detailed data semantics are recorded in printed documents or manuals. These manuals can assist an analyst in understanding the data provided and for developing meaningful applications. However, the meaning of the data may change over time especially if it is being provided by another organization or is being used for multiple purposes. The analyst cannot always be aware of and account for these changes. As the semantics of the data change, the data may no longer be meaningful to an application. For example, an application that uses stock prices in a calculation is likely to be adversely affected if the source of this data decides to change from reporting stock prices in dollars to reporting them in francs. But the same application is unaffected if the data source changes the currency used for reporting bond prices. Thus, certain changes in data semantics might allow for an application to continue running correctly, while other changes will require modifications to the application. Printed material describing changes to data semantics are of little use in deciding automatically what should be done. Additionally, traditional manual approaches, besides being error-prone, are rapidly becoming infeasible as the number of sources and frequency of changes accelerate.[1] As an alternative, this work describes methods for representing and manipulating data semantics in a manner that will allow for automatic determination of the effect of changing data semantics. Additionally, these methods can be used to detect and respond to changes in an application's data semantic specifications.

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 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 price except for equities traded on the Madrid Stock Exchange where trade price represents

[1] As an example of the number of sources, in the United States there are currently over 3000 online data sources just for financial information [Cua87].

Instrument_Type	Instrument_Name	Exchange	Trade_Price
Equity	IBM	NYSE	115.25
Equity	Telecom SP	Madrid SE	1107.25

Figure 1: The FINANCE Relation

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 metadata 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, changing the database schema to represent metadata in this way is awkward and does not simply allow for the representation of numerous types of metadata. Furthermore, this approach is intrusive in the sense that it requires the data source to be changed which may not be feasible especially if the data source is a separate organization.

Alternatively, we suggest that a metadata management system can use simple procedures to associate metadata with a given attribute or data type. In the next section we describe a representation model for data semantics and introduce examples from the financial domain.

3.1 A Model for Data Semantics

By providing a model for the representation of data semantics we are able to define the range of applicability of our methods for semantic reconciliation. Additionally, we provide a language for representing these semantics.

We begin by defining the *semantic domain* of an attribute (or data type) T as the set of attributes (or data types) used to define the semantics of T and note this as

sem(T) = $<X_1, X_2, X_3, ..., X_n>$ where each X_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

sem(t) = $<x_1, x_2, x_3, ...x_n>$ where $x_i \epsilon$domain(X_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_price, us_dollars>

where the value 115.25 is a represents the latest 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 tϵdomain(T). The *assignment domain* of attribute **T** is defined as

$$\text{assign(T)} = <Y_1, Y_2, Y_3, ..., Y_n>$$

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

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

Using the semantic and assignment domains we can define semantic assignability as the existence of some procedure F that can be used to determine values in the semantic domain for values $t \epsilon T$.

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

sem(Trade_Price) = <Trade_Price_Status, Currency>

assign(Trade_Price) = <Instrument_Type, Exchange>

F(assign(Trade_Price)) = sem(Trade_Price)

For a given trade price, the instrument type being traded and the exchange that it is trade on are used to determine the currency and status of the trade price. A procedure for assigning values to the semantic domain may simply be a set of rules where the antecedent of a rules contains constraints on the assignment domain and the consequent of the rule assigns values to the semantic domain.

An attribute **T** is *trivially assignable* if T is semantically assignable over some procedure F such that for each tϵdomain(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 Xϵsem(T) X is primitive. Examples of primitive attributes might include Instrument_Type, Exchange, Currency and Trade_Price_Status. For example, the Currency attribute may have an empty assignment and semantic domain; thus the value of Currency, say, us_dollars, is a complete description of the Currency value.

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* if it is semantically assignable and either it is primitive or for all Yϵassign(T), Y is semantically definable and for all Xϵsem(T), X is semantically definable. An attribute **T** as *semantically simple* if it is primitive or T is assignable and for all Yϵassign(T), Y is primitive and for all Xϵsem(T), X is primitive. All simple attributes are semantically definable. Procedures defining the semantics of simple attributes do not require access to the assignment procedures of other semantically definable attributes. Other semantically definable attributes might require access to the procedures for determining the semantics of the attributes in the assignment domain.

