Representation and Reasoning about Changing Semantics in Heterogeneous Data Sources

(SWDB/VLDB)

Hongwei Zhu
Stuart E. Madnick
Michael D. Siegel

Working Paper CISL# 2004-14

August 2004

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E53-320
Massachusetts Institute of Technology
Cambridge, MA 02142

Representation and Reasoning about Changing Semantics in Heterogeneous Data Sources

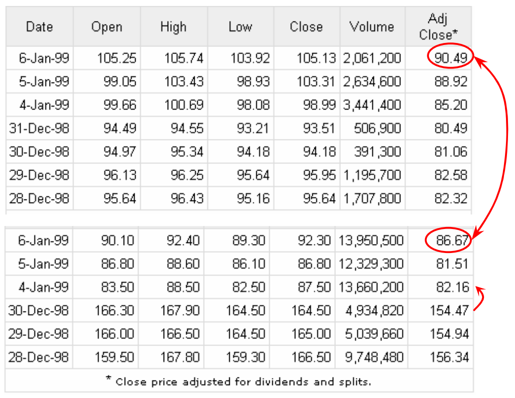
Hongwei Zhu, Stuart E. Madnick, Michael D. Siegel

MIT Sloan School of Management
30 Wadsworth Street, MA, 02142, USA
{mrzhu, smadnick, msiegel}@mit.edu
http://interchange.mit.edu/coin

Abstract. Changes of semantics in data sources further complicate the semantic heterogeneity problem. We identify four types of semantic heterogeneities related to changing semantics and present a solution based on an extension to the Context Interchange (COIN) framework. Changing semantics is represented as multi-valued contextual attributes in a shared ontology; however, only a single value is valid over a certain time interval. A mediator, implemented in abductive constraint logic programming, processes the semantics by solving temporal constraints for single-valued time intervals and automatically applying conversions to resolve semantic differences over these intervals. We also discuss the scalability of the approach and its applicability to the Semantic Web.

1 Introduction

The Web has become a large database, from which obtaining meaningful data is becoming increasingly difficult. As a simple example, try querying historic stock prices for Daimler-Benz from Yahoo. Figure 1 shows what Yahoo returned for the prices at stock exchanges in New York and Frankfurt.



Date	Open	High	Low	Close	Volume	Adj Close*
6-Jan-99	105.25	105.74	103.92	105.13	2,061,200	90.49
5-Jan-99	99.05	103.43	98.93	103.31	2,634,600	88.92
4-Jan-99	99.66	100.69	98.08	98.99	3,441,400	85.20
31-Dec-98	94.49	94.55	93.21	93.51	506,900	80.49
30-Dec-98	94.97	95.34	94.18	94.18	391,300	81.06
29-Dec-98	96.13	96.25	95.64	95.95	1,195,700	82.58
28-Dec-98	95.64	96.43	95.16	95.64	1,707,800	82.32

6-Jan-99	90.10	92.40	89.30	92.30	13,950,500	86.67
5-Jan-99	86.80	88.60	86.10	86.80	12,329,300	81.51
4-Jan-99	83.50	88.50	82.50	87.50	13,660,200	82.16
30-Dec-98	166.30	167.90	164.50	164.50	4,934,820	154.47
29-Dec-98	166.00	166.50	164.50	165.00	5,039,660	154.94
28-Dec-98	159.50	167.80	159.30	166.50	9,748,480	156.34

* Close price adjusted for dividends and splits.

Fig. 1. Stock prices for Daimler-Chrysler from Yahoo. Top: New York; Bottom: Frankfurt

What conclusions will you draw from the retrieved data? Perhaps you regret that you were not arbitraging the substantial price differences between the exchanges, or feel lucky that you sold the stock in your Frankfurt account at the end of 1998? Both conclusions are wrong. Here, not only are the currencies for stock prices different at the two exchanges, but the currency at Frankfurt exchange also changed from German Marks to Euros at the beginning of 1999 (the New York exchange remained as US dollars). Once the data is transformed into a uniform context, e.g., all prices in US dollars, it can be seen that there is neither significant arbitraging opportunity nor abrupt price plunge for this stock.

The example illustrates the kinds of problems that the Semantic Web aims to solve. We need not wait until the full implementation of the Semantic Web for meaningful data retrieval. Context Interchange (COIN) framework [7, 10, 11], originated from the semantic data integration research tradition, shares this goal with the Semantic Web research. With the recent temporal extension that processes heterogeneous and changing semantics, described in this paper, COIN provides an extensible and scalable solution to the problem of identifying and resolving semantic heterogeneities. COIN is a web-based mediation approach with several distinguishing characteristics:

- Detection and reconciliation of semantic differences are system services and are transparent to users. Thus with COIN, the historic stock prices in the simple example are automatically transformed before they are returned to the user;
- Mediation does not require that semantic differences between each source-receiver pair to be specified a priori, rather, it only needs a declarative description of each source's data semantics and the methods of reconciling possible differences. Semantic differences are detected and automatically reconciled at the time of query. Scalability is achieved by the use of ontology, context inheritance, and parameterization of conversion functions;
- Mediation is implemented in abductive constraint logic programming. As a result, it allows for knowledge level query and can generate *intensional* answers as well as *extensional* answers. Efficient reasoning is achieved by combining abduction with concurrent constraint solving.

