Reasoning About Temporal Context Using Ontology and Abductive Constraint Logic Programming

Paper 211

Hongwei Zhu
Stuart E. Madnick
Michael D. Siegel

August 2004

Reasoning about Temporal Context using Ontology and Abductive Constraint Logic Programming
(PPSWR/ICLP)

Hongwei Zhu
Stuart E. Madnick
Michael D. Siegel

Working Paper CISL# 2004-15

August 2004

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

Reasoning about Temporal Context using Ontology and Abductive Constraint Logic Programming

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. The underlying assumptions for interpreting the meaning of data often change over time, which further complicates the problem of semantic heterogeneities among autonomous data sources. As an extension to the Context Interchange (COIN) framework, this paper introduces the notion of temporal context as a formalization of the problem. We represent temporal context as a multi-valued method in F-Logic; however, only one value is valid at any point in time, the determination of which is constrained by temporal relations. This representation is then mapped to an abductive constraint logic programming framework with temporal relations being treated as constraints. A mediation engine that implements the framework automatically detects and reconciles semantic differences at different times. We articulate that this extended COIN framework is suitable for reasoning on the Semantic Web.

1 Introduction

The Web, and many other data sources, accumulate a large amount of data over time. In certain cases, it is even required by law for organizations to store historical data and make sure it is accurate and easy to retrieve[1]. It is critical that the retrieved data can be correctly interpreted, especially in the context of the Semantic Web where users or agents make decisions based on information coming from multiple autonomous sources. This well known semantic heterogeneity problem is further complicated when the semantics of data not only differs across sources, but also changes over time.

As an example, suppose an arbitrage specialist in New York City wants to study longitudinal stock price differences among exchanges. Within several keystrokes and mouse clicks at Yahoo Finance site, he retrieves the historical prices for Daimler-Chrysler at New York and Frankfurt exchanges, see Figure 1. He is astonished by what he sees: 1) the prices at the exchanges are extraordinarily different (*what an*

[1] See Robert Sheier on "Regulated storage" in ComputerWorld, 37(46), November 17, 2003. Health Insurance Portability Act requires healthcare providers keep records till two years after death of patients; Sarbanes-Oxley Act requires auditing firms retain records of financial statements.

arbitraging opportunity); and 2) the price at Frankfurt plunged by almost a half at the turn from 1998 to 1999 (*too bad if someone bought the stock right before the price decline*)! Possible conclusions from the observations are in parentheses.

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

These are wrong conclusions based on false observations, all resulted from unresolved heterogeneous and changing semantics. 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. Once the data is transformed into the analyst context, i.e., all prices in US dollars, it can be seen that there is neither significant arbitraging opportunity nor abrupt price plunge for this stock. Unfortunately, the technologies used by the analyst do not sufficiently represent and make use of the semantics of the data being exchanged.

The example illustrates the kinds of problems that the Semantic Web aims to solve. Context Interchange (COIN) framework [7, 10], originated from the semantic data integration research tradition, shares the common goal and provides an extensible solution to semantic heterogeneity problems. COIN is a web-based mediation approach with several distinguishing characteristics:

- Detection and reconciliation of semantic differences are a system service and are transparent to users;
- Mediation does not require that any semantic differences between each source-receiver pair to be specified a priori, rather, it only needs a declarative description of data semantics and the methods of reconciling possible differences. Semantic differences are detected and reconciled at the time of query; and
- 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.

As elaborated later, these features make COIN suitable for the Semantic Web. In this paper, we describe the COIN framework with a focus on the representation and reasoning of changing semantics. The use of SQL is for convenience purpose and should not be construed as a constraint of the framework, which will become clear as we describe the logic formalism of COIN.

2 Context Interchange by Example

Before a formal definition is given, we call implicit metadata knowledge such as the currency for price *context* and the history of time varying metadata *temporal context*. The temporal context in the previous example is quite simple. To illustrate the COIN approach, let's consider a slightly more complicated example in Figure 2, where various aspects of data semantics in a company financials data source change at different times.