As examples of semantic assignment consider the list of primitive attributes in Figure 2. In each case the semantic and assignment domain is empty therefore only the domain of the the attribute is defined. In Figure 3 Trade_Price is presented as a semantically definable (and simple) attribute.

domain(Instrument_Type) = <equity, future>.
domain(Exchange) = <nyse, madrid>.
domain(Currency) = <us_dollars, french_francs, pesetas>.
domain(Trade_Price_Status) = <latest_price, latest_nominal_price>.

Figure 2: Examples of Primitive Attribute Domains

This procedure contains a case statement that is used to define rules for determining the meaning of data. For example, based on the knowledge that an instrument is an equity traded on the Madrid Stock Exchange this procedure determines the meaning of the trade price as the latest trade price in pesetas. For all equities not traded on the Madrid Stock Exchange the procedure determines that the trade price is reported as the latest price in the currency of the exchange. These procedures are simply a packaging of rules that describe the data semantics.

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

```
case Instrument_Type of
'equity' :      case Exchange of
                'madrid' :      Trade_Price_Status := 'latest_nominal_price'
                                Currency := 'pesetas'
                otherwise :     Trade_Price_Status := 'latest_price'
                                Currency := (currency-of Exchange)
                endcase
otherwise :     Trade_Price_Status := 'latest_price'
                Currency := (currency-of Exchange)
endcase
```

Figure 3: Trade_Price as a Semantically Definable Attribute

In the next section we show how this model for metadata representation can be used in determining semantic reconciliation.

4 Using Metadata for Semantic Reconciliation

Semantic reconciliation is part of the process that ascertains logical connectivity [WM88] between the database and the application. We intend to use metadata, as described in Section 3.1, 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)

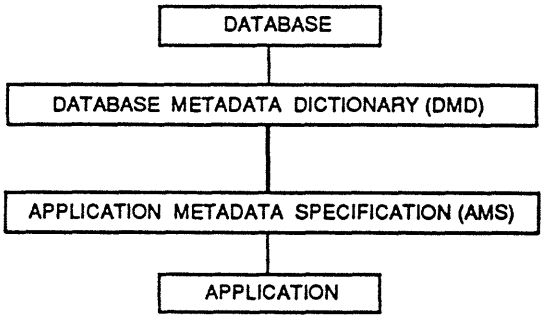

Figure 4: Systems Architecture Using Metadata

```
case Instrument_Type of
Equity :        case Exchange of
                'madrid' :      Trade_Price_Status := 'latest_nominal_price'
                                Currency := 'pesetas'
                endcase
endcase
```

Figure 5: Application Specification for Trade_Price

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 procedures that define the semantic assignments for each of these attributes. The *application metadata specification* (AMS) contains the applications definition of the semantic and assignment domain and the set of assignment functions for the attributes required by the application. The AMS can be thought of as an application's data semantics specifications while the DMD defines the semantics of the data.

An example of an AMS is shown in Figure 5. The specification includes a *precondition* that restricts the domain of the required data to equities traded on the Madrid Stock Exchange. For data fitting this precondition, the application expects that the source will report the trade price as the latest nominal price in pesetas.

To decide whether a database can supply meaningful data to an application we must determine if the procedures in the DMD guarantee the data semantics specified by the AMS. That is, for each attribute in the AMS the DMD procedure for that attribute must provide assignments to the semantic domain that match those in the AMS. For example, given the specification shown in Figure 5, does the procedure for Trade_Price shown in Figure 3 provide the correct instantiations of the semantic domain? If this is the case, then the DMD *subsumes* the AMS. More precisely, the DMD *subsumes* the AMS if for every precondition of every attribute in the AMS the DMD produces a set of instantiations of the semantic domain that *matches* the instantiations defined in the specification. Matching instantiations between the specification and the DMD requires that the instantiations be shown to be *semantically equivalent*; where semantic equivalence for each

attribute is defined by the application. If the DMD subsumes the AMS then this establishes semantic consistency between the application and the database (i.e., the database will supply meaningful data). Methods for defining semantic equivalence are described in Section 4.2. In the next section we describe an algorithm for determining subsumption.