In this paper, we will focus on the representation and reasoning of changing semantics in COIN. Since it is an extension to early implementations, it is capable of mediating static semantic heterogeneities as well as those that change over time.

2 Temporal Semantic Heterogeneities

Temporal semantic heterogeneities refer to the situation where the semantics in data sources and/or receivers changes over time. We categorize them into four types, which are described below, then followed by an illustrative example.

2.1 Categories of Temporal Semantic Heterogeneities

Representational heterogeneity. The same concept can be represented differently in different sources and during different time periods. In the stock price example, the same concept of stock price is represented in different currencies, and for the Frankfurt exchange, the currency also changed in the beginning of 1999. This introduces representational temporal heterogeneity when the receiver needs price data in US dollars. Representational heterogeneity often results from the differences in unit of measures, scale factors, and other syntactic characteristics.

Ontological heterogeneity. The same term is often used to refer to slightly different concepts; in the same source the concepts referred to by a term may shift over time, which introduces ontological temporal heterogeneities. For example, *profit* can refer to *gross profit* that includes all taxes collected on behalf of government, or *net profit* that excludes those taxes. The referred concept may shift from one type of profit to another because of change of reporting rules.

Aggregational heterogeneity. When the data of interest is an attribute of an entity that can consist of other entities, aggregational heterogeneity arises if the component entities vary in different situations. A detailed example is presented in [17], where depending on regulatory requirements and purposes, the financial data of corporations may or may not include those from certain branches, subsidiaries, and majority owned foreign joint ventures. These rules and purposes may change over time, which introduces aggregational temporal heterogeneities. We will give a more detailed example of this category later in this section.

There are certain connections between ontological and aggregational heterogeneities. For example, the question "*does profit include taxes*" concerns ontological heterogeneity; while the question "*does profit for corporation x include that of its subsidiaries*" concerns aggregational heterogeneity. The latter can be seen as a more complicated version of the former in that the heterogeneity results from the entity that the data is about, not the data itself. In addition, data aggregation rules are often more complicated than ontological concept definitions. We will use this connection in COIN to encode and process aggregational heterogeneity.

Heterogeneity in temporal entity. The representation and operations for the domain of time vary across systems. As a result, there exist heterogeneities that include, for example, location dependencies such as time zones, differences in representation conventions, calendars, and granularities. Although it is a type of representational heterogeneity, we treat it as a special category because of the complexity of the domain of temporal entity.

2.2 An Illustrative Example

We use the following example to illustrate representational and aggregational heterogeneities. Readers are referred to [17] for a more complicated aggregational example and to [21] for an example of representational and ontological heterogeneities.

The example involves two sources and one receiver[1]. The receiver is interested in the longitudinal economic and environmental changes in the Balkans area, before and after the war in Yugoslavia. As shown in Figure 2, the sources organize the data by sovereign country, while the receiver is interested in data for the region covered by the former Yugoslavia. The sources also make other implicit assumptions for data in terms of currencies and scale factors. We call these assumptions for interpreting the data *contexts* and identify them using context labels, *c_srs* and *c_target*, in Figure 2.

As the web and traditional databases are often used today, the receiver knows what data is available in the sources and wants to query them directly using query Q – but the user may not know about (nor want to deal with) the differences in contexts. Thus, a direct execution of query Q over the sources would bring back wrong answers because the query does not consider context differences, such as those in currency and scale factor. Additionally, in 1992 former Yugoslavia was divided into five sovereign countries, each with its own distinctive currency (See Table 1). Therefore, the results for data after 1992 are also wrong because of unresolved aggregational differences, i.e., the data represents only a sub-area of the entire region expected by the receiver.

Context *c_src*

1. *Monetary values* are in official currency of the country, with a scale factor of 1M;
2. *Mass* is a rate of tons/year with a scale factor of 1000;
3. All *other numbers* have a scale factor of 1;
4. All values are aggregated by sovereign country.

Schema of source 1:
```
    Statistics(Country, Year, GDP, Population)
```

Schema of source 2:
```
    Emissions(Country, Year, CO2)
```

Context *c_target*

1. *Monetary values* are always in USD, with a scale factor of 1M;
2. *Mass* is in tons/year;
3. *Other numbers* have a scale factor of 1;
4. Value for country denoted by 'YUG' is aggregated to the geographic area of former Yugoslavia.