Context *c_src*

1. All *monetary values* are in French Francs until 2000 and in Euros afterwards;
2. All *monetary values* have a scale factor of 1M until 1999, 1K until 2001, and 1M from 2002
3. *Profit* is tax excluded until 2000, tax included afterwards
4. All *other numbers* have a scale factor of 1 until 2001 and 1K afterwards

Financials

Year	Num_Employee	Profit	Tax
…			
1999	5100	4.2	1.1
2000	12000	13000	2500
2001	25.3	20000	4800
2002	30.6	35.3	7.97
…			

Context *c_target*

1. All *monetary values* are always in USD;
2. All *monetary values* always have a scale factor of 1K
3. *Profit* is always tax included
4. All *other numbers* always have a scale factor of 1K

Query Q1:

```
Select Year,Num_Employee,Profit
From Financials
Where 2000=<Year;
```

Fig. 2. Temporal context example. Various aspects of data semantics change at different times

This example involves one data source with asynchronously changing semantics shown on the left side, and one data receiver, whose context is stated on the right side in Figure 2. The situation becomes much more difficult for humans to handle when there are multiple sources being used to serve multiple receivers – each with a potentially different context. Although our design is intended for the more complex case, we will use the simple situation of Figure 2 for presentation purposes. We distinguish two kinds of temporal contexts: 1) *representational* – different representations for the same concept, e.g., different currencies and scale factors for monetary values; and 2) *ontological* – similar concept with slight differentiation, e.g., profit with taxes included or excluded. In resolving ontological differences, representational differences of related concepts should be resolved as well.

In this example, the user knows what can be queried from the source but is unaware of the context differences. This is similar to how the Web is typically used. Clearly, a direct execution of the sample query Q1 in Figure 2 over the source will

return data that is not correct in the user context. With COIN, however, the user can issue the query and expect that the results can be correctly interpreted in his context. To relieve the users of the burden of keeping track of and reconciling context differences, data semantics of sources and receivers need to be explicitly recorded as a set of context axioms and elevation axioms with reference to an ontology. As shown in Figure 3, the Context Mediator detects context differences at run time and rewrites user queries into mediated queries (*MQ*) that reconcile these differences. The MQ can be returned to the user as an *intensional* answer to the original query, or it can be sent to the query optimizer to generate an optimized query plan for the executioner to retrieve data, perform necessary conversions, and assemble the data records as an *extensional* answer. So when query Q1 is issued, the mediator can generate the following MQ1:

```
MQ1: Select Year, Num_Employee*0.001, Profit*O.Rate+Tax*O.Rate
     From Financials, (Select Rate from Olsen, Financials
            where Expressed='FRF' and Exchanged='USD' and Date=Year) O
     Where Year=2000
     Union
     Select Year, Num_Employee*0.001, Profit*O.Rate
     From Financials, (Select Rate from Olsen, Financials
            where Expressed='EUR' and Exchanged='USD' and Date=Year) O
     Where Year=2001
     Union
     Select Year, Num_Employee, Profit*O.Rate*1000
     From Financials, (Select Rate from Olsen, Financials
            where Expressed='EUR' and Exchanged='USD' and Date=Year) O
     Where 2002=<Year;
```

MQ1 considers all the semantic changes since year 2000 and reconciles the semantic differences between the source and the receiver through three sub-queries. Note an auxiliary data source *Olsen* for currency conversion is introduced in MQ1. The execution of an optimized MQ1 will return the dataset whose values have been transformed to conform to the user context.