4.1 Metadata Subsumption

Determining subsumption may appear to be similar to the known untractable problem of determining when two programs will produce the same output. However, given certain restrictions on the DMD and the AMS we can provide a tractable algorithm for determining when the specifications in the AMS are subsumed by the set of procedures in the DMD.

It is assumed that the procedures in the DMD and the specifications in the AMS are complete and consistent. For simplicity, it is also assumed that the application and the database contain at most one assignment procedure for each attribute and that these procedures are semantically simple. It is important that the restriction in these procedures be on primitive data types so that comparisons can be made between the values in these restrictions. Initially, we place the additional restriction that, for each attribute, the semantic and assignment domains for an application specification are subsets of the semantic and assignment domains in the DMD. This is the most restrictive of our assumptions, but we show later how more general specifications can be included in the AMS.

Even though this representation is restrictive, we have found it to be very valuable for two reasons. First, the restrictive representation allows us to develop a base methodology and a computable algorithm for addressing semantic reconciliation. This base methodology can be generalized (see Section 5) to deal with a wide range of semantic issues. Second, in our analysis of some existing databases, we have found that even this restricted representation is capable of capturing the semantics of many types of data.

Based on this set of assumptions we describe an algorithm for determining subsumption. The *Subsume* algorithm, defined in Figure 6, begins by constructing a graph representing the procedure for an attribute found in the DMD. For example, Figure 7 illustrates the graph derived from the procedure for Trade_Price defined in Figure 3.[2] The construction of this graph follows directly from the structure of the case statement. The graph forms a tree structure where each level of the tree represents a restriction on a different primitive data type identified by the node at that level. The leaves of the tree contain the metadata instantiations for a given precondition as defined by the path from the root node.

These graphs will be used to determine the possible instantiations defined by the DMD based on the preconditions found in the application. For a precondition selected in Step 2, the algorithm collects all possible instantiations defined by the DMD. For each possible precondition the algorithm requires that the instantiation in the specification be *semantically equivalent* to the instantiations defined by the DMD. If this is true for all preconditions of all attribute specifications then the algorithm succeeds, implying that the database can supply the application with meaningful data. This algorithm will, for a given precondition and DMD procedure, find exactly the correct set of instantiations [SM89a]. The subsumption algorithm is run for each attribute procedure found in

[2]The *otherwise* part of the case statement is shorthand for the enumeration all values for that data type that do not appear on edges leaving the node. For example, *otherwise* for *case Exchange of* in Figure 3 can be thought of as all exchanges besides the Madrid Stock Exchange.

1. For attribute M_A in the AMS construct the graph of the procedure for the same attribute M_D found in the DMD.
2. Select a precondition in M_A. Note the restrictions in this precondition and record the associated metadata instantiation as I_A.
3. For each data type defined by a level of the database graph mark all edges that match the value of a restriction identified in (2). If there is no restriction on the data type then mark all edges.
4. Remove all unmarked edges. The leaves of the remaining paths (from the root) define the set of all possible metadata instantiations, I_D, for the precondition defined in (2).
5. If I_D is semantically equivalent to I_A then continue else **fail**.
6. Return to Step 2 for every precondition specified in the case statement for M_A.
7. If no failure occurred, then **succeed**.

Figure 6: Subsumption Algorithm - **Subsume**

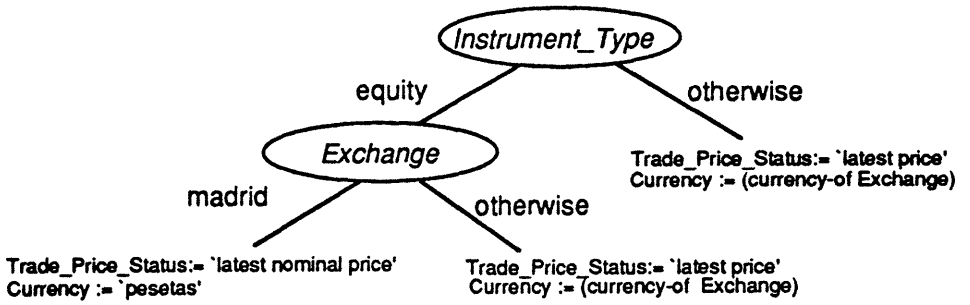

Figure 7: Graph of DMD Procedure for Trade_Price

I_A is
  Trade_Price_Status := 'latest_nominal_price'
  Currency := 'pesetas'