Query Q[2]:
```
Select S.Country,S.Year,GDP,CO2
From Statistics S, Emissions E
Where S.Country=E.Country and
S.Year=E.Year and S.Country='YUG';
```

Fig. 2. Temporal context example, with subtle changes in data semantics

Table 1. Five countries resulting from the division of the former Yugoslavia

Country	Code	Currency	Currency Code
Yugoslavia[3]	YUG	New Yugoslavian Dinar	YUM
Bosnia and Herzegovina	BIH	Marka	BAM
Croatia	HRV	Kuna	HRK

[1] This example has been simplified in this paper to reduce space while maintaining the key details. The actual situation involves many more sources as well as multiple users, each with a potentially different context.

[2] For this example and demonstration to follow, the Query Q is expressed in the Structured Query Language (SQL). The basic COIN approach can be applied to any query language.

[3] The Federal Republic of Yugoslavia was renamed Serbia and Montenegro in 2003. We will not encode this change in the example to simplify illustration.

Macedonia	MK D	Denar	MKD
Slovenia	SVN	Tolar	SIT

Compared to the stock quote example, the semantic changes in this example are more subtle in that there seem to be no semantic changes in the verbal context descriptions in Figure 2, it is the meaning of the country code 'YUG' that changes over time. To account for this change, we need to make it explicit either in the source context or in the receiver context. We choose the latter in the following discussions. In addition, the aggregational heterogeneity also dynamically introduces new representational heterogeneities, e.g., currency differences will be encountered in aggregating data for each component country. Another interesting characteristic is that in this simple example the two sources share the same context (in reality, there are likely many context differences amongst the diverse sources).

3 COIN Framework and Architecture

The COIN framework consists of a deductive object-oriented data model for knowledge representation and a general purpose mediation service module that detects and resolves semantic conflicts in user queries at run-time (see Figure 3).

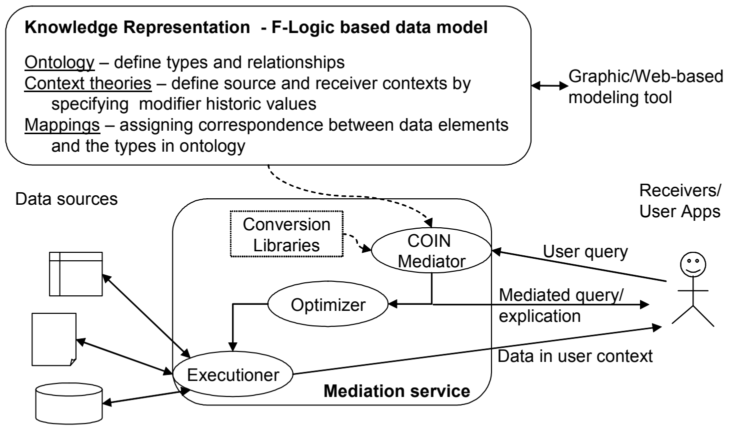

Knowledge Representation - F-Logic based data model
Ontology – define types and relationships
Context theories – define source and receiver contexts by specifying modifier historic values
Mappings – assigning correspondence between data elements and the types in ontology
Graphic/Web-based modeling tool
Data sources
Receivers/ User Apps
Conversion Libraries
COIN Mediator
User query
Optimizer
Mediated query/ explication
Data in user context
Executioner
Mediation service

Fig. 3. Architecture of the COIN system

Knowledge representation in COIN consists of three components:

– Ontology – to define the semantic domain using a collection of semantic types and their relationships. A type corresponds to a concept in the problem domain and can be related to another in three ways: 1) as a subtype or super-type (e.g., *profit* is a subtype of *monetary value*; 2) as a named attribute (e.g., *temporal entity* such as year is a temporal attribute of *GDP*); and 3) as a modifier or contextual attribute, whose value is specified in context axioms and can functionally determine the interpretation of instances of the type that has this modifier (e.g., *monetary value* type has a *scale factor* modifier). There is a distinguished type *basic* in the ontology that serves as the super type of all the other types and represents all primitive data types. Objects are instances of the defined types;

– Context theories – to specify the values of modifiers for each source or receiver and the conversions for transforming an object in one context to another. The context of each source or receiver is uniquely identified with a context label, e.g., *c_src* and *c_target* in the example. The value specification for modifiers can be a simple value assignment or a set of rules that specify how to obtain the value. Conceptually a context can be thought to be a set of <*modifier*, *object*> pairs, where *object* is a singleton in most non-temporal cases; and
– Semantic mappings – to establish correspondences between data elements in sources and the types in the ontology, e.g., *GDP* in the example in Figure 2 corresponds to *monetary value* in the ontology.