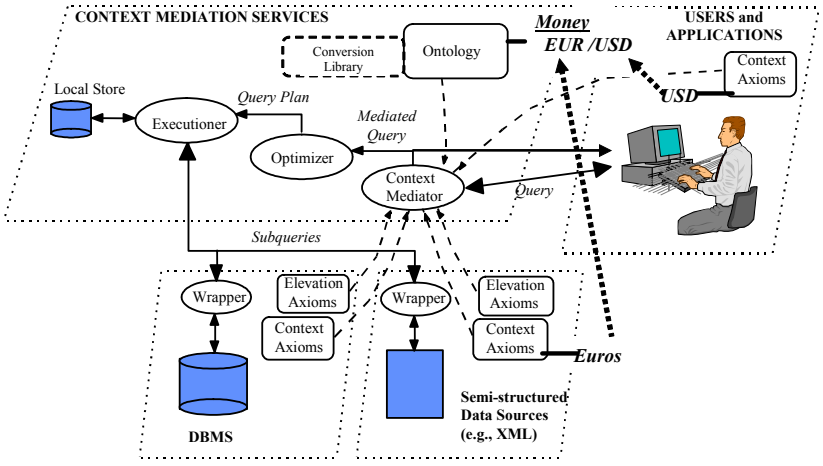

Fig. 3. Architecture of the COIN system

3 Temporal Context in COIN Framework

The example in Figure 2 can be understood with the notion of *context* as in [17]. Each tuple in the *Financials* table represents a statement about the company, which is true only within the given context *c_src*. Each statement is not true when it is directly restated in the *c_target* context. The COIN framework provides a logic formalism for describing context and a mediation service for restating statements from source contexts in the receiver context. For example, it is true that the number of employees in 1999 is 5100 in the *c_src* context, which can be expressed as

$$\overline{c}: \quad ist(c\_src, num\_employee(1999, 5100).$$

a correct restatement of which in the *c_target* should be

$$\overline{c}: \quad ist(c\_target, num\_employee(1999, 5.1).$$

because the scale factors in the two contexts are different.

The COIN system automates the process that restates the facts from the source context in the target context. This is achieved by formally capturing context knowledge and automatically detecting and reconciling context differences.

The original COIN framework was based on a snapshot data model that lacks the capability of describing and reasoning about changes of semantics. We will describe the ongoing research that non-trivially extends COIN to represent and process temporal context, the understanding of which can be benefited by a brief introduction to the existing COIN framework; further details can be found in [7, 10].

3.1 Overview of the COIN Framework

Knowledge representation in COIN is based on an object oriented deductive data model that consists of three components:

- Ontology – to define the semantic domain using a collection of semantic types and their relationships. A type 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 *profit*); 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;
- Elevation axioms – to establish correspondences between data elements in sources and the types in the ontology, e.g., tax in the example in Figure 2 corresponds to *monetary value* in the ontology; and
- Context axioms – 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. Thus, conceptually a context can be thought to be a set of <*modifier*, *object*> pairs, where *object* is a singleton in most non-temporal cases.

These components can be naturally represented using F-Logic [15], which has rich constructs for describing types and their relationships and has formal semantics for inheritance and overriding. 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.

The detection and reconciliation of context differences are through a mediation engine implemented in abductive constraint logic programming, details of which are provided later.

3.2 Representation of Temporal Context

For temporal context representation, the ontology is augmented with 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 time point and time interval. Any other concepts or types whose value or semantics change over time are related to temporal concepts via named attributes or modifiers in the COIN ontology, e.g., *monetary value* has a temporal attribute of type *temporal entity*, which can be described in F-Logic as:

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

A graphical representation of the ontology for the example is given in Figure 4.

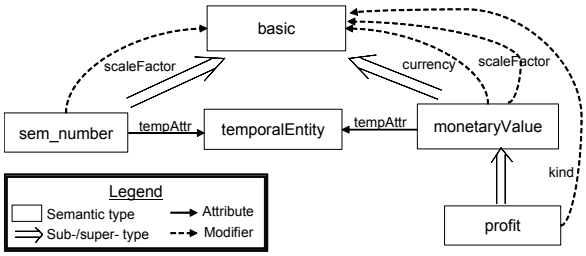

Fig. 4. A graphical representation of the example ontology

The changing semantics of data is captured by specifying the history of the modifiers. Thus modifiers are multi-valued over the entire history; but at any point in time, there is only a single valid value for each modifier. For example, to describe the tem-

poral context regarding the currency in the example, we first declare in the ontology that *currency* is a multi-valued *modifier* for *monetary value*:

$$monetaryValue[currency(ctxt) \Rightarrow\Rightarrow basic].$$

Next, we specify the values and their corresponding time intervals using the following context axiom (which needs to be rewritten in clausal form):

$$\begin{aligned} &\forall X : monetaryValue\ \exists Y : basic \vdash \\ &\quad X[currency(c\_src) \rightarrow Y] \wedge \\ &\quad (Y[value(c\_src) \rightarrow 'FRF'] \leftarrow X[tempAttr \rightarrow T] \wedge T \overset{c\_src}{\in_t} I_{\leq 2000}) \wedge \\ &\quad (Y[value(c\_src) \rightarrow 'EUR'] \leftarrow X[tempAttr \rightarrow T] \wedge T \overset{c\_src}{\in_t} I_{2001\leq}). \end{aligned} \quad \textbf{(1)}$$

where $\vdash$ is used for pre-declaration for object types, $I_{\leq 2000}$ represents the time interval up to year 2000 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:

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

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.