I_D is
  Trade_Price_Status := 'latest_nominal_price'
  Currency := 'pesetas'

Figure 8: Instantiations to be Tested for Semantic Equivalence (Step 5)

the AMS. If the algorithm succeeds for all attribute procedures found in the AMS then all data provided by the database is meaningful to the application.

For the AMS defined in Figure 5 the algorithm begins by constructing the graph shown in Figure 7 for the Trade_Price attribute. Step 2 of the algorithm identifies the only precondition in the specification. This precondition contains the following restrictions on primitive data types:

Instrument_Type ='equity'
Exchange = 'madrid'

The instantiation, I_A, associated with this precondition is identified in the top half of Figure 8. Next, for each restriction on the assignment domain we mark the edge in the graph matching the restriction. In this example the single edge labeled 'equity' and the single edge labeled 'madrid' are marked. All unmarked edges are removed leaving a single path from the root to the leaf containing the instantiation I_D shown in the bottom of Figure 8. Step 5 of the subsumption algorithm requires the instantiations I_A and I_D to be semantically equivalent. Methods used to define and determine semantic equivalence are presented in the next section. Given that the instantiations are equivalent, the algorithm continues by attempting to locate any other precondition in the specification. As there are no other preconditions in this attribute procedure, the algorithm finishes without failure and therefore subsumption holds for the Trade_Price attribute.

Another example of this process is shown for the application specification in Figure 9. Here the application requires that all instruments traded on the Madrid Stock Exchange be reported as the latest nominal price in pesetas. According to this restriction we mark the single edge labeled 'madrid' and because there is no restriction on Instrument_Type we mark all edges leaving the node labeled with this data type. The resulting spanning tree is shown in Figure 10.

Subsumption for this example requires that the instantiations shown in Figure 11 be semantically equivalent. The function call (currency-of Exchange) is replaced by the function call (currency-of 'madrid') as a result of the restriction on Exchange identified in the precondition. Semantic equivalence for these instantiations are described in the next section.

4.2 Defining Semantic Equivalence

The definition of semantic equivalence is left to the application developer and is included as part of the application metadata specification. A sample definition of semantic equivalence is shown

```
case Exchange of
'madrid' :      Trade_Price_Status := 'latest_nominal_price'
                Currency := 'pesetas'
endcase
```

Figure 9: Application Specification for Trade_Price

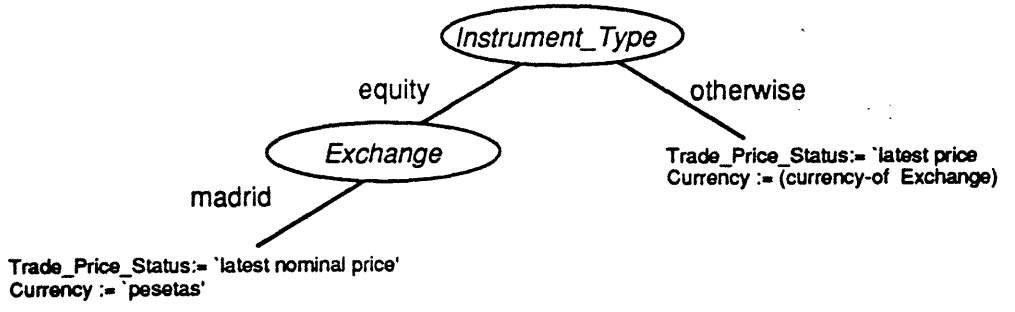

Figure 10: Spanning Tree for the AMS in Figure 9

I_D is

```
    Trade_Price_Status := 'latest_nominal_price'
    Currency := 'pesetas'
and
    Trade_Price_Status := 'latest_price'
    Currency := (currency-of 'madrid')
```