These components can be expressed in OWL or F-Logic [15]. Since COIN predates OWL, F-Logic was the language of choice because of its rich constructs for describing types and their relationships and has formal semantics for inheritance and overriding. In practice, we translate the F-Logic expressions into a Horn logic program and implement the semantics of inheritance and overriding in the context mediator component described next. For succinctness, we continue to use the F-Logic syntax in the rest of the paper. Attributes and modifiers are represented as functions or methods of the defined types; since modifier values vary by context, methods for modifiers are parameterized with a context label. Comparison between objects is only meaningful when performed in the same context, i.e., suppose *x* and *y* are objects,

$$x \overset{c}{\Diamond} y \Leftrightarrow x[value\ (c) \rightarrow u] \wedge y[value\ (c) \rightarrow v] \wedge u \Diamond v.$$

where $\Diamond$ is one of the comparison operators for primitives in $\{=, \neq, <, \leq, >, \geq, \ldots\}$, and the *value* method is a parameterized function that returns the primitive value of an object. A *value* method call invokes the comparison of modifier values in source context and receiver context *c*, if difference is detected, conversion functions are invoked.

The core component in the mediation service module is the COIN mediator implemented in abductive constraint logic programming. It takes a user query and produces a set of mediated queries ($\mathcal{MQ}$s) that resolve semantic differences. This happens by first translating the user query into a Datalog query and using the encoded knowledge to derive the $\mathcal{MQ}$s that incorporate necessary conversions from source contexts to receiver context. The query optimizer and processor [2] optimize the $\mathcal{MQ}$s using a simple cost model and the information on source capabilities, obtain the data, perform the conversions, and return final datasets to the user.

We also developed web-based [16] and graphical [13] tools for data administrators to design ontologies and input context knowledge. As part of ongoing effort of connecting COIN with the Semantic Web, we are also developing OWL and RuleML based representations for the COIN ontology and context knowledge; a prototype is described in [19]. These prototypes also translate the captured knowledge into Prolog syntax required by the current implementation of the mediation service.

4 Representation of Changing Semantics

Like many existing ontologies, previously the ontologies in COIN were based on a snapshot view of the world and lacked the capability of capturing changing semantics. To overcome this limitation, we incorporate in COIN ontologies explicit time concepts such as the ones defined in DAML Time Ontology [12]. *Temporal entity* is the most general concept and can be further specialized into *instant* and *interval*. There is emerging research that aims to systematically temporalize static ontologies [18]; for simple ones, we can manually create a temporal ontology by relating concepts whose value or semantics changes over time to temporal concepts via named attributes. Figure 4 shows a graphical representation of the ontology for the example.

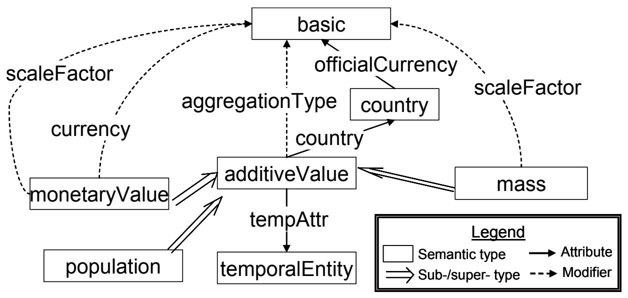


Fig. 4. A graphical representation of the example ontology

We should note that, like other types of conceptual modeling, there could be multiple variants of the example ontology that convey the same ideas in the problem domain, some of which may be even better. We use this one for illustration purposes.

The type *basic* represents system primitives and is the parent of all the other types; this relationship is omitted in the graph to eliminate clutter. The context regarding data aggregation is captured by the modifier *aggregationType* of the semantic type *additiveValue*, which serves as the parent all other types that can be aggregated. Attributes are also added to this parent type to relate to types *country* and *temporalEntity*. Through inheritance and overriding the child types obtain and specialize, if necessary, these relationships and context specifications.

In the following discussion, we assume certain familiarity with the syntax and the semantics of F-Logic. Following Gulog [5], a sub-language of F-Logic, we use $\vdash$ to separate variable type declarations from the rest of a formula. For succinctness we also use non-clausal forms, which eventually will be rewritten in clausal form and translated into equivalent Horn logic clauses.

We represent a named attribute in the ontology as a function and a modifier as a multi-valued method parameterized with context label. For example, the following formula declares that *additiveValue* has *tempAttr* with a return type of *temporalEntity*:

$$additiveValue[tempAttr \Rightarrow temporalEntity].$$

Similarly, the following declares modifier *aggregationType* for *additiveValue*:

$$additiveValue[aggregationType(ctxt) \Rightarrow\!\!\Rightarrow basic].$$

The changing semantics is represented by context axioms that specify the entire history of multi-valued modifiers. In the example we have decided to make the semantic change explicit in the receiver context, which means that before the balkanization in 1991, the receiver aggregates data at the country level for Yugoslavia, and for data after 1992, the receiver wants to aggregate at the level of geo-region covered by former Yugoslavia. This is expressed using the following context axiom:

$$\begin{aligned}
&\forall X : additiveValue\ \exists Y : basic \vdash \\
&\quad X[aggregationType(c\_target) \rightarrow Y] \wedge X[country \rightarrow C] \wedge C \overset{c\_target}{=} \text{'}YUG\text{'} \wedge \\
&\quad (Y[value(c\_target) \rightarrow \text{'}country\text{'}] \leftarrow X[tempAttr \rightarrow T] \wedge T \overset{c\_target}{\in_t} I_{\le 1991}) \wedge \\
&\quad (Y[value(c\_target) \rightarrow \text{'}georegion\text{'}] \leftarrow X[tempAttr \rightarrow T] \wedge T \overset{c\_target}{\in_t} I_{1992\le}).
\end{aligned} \qquad \textbf{(1)}$$

where $I_{\le 1991}$ represents the time interval up to year 1991, similarly $I_{1992\le}$ is the interval since year 1992, and $\in_t$ is a temporal inclusion relation, which can be translated into a set of comparisons between time points by introducing functions that return the beginning and ending points of an interval:

$$\begin{aligned}
&\forall T : temporalEntity, I : temporalEntity \vdash \\
&\quad T \overset{c}{\in_t} I \Leftrightarrow (begin(I) \overset{c}{\le} begin(T)) \wedge (end(T) \overset{c}{\le} end(I)).
\end{aligned}$$

Conceptually, we can think of temporal context as a set of <*modifier*, *history*> pairs with *history* being a set of <*object*, *time_interval*> pairs or a set of time-stamped objects. By abusing syntax, we say the temporal context pertaining to aggregation for the receiver is $< aggregationType, \{<\text{'}country\text{'}, I_{\le 1991}>,\ <\text{'}georegion\text{'}, I_{1992\le}>\} >$.

A multi-valued modifier is still single-valued over a certain time interval. Thus, there exist overlapping intervals over which all involved modifiers are single-valued in each context. Within these intervals, the determination and the resolution of context differences are identical to those in a snapshot model. Therefore conversion functions for processing changing semantics are identical to those in snapshot COIN. For the interval after 1992, semantic differences exist between the sources and the receiver in terms of aggregation because the value of *aggregationType* modifier in the sources is '*country*', while it is '*georegion*' in the receiver context. Conversions are needed to resolve this difference. Conversion functions are defined for the subtypes of *additiveValue*, e.g., for *moentaryValue* we have:

$$\begin{aligned}
&X : monetaryValue \vdash \\
&\quad X[cvt(aggregationType, c\_target) @ c\_src, u \rightarrow v] \leftarrow \\
&\quad\quad X[tempAttr \rightarrow T] \wedge statistics\ \_(C_1, T_1, M_1, \_) \wedge statistics(C_2, T_2, M_2, \_) \wedge \\
&\quad\quad statistics\ \_(C_3, T_3, M_3) \wedge statistics\ \_(C_4, T_4, M_4, \_) \wedge \\
&\quad\quad C_1 \overset{c\_target}{=} \text{'}BIH\text{'} \wedge C_2 \overset{c\_target}{=} \text{'}HIV\text{'} \wedge C_3 \overset{c\_target}{=} \text{'}MKD\text{'} \wedge C_4 \overset{c\_target}{=} \text{'}SVN\text{'} \wedge T_1 \overset{c\_target}{=} T \wedge \\
&\quad\quad T_2 \overset{c\_target}{=} T \wedge T_3 \overset{c\_target}{=} T \wedge T_4 \overset{c\_target}{=} T \wedge M_1[value(c\_target) \rightarrow m_1] \wedge \\
&\quad\quad M_2[value(c\_target) \rightarrow m_2] \wedge M_3[value(c\_target) \rightarrow m_3] \wedge \\
&\quad\quad M_4[value(c\_target) \rightarrow m_4] \wedge v = u + m_1 + m_2 + m_3 + m_4.
\end{aligned} \qquad \textbf{(2)}$$

The function essentially states that to process *aggregationType* semantic difference of a *monetaryValue* object, its value *u* in *c_src* context is converted to a value *v* in *c_target* context by finding the other *monetaryValue* objects that correspond to the four other countries as indicated by the codes at the same year, convert them into primitive values in *c_target*, and make *v* the sum of these primitives. The function for *mass* subtype is similarly defined. Before calling this function, functions that convert for *currency* and *scaleFactor* should have been called to arrive at this interim value *u*; in calling *value* functions for the four objects C_1-C_4 in the body of the function, conversions for *currency* and *scaleFactor* are dynamically called to ensure that they are summed in the same context. Thus, it is necessary to specify the precedence of modifiers, i.e., the order in which the modifiers should be processed.