Another way of thinking about history specification is to regard it as a set of rules with applicability of each being constrained by an appropriate temporal relation. With this view, temporal context representation is analogous to *data level context* in snapshot based COIN, by *data level context* we mean that within a named context a modifier can have different values depending on other characteristics of the object. [11] gives an example of this kind where the scale factor for *monetary value* is 1 if it is not in Japanese Yen and is 1000 if otherwise:

$$\begin{aligned} &\forall X : monetaryValue\ \exists Y : basic \vdash \\ &\quad X[scaleFactor(c) \rightarrow Y] \wedge \\ &\quad (Y[value(c) \rightarrow 1] \leftarrow X[currency(c) \rightarrow Z] \wedge Z \overset{c}{\neq} 'JPY') \wedge \\ &\quad (Y[value(c) \rightarrow 1000] \leftarrow X[currency(c) \rightarrow Z] \wedge Z \overset{c}{=} 'JPY'). \end{aligned}$$

This data level context specification is similar to temporal context especially after the temporal relation is transformed into a set of comparisons between time points. Thus temporal context is analogous to data level context even though they are conceptually quite different. This analogy suggests that the mediation engine for snapshot COIN can be extended with additional rules for temporal relations to process temporal context.

Because modifiers are single valued at any given point in time, the conversion functions for temporal context handling are the same as those in snapshot COIN. Earlier we described conversions as part of context specification. Through parameterization, a conversion function can be defined more generally to be used in multiple contexts. For example, the following currency conversion function can be used to

convert monetary values from any arbitrary context *c1* to any other arbitrary context *c2*:

$$
\begin{aligned}
&x : monetaryValue \vdash \\
&\quad x[cvt(currency, c2)@c1, u \rightarrow v] \leftarrow \\
&\qquad x[currency(c1) \rightarrow C_f] \wedge x[currency(c2) \rightarrow C_t] \wedge x[tempAttr \rightarrow T] \wedge \\
&\qquad olsen\_(A, B, R, D) \wedge C_f \overset{c2}{=} A \wedge C_t \overset{c2}{=} B \wedge T \overset{c2}{=} D \wedge \\
&\qquad R[value(c2) \rightarrow r] \wedge v = u * r.
\end{aligned}
\tag{2}
$$

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.

3.3 Mediation with Abductive Constraint Logic Programming

The task of mediation is to translate a user query that assumes everything is in user context to a mediated query (*MQ*) that reconciles context differences detected during the mediation process; when the MQ is subsequently executed, facts stated in source contexts are correctly restated in the user context. This task corresponds to the abductive logic programming (ALP) framework [13] very well. As we see earlier, temporal relations in temporal context representation can be considered as temporal constraints over the time domain, which suggests constraint solving is necessary. It follows that mediation can be implemented with the abductive constraint logic programming (ACLP) [14] where abduction, consistency checking, and constraint propagation are interleaved.

As an extension to ALP, 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 ground and constrained non-ground abducible predicates.

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 the sources and other callable external functions constitute $\mathcal{A}$. We also allow constraints to be abducibles. $\mathcal{IC}$ consists of integrity constraints in data sources and any constraints introduced in the user query.

Abductive inference in COIN is a modified SLD-resolution [6] in that literals corresponding to predicates in data sources are abducted without evaluation; constraints over non-temporal types are also directly abducted. Constraints introduced in user

query, by the conversion functions, and related to *temporal entity* types are evaluated after they are abducted. The prototype is implemented using constraint logic programming environment ECLiPSe (http://www.icparc.ic.ac.uk/eclipse/) with the extension of Constraint Handling Rules (CHR) [9]. Naturally, we use the constraint store to collect the abucibles; when a constraint is abducted, certain CHR rules will be triggered to simplify/propagate the constraint or signify a failure to cause backtracking of abduction. At the end of a successful resolution, predicates in the constraint store constitute the abductive answer.