I_A is

```
    Trade_Price_Status := 'latest_nominal_price'
    Currency := 'pesetas'
```

Figure 11: Instantiations to be Tested for Semantic Equivalence (Step 5)

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

sem(Trade_Price$_D$) $\equiv$ **sem**(Trade_Price$_A$) if
string-equivalent(Trade_Price_Status$_D$, Trade_Price_Status$_A$)
string-equivalent(Currency$_D$, Currency$_A$)

Figure 12: Semantic Equivalence for Trade_Price

in Figure 12 where the application requires that the instantiations from the DMD and the AMS be identical strings. Under this definition, the instantiations I_A and I_D shown in Figure 8 are semantically equivalent.

For the same definition of semantic equivalence, the sets of instantiations shown in Figure 11 are not equivalent. The function *currency-of* must be evaluated before we can test for semantic equivalence between currencies. This requires that the DMD evaluate the function and its argument before the test for equivalence. Evaluation of the function, *currency-of*, can be done at "compile-time"(i.e., schema design time) if it is assumed the results of the function do not change over time (i.e., that exchanges do not change currencies). Otherwise, the evaluation of such functions and the determination of semantic equivalence may have to be done at "run-time" (i.e., data access time). Assuming that the currency for the Madrid Stock Exchange is pesetas, then the currency values are semantically equivalent. However, the application requires the latest nominal price while the database reports either the latest nominal price or the latest price. Because both of these instantiations are not equivalent to the specifications of the application[3] the subsumption algorithm fails; thus it cannot be guaranteed that the database will supply meaningful data to the application. In the case of this example, the instantiations must be compared at run-time to determine if the database is providing meaningful data. Thus the subsumption algorithm can be used at compile-time to predict for each attribute whether the database will necessarily provide data with the correct semantics or if it will never provide data with the correct semantics. For attributes whose subsumption is not definable at compile-time, the algorithm can be used at run-time to determine if the database is providing the correct semantics.

There are a number of advantages to allowing the application metadata specification to define semantic equality. Most significant is that not all applications will have the same requirements for data semantics. For example, an application may require that two trade prices are in the same currency but may not be concerned about the trade price status of these trade prices. This application could define semantic equivalence simply as the *string-equivalence* of the currencies. With this definition the instantiations shown in Figure 11 are semantically equivalent and subsumption follows.

[3]The **and** condition between the two database instantiations in Figure 11 requires that both sets of instantiations satisfy the application specifications.

4.3 Conversion of Data Semantics

Allowing the application to specify the definition of semantic equivalence has another advantage; methods defined by the application to convert the data semantics can be included in these definitions. Additionally, methods defined in the DMD for the conversion of data semantics can be used by the application in the definition of semantic equivalence. For example, if the DMD contains a method, *convert-currency*, that converts one currency to another, then the application can define the semantic equivalence of values of currency as *convert-currency(currency$_D$, currency$_A$)*. Where there is equivalence between currencies if they are string equivalent or the currency defined in the DMD can be converted into the currency of the AMS. More general definitions of semantic equivalence can be defined using these types of *conversion* functions.

By including functions in the definition of semantic equivalence these definitions identify the means for conversion of data semantics. For example, the function *convert-currency* must have an implementation that provides the exchange rate for converting a trade price in one currency to a trade price in another currency. Changes in data semantics, at the database or in the application specification, can in some instances be offset by the use of such functions. For example, assume a database were to suddenly decide to change to reporting all trade prices in US dollars. Then an application, like the one specified in Figure 5 would be unaffected if semantic equivalence for currencies is based on the use of a currency conversion function. This still requires that at run-time a function exists to convert US dollars to pesetas.

4.4 Semantic Reconciliation and System Operation

The methods that have been described thus far can be used to identify semantic conflicts between a data source and an application. In some instances these methods can be used to resolve semantic conflicts. Additionally, these same methods can be used in a system where the source data semantics or the application data semantic specifications are allowed to change. If the connectivity between the application and the source are uneffected by these changes or if appropriate conversions can be made then the application can continue; otherwise the application must be notified of the incompatibility.