This function is not general because of the use of constants in places of context parameters and the references to a semantic relation corresponding to a relation in data source. Using COIN for aggregational heterogeneities is a new research area that we are currently investigating to produce general methodology. For most other types of heterogeneities, though, more general conversions exist and can be utilized in multiple problem domains by collecting them into the conversion library. Generalization is achieved by parameterizing the function with variables for context labels. For example, the following currency conversion function can be used to convert monetary values from any arbitrary context C_1 to any other arbitrary context C_2:

$$\begin{aligned}
&X : monetaryValue \vdash \\
&\quad X[cvt(currency, C_2) @ C_1, u \to v] \leftarrow \\
&\qquad X[currency(C_1) \to C_f] \wedge X[currency(C_2) \to C_t] \wedge X[tempAttr \to T] \wedge \\
&\qquad olsen\_(A, B, R, D) \wedge C_f \overset{C_2}{=} A \wedge C_t \overset{C_2}{=} B \wedge T \overset{C_2}{=} D \wedge \\
&\qquad R[value(C_2) \to r] \wedge v = u * r.
\end{aligned} \qquad \mathbf{(3)}$$

where *olsen_* corresponds to an external relation that gives exchange rate between two currencies on any specified date.

A recent effort [8] introduced automatic conversion composition based on equational relationships between contexts, e.g., given conversions 1) between base price and tax-included price; and 2) between tax-included price and final price, the conversion between base price and final price can be composed using symbolic equation solvers.

5 Reasoning about Changing Semantics in COIN Mediation

The mediator is to translate a user query that assumes everything is in user context to a set of $\mathcal{MQ}$s that reconcile context differences between each involved source and the user. The following pseudo code sketches the intuition of the procedure:

```
For each source attribute appearing in user query
  Instantiate into object of type in ontology according to mappings
  Find the direct and inherited modifiers
  Order modifiers according to precedence
  For each modifier
    Choose a value and put corresponding temporal constraint in store
    If constraints are consistent
      Compare values in source and receiver
      If different, call conversion function and put abducibls in store
Construct MQ using abducibles
```

We implement this procedure using abductive constraint logic programming (ACLP) [14]. Briefly, ACLP is a triple <$\mathcal{P}$, $\mathcal{A}$, $\mathcal{IC}$>, where $\mathcal{P}$ is a constraint logic program, $\mathcal{A}$ is a set of abducible predicates different from the constraint predicates, and $\mathcal{IC}$ is a set of integrity constraints over the domains of $\mathcal{P}$. Query answering in ACLP is that given a query $q(\vec{X})$, generate a set of abductive hypothesis Δ and a substitution θ so that $\mathcal{P} \bigcup \Delta$ entails $q(\vec{X})\theta$ and is consistent; Δ consists of abducible predicates and simplified constraints.

The COIN framework can be straightforwardly mapped to ACLP. Knowledge representation in COIN can be translated into an equivalent normal Horn program [1]; or alternatively, the knowledge representation can be directly expressed in first order Horn clauses. This corresponds to $\mathcal{P}$. Predicates and arithmetic operators allowed by the query languages of sources and other callable external functions constitute $\mathcal{A}$. $\mathcal{IC}$ consists of integrity constraints in data sources and any constraints introduced in the user query. The user query corresponds to query $q(\vec{X})$ in ACLP; the $\mathcal{MQ}$s are constructed from the set Δ and the substitution θ.

Abductive inference in COIN is a modified SLD-resolution [6] in that literals corresponding to predicates in data sources are abducted without evaluation; constraints are abducted and subsequently propagated/simplified. The constraints include basic arithmetic comparisons, equational relationships among arithmetic operators for conversion composition [8], and temporal relations for processing changing semantics. The mediator is implemented using constraint logic programming environment ECLiPSe [20] with the extension of Constraint Handling Rules (CHR) [9]. Naturally, we use the constraint store to collect the abucibles. At the end of a successful derivation, an $\mathcal{MQ}$ is constructed using the predicates and constraints collected in the constraint store.

As shown earlier, context axioms for multi-valued modifiers contain temporal inclusion comparison, which can be transformed to a conjunction of comparisons of the end points using comparison relation $\leq$. We implement temporal relations as a constraint *tle*, with *tle(X, Y)* meaning temporal entity *X* is *before* (inclusive) *Y*. Here *X* and *Y* are variables of primitive temporal entities in the same context. Similar to semantic relations, we use *tle_* for constraint over semantic objects.

In the process of mediation, temporal constraints appearing in a context axiom are abducted into the constraint store after the head of the axiom clause is unified with the goal atom. Applicable CHR rules are triggered immediately after abduction to simply or propagate the constraints. Inconsistency of the constraint store signifies a failure and causes backtracking. The temporal constraints in the consistent store after a successful derivation determine the common interval over which all involved modifiers are single-valued. Suppose the query language in source accepts $\leq$ for temporal entity comparison, the *tle* constraint in the store is translated back to $\leq$ to construct the $\mathcal{MQ}$s.