The current implementation is an enhancement to the procedure described in [4] with significant extensions to handle conversion composition [8] and temporal context. We will focus on temporal context handling, thus we only briefly describe the procedure here and refer readers to [4] for other details.

The mediator accepts queries in clausal form. Auxiliary components in our prototype translate a SQL query into its equivalent logic form for mediation and translate the MQ back into SQL for query planning and execution. The example query can be translated to the following clausal form:

$$\leftarrow answer(y, n, p).$$
$$answer(y, n, p) \leftarrow financials(y, n, p, t) \wedge 2000 \le y.$$

where we use predicate *answer* to simulate *projection*. This query is naïve in the sense that it directly references the primitive objects in data sources without concern about potential context differences. The query is further translated into a well formed context-aware query by 1) replacing each primitive object with its Skolemized semantic counterpart as specified in elevation axioms. We call a so transformed relation a *semantic relation*; 2) replacing comparison operators for non-temporal objects with $\overset{c\_target}{\Diamond}$; 3) replacing comparison for temporal objects with temporal relations (e.g., replacing $\le$ with *tle_*, which stands for inclusive *before*); and 4) invoke the parameterized *value* function for each projected object; *value* function calls are also invoked in places where $\overset{c\_target}{\Diamond}$ is used. After this transformation, the naïve query becomes:

$$\begin{aligned} answer(y, n, p) \leftarrow\ & financials\_(sk_y, sk_n, sk_p, sk_t) \wedge tle\_(sk_{2000}, sk_y) \wedge \\ & sk_y[value(c\_target) \rightarrow y] \wedge sk_n[value(c\_target) \rightarrow n] \wedge \\ & sk_p[value(c\_target) \rightarrow p]. \end{aligned}$$

This is the query that is fed to the mediator. Literals corresponding to semantic relations are unified with elevation axioms and the corresponding source predicates are abducted. Constraints are abducted and immediately processed if corresponding CHR rules exist; abduction backtracks after a failure in constraint process. Value functions are processed with the following algorithm:

```
For each value function
  Find direct and inherited modifiers of the object
  For each modifier
    Compare modifier values in source and target contexts
    If they differ
      Find conversion function for the object
      If not found, find conversion function of parents
      If not found, invoke function composition
```

```
        Invoke found/composed conversion and return a primitive
    If no modifier, return the primitive value of the object ■
```

This algorithm implements the semantics and simulates non-monotonic inheritance afforded in the knowledge representation language. It is recursive in that modifiers are objects, comparison of which introduces *value* function calls; declaratively defined conversions usually introduce *value* function calls as well.

Given the well formed query, the resolution of the first literal generates the abducible *financials*(y_0, n_0, p_0, t_0). Similarly $tle(2000, y_0)$ is abducted into the constraint store when $tle\_(sk_{2000}, sk_y)$ is resolved. The object corresponding to Year attribute has no modifier, thus this sub-goal succeeds without generating any abducible. The other two *value* function calls will non-trivially involve the above algorithm. For elucidation purpose, let's focus on the resolution for value function call on *profit* object. Modifier *kind* for describing whether taxes are included is directly found; other modifiers including *currency* and *scale factor* are found through inheritance from *monetary value* type. Let's focus on the conversion for currency differences. In the conversion function for currency, a *value* function call (3rd line in formula 2) is made to find the currency in the source, which will find the first rule for history specification that essentially says before year 2000 the currency is 'FRF' (3rd line in formula 1). Further resolution will post $tle(y_0, 2000)$ into the constraint sotre. This constraint, along with the constraint $tle(2000, y_0)$ that was posted earlier, will trigger the following CHR rule:

```
antisymmetry @ X tle Y, Y tle X <=> X=Y.
```

which is a simplification rule that replace the two *tle* temporal constraints with a single equality constraint. There are other CHR rules for handling cases such as transitivity, overlapping, and inconsistency. When this resolution finishes successfully, the constraint store contains all the abudcibles that can be translated into the first subquery in MQ1. With backtracking, all the other answers are found.