Ideally, semantic consistency between the database and application would be established when the AMS and DMD are defined. However, the demands for checking semantic consistency may require that the metadata be examined when individual data is retrieved. Additionally, the subsumption algorithm must be executed whenever there are changes in the AMS or the DMD. The coordination of the subsumption process requires that the system track which attributes are always meaningful, which ar never meaningful, and which have to be checked at run-time. The part of system operation is being carefully examined in the development of the system prototype.

The use of metadata for semantic reconciliation has been demonstrated in a restricted semantic definition environment. In the next section we examine the impact of relaxing some these restrictions.

5 More General Methods for Semantic Reconciliation

We have described methods for the use of metadata to identify and resolve semantic conflicts between an application and a database. These methods have provided a framework for more general

```
case Instrument_Type of
'Equity':       case Country of
                'spain' :       Trade_Price_Status := 'latest_nominal_price'
                                Currency := 'pesetas'
                endcase
endcase
```

Figure 13: Application Specification Using Additional Attributes

representation schemes for data semantics. For example, metadata specifications defined by the application can generalized by allowing the application to describe preconditions on attributes not found in the assignment domain of the DMD procedure. If this additional attribute is independent of attributes in the assignment domain in the DMD then the algorithm for subsumption is unchanged. However, if the additional attribute in the specification is correlated with any of the attributes in the assignment domain in the DMD then these correlations must be resolved. For example, consider the specification shown in Figure 13. The precondition contains two restrictions, the instrument must be an *equity* traded in *Spain.* Logically, this would mean any instrument traded on a Spanish stock exchange. It is necessary to change the restriction on Country, found in the application specification, into a restriction on Exchange. This change must be done by the DMD where it determines which exchanges are found in Spain. Then the subsumption algorithm can be applied to restrictions on the Exchange attribute. The algorithm for subsumption must include methods to identify these correlations and, when appropriate, change instances of attributes in the specification to instances of attributes found in the assignment domain of the procedure in the DMD.

A more rigorous definition of attributes defined as abstract data types [SM89b] can be used to simplify the process of locating correlated data types. Locating correlated data types may require inference rules to limit the possibility of unrestricted search for appropriate correlations. The coercion of data type instances must be done at the DMD, based on its knowledge about the relationship between data types (i.e., the DMD must use the database to evaluate this relationship).

The effect of relaxing assumptions in the representation model for data semantics is to increase system complexity and require that more of the process of semantic reconciliation be done at run-time. Restrictions on the complexity of the instantiations can improve system performance but may limit the ability to include general procedures for metadata definition. A more general approach to defining the DMD and AMS are considered as part of future research. In the next section we examine how the methods developed for the source-receiver model can be used for establishing semantic reconciliation in other models.

6 Semantic Reconciliation in Multidatabase Systems

Methods for semantic reconciliation using metadata have been presented for systems that conform to the source-receiver model. These results can easily be extended so that they are useful in multi-database systems. For example, an application might require data from a number of independent

data sources, where each source provides a unique set of data. For such a system, the AMS must identify which data source is expected to supply values of a given data type. The problem is then viewed in the source-receiver model between the specification in the AMS and the DMD for the database supplying instances of the data type. If we limit the systems to the conditions defined in Section 4.1, then the Subsume algorithm can be used to evaluate semantic consistency between an application and multiple data sources.

It is important to consider how these methods can be extended for semantic reconciliation in complex multidatabase systems. These methods can be used to provide a semantic interface to a loosely-coupled federated database system [JPL*88,LA86,She87]. A federated schema for such a system is a view over schemas of the component databases. Methods for semantic reconciliation can be used directly for the interface between an application and this federated schema. Each component database provides a database metadata dictionary for use in defining the federated database metadata dictionary in the federated schema. No schema integration is required in such a system.

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,Tem87]. 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.