The context axiom (2) corresponds to two clauses after it is rewritten in the clausal form; each clause contains temporal constraints corresponding to the interval for pre- or post-balkanization of former Yugoslavia. When the user query in Figure 2 is mediated, each clause posts a temporal constraint into the store and produces a successful derivation. No inconsistency is produced in the example; refer to [21, 22] for an example where inconsistencies occur when multiple modifiers change values at different times. The $\mathcal{MQ}$s in Datalog syntax are a disjunction of two sub-queries; each deals with a time interval in context axiom (2):

```
answer('V8', 'V7', 'V6', 'V5') :-
     'V5' is 'V4' * 1000.0,
     olsen("YUM", "USD", 'V3', 'V7'),
     'Statistics'("YUG", 'V7', 'V2', 'V1'),
     'Emissions'("YUG", 'V7', 'V4'),
     'V7' =< 1991,
     'V6' is 'V2' * 'V3'.
answer('V96', 'V95', 'V94', 'V93') :-
     'V92' is 'V91' * 1000.0, 'V90' is 'V89' * 1000.0,
     'V88' is 'V87' * 1000.0, 'V86' is 'V85' * 1000.0,
     'V84' is 'V83' * 1000.0, 'V82' is 'V90' + 'V92',
     'V81' is 'V88' + 'V82', 'V80' is 'V86' + 'V81',
     'V93' is 'V84' + 'V80', 'V79' is 'V78' * 'V77',
     'V76' is 'V75' * 'V74', 'V73' is 'V72' * 'V71',
     'V70' is 'V69' * 'V68', olsen("SIT", "USD", 'V67', 'V95'),
     Statistics'("SVN", 'V95', 'V66', 'V65'),
     olsen("MKD", "USD", 'V68', 'V95'),
     'Statistics'("MKD", 'V95', 'V69', 'V64'),
     olsen("HRK", "USD", 'V71', 'V95'),
     'Statistics'("HRV", 'V95', 'V72', 'V63'),
     olsen("BAM", "USD", 'V74', 'V95'),
     'Statistics'("BIH", 'V95', 'V75', 'V62'),
     olsen("YUM", "USD", 'V77', 'V95'),
     'Emissions'("SVN", 'V95', 'V83'),
     'Emissions'("MKD", 'V95', 'V85'),
     'Emissions'("HRV", 'V95', 'V87'),
     'Emissions'("BIH", 'V95', 'V89'),
     'Statistics'("YUG", 'V95', 'V78', 'V61'),
     'Emissions'("YUG", 'V95', 'V91'),
     1992 =< 'V95', 'V60' is 'V66' * 'V67',
     'V59' is 'V76' + 'V79', 'V58' is 'V73' + 'V59',
     'V57' is 'V70' + 'V58', 'V94' is 'V60' + 'V57'.
```

The first sub-query corresponds to the pre-balkanization period, when there is no aggregational difference between the sources and the receiver; only representational differences for GDP and CO_2 emissions exist. The second sub-query corresponds to the post-balkanization period, when both aggregational differences for GDP and CO_2 emissions and representational differences exist. Note the currency conversions dynamically introduced in the conversion for aggregational difference have been properly incorporated.

These $\mathcal{MQ}$s are returned to the user as an intensional answer to the original query. With all conversions declaratively defined, these $\mathcal{MQ}$s in fact convey a great deal of useful information to the user. The $\mathcal{MQ}$s are then passed to the optimizer and executioner components that access actual data sources to return final data to the user.

6 Discussion

It is common that data semantics in sources and receivers changes over time. One can draw wrong conclusions when these changes are not adequately represented and properly processed. In this paper, we identified four categories of semantic heterogeneities related to changing semantics. We also presented recent results that extend the COIN framework for representing and processing changing semantics, with a focus on the treatment of aggregational temporal heterogeneities.

As we explained in [21], the COIN framework is applicable to the Semantic Web for several reasons. The use of SQL in the example should not be construed as a limitation. In fact, the data model, the representation language, and the ACLP implementation of COIN mediator are all logic based. Thus, the framework can be applied broadly. To adapt to another query language, we can simply include the logic form of the language constructs as abducibles. More importantly, semantic differences are automatically detected and resolved using declarative descriptions of semantics. This approach is very well in line with the Semantic Web, where each source furnishes a description of its semantics for agents from other contexts to process. In addition, a recent extension to the basic COIN system added an ontology merging capability to allow large applications to be built by merging separate ontologies [7]. This is very similar to how agents work with distributed ontologies on the Semantic Web. Lastly, we are also experimenting with OWL and RuleML based languages for ontology and conversion function representation. These formalisms will allow the COIN mediation service to process other autonomously created ontologies on the Semantic Web.