Although finding the values of all modifiers with non-conflicting temporal constraints is a combinatorial search, many of the search branches are very shallow, e.g., given the constraint introduced in user query, the search branch that tries scale factor before 1999 fails immediately after this temporal constraint is posted into the store.

4 Discussion

With the extension of temporal context handling, the COIN framework is now capable of solving a much wider range of semantic heterogeneity problems. Any changes in representational and ontological semantics can be represented and reasoned about within the extended COIN framework. For example, [5] describes a spatial-temporal scenario where historic statistics of the economy (e.g., GDP) and environment (e.g., CO_2 emissions) of each sovereign country is stored in several sources. Let us consider a user who performs longitudinal studies in the Balkans and is used to using 'YUG' to refer to the geographic area bounded by the former Yugoslavia. Because the area has gone through a series of balkanizations, notably in 1991 the region broke up into five sovereign states, the user's query that worked before the broken-up would stop

working correctly afterwards. With COIN, however, the query continues to work correctly once the temporal context is appropriately encoded. We have successfully tested this scenario with our prototype. The MQ, which will not be shown here, is a couple pages long because it needs to combine data from appropriate individual countries, convert currency differences for each, and reconcile other context differences (e.g., scale factors, etc.).

This Balkans example is interesting because it demonstrates that the extended COIN framework can process different aggregation rules that are applicable at different time periods. There are many accounting rules of this nature. Just like in our solution approach we draw analogy between temporal context and non-temporal data level context, there are also analogous non-temporal aggregation rules. For example, instead of depending on time periods, it depends on purposes (SEC filing, risk assessment, taxation, etc) that the rules differ for whether the total revenue of a corporation should include those of foreign branches, subsidiaries, subsidiaries of branches and subsidiaries, and other companies majority-owned by the corporation. As is demonstrated in the Corporate Householding research [16], COIN framework can be applied to this scenario as well to represent and reasoning about those complex rules.

The analogy between temporal context and data level context is important. It allows us to use computationally more mature technologies such as constraint solving to reconcile temporal context differences. This was not obvious when we first approached the problem of changing semantics because the nature of the problem naturally suggests the use of temporal logics.

The COIN framework is applicable to the Semantic Web for several reasons. Although we used relational sources in the example, COIN is not restricted to the relational data model because the framework is in fact based on an object oriented logic model, which is expressive enough to encompass non-relational data sources. The ACLP based implementation of the mediator generates the MQ from a set of abducibles, which include predicates admissible by most query languages, be it relational, keyword based, or ontology based. More importantly, the COIN framework relies on the description of data semantics, not on the description of the semantic differences; the latter are automatically detected and reconciled when a query is issued. This 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. And lastly, 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.

In this paper we assumed that time is uniformly represented in all sources and receivers. In reality, temporal entities can be heterogeneous across systems. For future research, we would like to extend the notion of temporal context to deal with semantic heterogeneities among temporal entities. This may be achieved by introducing the full Time ontology into knowledge representation. The conversions for reconciling these sorts of heterogeneity can be implemented as external function calls to web services that specifically handle time zones, calendars, and granularities [2, 3].

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. C. Bettini, "Web services for time granularity reasoning," TIME-ICTL'03, 2003.
3. C. Bettini, S. Jajodia, and X. S. Wang, Time Granularities in Databases, Data Mining, and Temporal Reasoning: Springer, 2000.
4. S. Bressan, C.H. Goh, T. Lee, S. Madnick, M. Siegel, "A Procedure for Mediation of Queries to Sources in Disparate Context", ILPS'97, 1997.
5. N. Choucri, S. Madnick, A., Moulton, M. Siegel, H. Zhu, " Information Integration for Counter Terrorism Activities: The Requirement for Context Mediation1", IEEE Aerospace Conference, 2004.
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 and S. Madnick and 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. A.C. Kakas, R.A. Kowalski, F. Toni, "Abductive Logic Programming", Journal of Logic Programming, 2(6), pp. 719-770, 1993.
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. 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.
17. J. McCarthy, "Generality in Artificial Intelligence", CACM, 30(12), pp. 1030-1035, 1987.