The global schema is responsible for providing meaningful data to the application. The global schema includes a *global database metadata dictionary*, GDMD, that represents the semantic integration of a set of component databases. Methods for semantic reconciliation can be used directly between the application and the GDMD. The application sees the GDMD just as if it were a DMD for a single database. But unlike changing the data semantics in the DMD, the GDMD must react to changes in the semantics of a set of component databases as represented in the DMDs of these databases. As a result of these changes, the GDMD may be modified which may then affect the meaning of the data sent to the application. The GDMD can be defined by the integration of the component database DMDs using the union of the semantics provided by each DMD. The selection of a data source in this heterogeneous system will, in part, be determined by which source can provide data with the appropriate semantics.

The GDMD must be able to react to changes in the meaning of the data in the component databases assuring that the application can continue receiving meaningful data. In this way we extend the methods defined for semantic reconciliation in the source-receiver model to assist in the integration of multiple heterogeneous databases. Each component database will have a DMD and the GDMD will define the semantic integration of these component databases. The application specifications will play an active role in the selection of component databases, helping to dynamically locate meaningful data sources and appropriate translation (i.e., conversion) routines.

7 Summary

In this paper we describe methods for using metadata to automatically identify and resolve semantic conflicts between a data source and a receiver. When semantic connectivity is established, the receiver can be certain that the data supplied has the expected meaning. 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. In some instances, metadata provides a means for converting data semantics so that the source can provide meaningful data.

We describe 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 the representaion model for metadata, we show how an application can specify its requirements for data semantics. The definition of these semantic requirements is expressed as metadata including 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 AMS or DMD, in these definitions. These function can be used to automatically convert data semantics. Thus making it possible for the application to receive meaningful data from the source when such data could not normally be provided.

More general methods for semantic reconciliation are likely to require a more expressive representation, one that allows for data abstraction and encapsulation. Future work will consider the use of abstract data types [SM89b] as a means for generalizing these methods. This work will also consider the development of query languages and processing techniques that permit user access to both data and metadata.

The methods used for semantic reconciliation in the source-receiver model are also important in systems with multiple data sources. In a loosely-coupled federated database system these methods can be used to provide a single semantic interface to a federation of local databases. While in a tightly-coupled federated database system these methods can be used to define a semantically integrated global schema for multiple homogeneous or heterogeneous database systems. These methods can be used directly in the interface between an application and the global schema. Metadata can also be used in these systems to simplify schema integration among local databases. Changes that occur in the semantics of the underlying database must be detected and reflected as changes in the global schema. The semantic interface between the global schema and the local databases represents a significant part of our present research effort.

The need to express and manipulate metadata is important in source-receiver and multidatabase database systems. Previously, metadata has been used primarily for retrieval in large statistical databases. This paper describes how metadata can be used in the development of intelligent systems that can automatically react to changes in data semantics.

Acknowledgments

The authors would like to thank the members of the Composite Information Systems Laboratory at the Sloan School of Management for the numerous discussions pertaining to this topic. In addition, discussions with Robert Muller were helpful to us as we identified and analyzed this problem. 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.

[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.

[Cua87] Cuadra/Elsevier, editor. *Online Databases in the Securities and Financial Markets.* Elsevier Publishing Co., N.Y., N.Y., 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.

[JPL*88] G. Jakobson, G. Piatetsky-Shapiro, C. Lafond, M. Rajinikanth, and J. Hernandez. CALIDA: a system for integrated retrieval from multiple heterogeneous databases. In *Proceedings of the Third International Conference on Data and Knowledge Engineering*, pages 3–18, Jerusalem, Israel, 1988.

[LA86] A. Litwin and A. Abdellatif. Multidatabase interoperability. *IEEE Computer*, 10–18, December 1986.

[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.

[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.

[SM89a] M. Siegel and S. Madnick. *Identification and Resolution of Semantic Conflicts Using Metadata.* Technical Report #3102-89-MSA, Sloan School of Management, Massachusetts Institute of Technology, (Also submitted to SIGMOD 1990), 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.

[Tem87] M. Templeton et al. Mermaid - a front-end to distributed heterogeneous databases. In *Proceedings of the IEEE*, pages 695–708, 1987.

[WM88] R. Wang and S. Madnick, editors. *Connectivity Among Information Systems.* Volume 1 of *Composite Information Systems (CIS) Project*, Massachusetts Institute of Technology, 1988.