The COIN framework is also scalable. The number of required conversion functions does not depend on the number of sources/receivers involved; rather, it depends on the variety of contexts, i.e., number of modifiers and their unique values. With context inheritance (e.g., the two sources in the example share the same context, but partial sharing is also possible), parameterization of conversions functions (e.g., the conversion function for currency differences is applicable to any pair of contexts having different currencies), and conversion composition, the number of conversion functions required grows only when the addition of sources introduces new contexts and the existing conversion functions do not handle the conversions for the new contexts. In the illustrative example, suppose we add another two hundred additional sources that have semantic differences with the receiver only in terms of currency and scale factor, the number of conversion functions remains unchanged because the existing conversion functions can process these new contexts. Adding these new sources only involves adding context declarations and semantic mappings, which are declaratively defined.

We acknowledge that the conversion functions for processing aggregational heterogeneities in the illustrative example are not as general. As future research, we plan to investigate ontology modeling techniques and other representational constructs that may help generalize conversion functions of this type. We also note that the heterogeneities in temporal entities are not currently processed in the prototype. As another future research area, we plan to introduce the full Time ontology into knowledge representation and implement conversions as external function calls to web services that specifically handle time zones, calendars, and granularities [3, 4].

Acknowledgements

The work reported herein was supported, in part, by the Singapore-MIT Alliance (SMA) and the Malaysia University of Science and Technology (MUST)-MIT collaboration.

References

1. S. Abiteboul, G. Lausen, H. Uphoff, E. Waller, "Methods and Rules", SIGMOD Rec., 22(2), pp. 32-41, 1993.
2. T. Alatovic, "Capabilities Aware Planner/Optimizer/Executioner for COntext INterchange Project", MS Thesis, MIT, 2002.
3. C. Bettini, "Web services for time granularity reasoning," TIME-ICTL'03, 2003.
4. C. Bettini, S. Jajodia, and X. S. Wang, Time Granularities in Databases, Data Mining, and Temporal Reasoning: Springer, 2000.
5. Dobbie, G., and Topor, R. "On the Declarative and Procedural Semantics of Deductive Object-Oriented Systems," Journal of Intelligent Information Systems (4), 1995, pp 193-219.
6. K. Eshgi, Kowalski, R. "Abduction Compared with Negation as Failure", Proceedings of 6th Intl Conf. on Logic Programming, 1989.
7. Firat, "Information Integration using Contextual Knowledge and Ontology Merging," PhD Thesis, MIT, 2003.
8. A. Firat, S. Madnick, B. Grosof, "Financial information integration in the presence of equational ontological conflicts", WITS, 2002.
9. T. Frühwirth, "Theory and Practice of Constraint Handling Rules," Journal of Logic Programming, 37, pp. 95-138, 1998.
10. C.Goh, "Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems", PhD Thesis, MIT, 1997
11. C. Goh, S. Bressan, S. Madnick, and M. Siegel, "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information," ACM TOIS, vol. 17, pp. 270-293, 1999.
12. J. R. Hobbs, "A DAML Ontology of Time," LREC, 2002.
13. S. Jayasena, "Context Mediation Approach to Improved Interoperability amongst Disparate Financial Information Services", MS Thesis, Singapore-MIT Alliance CS Program, 2004.
14. A.C. Kakas, A. Michael, and C. Mourlas, "ACLP: Integrating Abduction and Constraint Solving," Journal of Logic Programming, 44, pp. 129-177, 2000.
15. M. Kiffer, G. Laussen, J. Wu, "Logic Foundations of Object-Oriented and Frame-based Languages", J. ACM, 42(4), pp. 741-843, 1995.
16. P.W. Lee, "Metadata Representation and Management for Context Mediation", MS Thesis, MIT, 2003.
17. S. Madnick, R. Wang, X. Xian, "The Design and Implementation of a Corporate Householding Knowledge Processor to Improve Data Quality", JMIS, 20(3), pp. 41-69, 2004.
18. J. Santos, S. Staab, "FONTE: Factorizing Ontology Engineering Complexity," International Conference On Knowledge Capture, ACM Press, Sanibel Island, FL, USA, 2003, pp. 146-153.
19. P.E.K. Tang, S. Madnick, K.L. Tan "Context Mediation in the Semantic Web: Handling OWL Ontology and Data Disparity through Context Interchange", Second International Workshop on Semantic Web and Database (in this volume), Toronto, Canada, 2004.

20. M. Wallace, S. Novello, J. Schimpf, " ECLiPSe: A Platform for Constraint Logic Programming", IC-Parc, Imperial College, London, August 1997.
21. H. Zhu, S.E. Madnick, M.D. Siegel, "Reasoning about Temporal Context using Ontology and Abductive Contraint Logic Programming", to appear in Proceedings of Practice and Principles of Semantic Web Reasoning (PPSWR'04), St. Malo, France, September 2004.
22. H. Zhu, S.E. Madnick, M.D. Siegel, "Effective Data Integration in the Presence of Temporal Semantic Conflicts", Proceedings of 11th International Symposium on Temporal Representation and Reasoning (TIME 2004), pp109-114, Normandie, France, July 1-3, 2004