# Context Interchange as a Scalable Solution to Interoperating Amongst Heterogeneous Dynamic Services

Hongwei Zhu, Stuart E. Madnick
MIT School of Engineering, MIT School of Engineering and MIT Sloan School of Management

***Abstract*—Many online services access a large number of autonomous data sources and at the same time need to meet different user requirements. It is essential for these services to achieve semantic interoperability among these information exchange entities. In the presence of an increasing number of proprietary business processes, heterogeneous data standards, and diverse user requirements, it is critical that the services are implemented using adaptable, extensible, and scalable technology. The Context Interchange (COIN) approach, inspired by similar goals of the Semantic Web, provides a robust solution. In this paper, we describe how COIN can be used to implement dynamic online services where semantic differences are reconciled on the fly. We show that COIN is flexible and scalable by comparing it with several conventional approaches. With a given ontology, the number of conversions in COIN is quadratic to the semantic aspect that has the largest number of distinctions. These semantic aspects are modeled as modifiers in a conceptual ontology; in most cases the number of conversions is linear with the number of modifiers, which is significantly smaller than traditional hard-wiring middleware approach where the number of conversion programs is quadratic to the number of sources and data receivers. In the example scenario in the paper, the COIN approach needs only 5 conversions to be defined while traditional approaches require 20,000 to 100 million. COIN achieves this scalability by automatically composing all the comprehensive conversions from a small number of declaratively defined sub-conversions.**

***Index Terms*— data integration, heterogeneous sources, ontology, scalability**

## I. Introduction

With the connectivity of the Web, enterprises of any size can potentially interact with many other enterprises and/or consumers anywhere in the world. Information exchange plays a key role in these interactions. The benefit of this global reach can be realized only when the information is exchanged efficiently and meaningfully. However, each entity typically has its own proprietary business processes, data standards, and individual preferences, which makes meaningful information exchange extremely challenging. Unless you are one of the very few big players in the market, e.g., Wal-Mart in the retail sector, so you can dictate that all your trading partners conform to your standard, you have to deal with various standards, interface with multiple processes, and accommodate diverse consumer preferences, which can also change autonomously over time. The traditional middleware approach that "hard-wires" all connections between systems obviously is not feasible in this diverse and dynamic environment.

To make the point clear, let us consider a concrete example where a comparison shopping service is provided globally. Vendors in different countries quote their prices in local currencies. In addition, what are included in the quoted prices often vary by local conventions and vendor choices. For example, in most European countries the quoted prices include the Value-Added Tax (VAT), whereas in the U.S. the quoted prices usually do not include such taxes. Furthermore, the comparison may only make sense to certain consumers when the price reflects the total cost that includes shipping and handling charges as well as taxes. Although most prices are given in unit of local currencies, some may use certain scale factors to trim out trailing zeros simply because it would be too cumbersome otherwise. For example, 1 US dollar is about 1.5 million Turkish liras; imagine how cumbersome it will be if all prices are in unit of Turkish liras – it is much more practical to quote Turkish prices in thousands or millions of Turkish liras. Thus a direct comparison between literal values among diverse sources may not make any sense. Such an online comparison service has to consider the context differences and dynamically transform data from thousands of online stores around the world into the consumer's context, e.g., convert prices in other currencies, different scale factors, and various definitions for price in terms that are appropriate in the consumer context.

Any viable solution to this problem has to be adaptable to accommodate changes gracefully, extensible to allow

H. Z. Zhu is with MIT Engineering Systems Division, School of Engineering, Room E53-336, Cambridge, MA 02142 USA. (e-mail: mrzhu@mit.edu).

S. E. Madnick is with MIT Engineering Systems Division, School of Engineering and MIT Information Technologies Group, Sloan School of Management, Room E53-321, Cambridge, MA 02142 USA (phone: 617-253-6671; fax: 617-253-3321; e-mail: smadnick@mit.edu).

easy addition and removal of data sources, and scalable to handle a large number of data sources efficiently. Although the Semantic Web [1] has the vision ultimately to address these issues, its realization will take some time. There are near term solutions, however. The Context Interchange (COIN) approach [2, 6, 7], originated from semantic data integration research over a decade ago, provides formalisms of context knowledge representation and a reasoning process that automatically reconciles semantic differences like the ones in the online comparison service example. After describing a motivating example, we discuss various implementation approaches, including COIN. Then we give an analysis of the adaptability, extensibility, and scalability of these approaches, which indicates that COIN is a very robust solution to interoperation of heterogeneous services.

## II. Motivating Example

Besides the comparison service example mentioned above, there are many other online services, such as comprehensive travel services involving multiple airlines, hotels, and car rental services, that have similar needs for interoperating amongst heterogeneous services. In addition, these same requirements can be often important to support the on-going updating of multiple data marts in a large organization or among multiple organizations as well as Business Intelligence (BI) and Enterprise Information Integration (EII) activities. Also, there is an increasing trend for logistics services and financial services to do Straight-Through-Processing (STP) whereby the flow of data through multiple systems, including data conversions and adjustments, is performed without human intervention. For consistency, we will continue to use the comparison service as our motivating example.

The purpose of an online comparison service is to allow the prices from all contexts to be compared in any context chosen by the consumer (i.e., data receiver). Here the term *context* refers to a set of implicit assumptions of various semantic aspects, with which the meaning of data is interpreted. Consider the example where a US consumer is interested in a product available online from South Korea and Turkey. South Korean vendors quote their prices in thousands of South Korea won (KRW, $1 is about 1000 KRW), not including taxes, while Turkish vendors quote prices with taxes included in millions of Turkish lira (TRL), but the consumer wants the comparison to be in unit of US Dollar (USD) with taxes and Shipping and Handling included. Thus the comparison service needs to perform certain conversions to meet the consumer's requirement. A consumer can be from any context and usually requires comparisons to be in his context, e.g., a Turkish consumer may want to compare prices in millions of lira with taxes included, and similarly a South Korean consumer will want comparison in his own context.

When the number of sources and contexts is small, these conversions can be implemented as conversion programs in the middleware. But, a practical service needs to deal with a large number of sources with changing standards and requirement. Let us assume that the comparison service covers 100 countries, each having its unique currency and each consisting of 100 vendors. Thus, there are a total of 10,000 sources in this example. For simplicity, let's assume the consumer chooses his context to be the same as one of the sources. Although all vendors in the same country may use the same currency for price, they may use different price definitions and scale factors. Table 1 summarizes the potential context differences in terms of just these four semantic aspects[1]: currency, scale factor, price definition, and date format (for the purpose of finding exchange rate at a given day).

TABLE 1. SEMANTIC VARIETIES IN 10,000 WORLDWIDE SOURCES

| **Semantic Aspect** | **Number of Distinctions** |
|---|---|
| Currency | 100 different currencies |
| Scale factor | 4 different scale factors, e.g., 1, 100, 1000, 1000000 |
| Price definition | 3 different definitions, e.g., base price, base+tax, and base+tax+SH |
| Date format | 3 different formats, e.g., yyyy-mm-dd, mm/dd/yyyy, and dd-mm-yyyy |

Thus, there could be 3600 (i.e., 100*4*3*3) different contexts amongst these sources; e.g., one source has US dollars for currency, scale factor being 1, price as tax and shipping and handling included, with mm/dd/yyyy date format; another source has Turkish liras for currency, scale factor being 1000000, price as only tax included, with dd-mm-yyyy date format, etc. The online comparison service needs to implement the conversions so that the comparison can be performed for sources in any context.

## III. Approaches to Interoperating Heterogeneous Services

We can view the 10,000 online vendors as 10,000 data sources, each providing price quoting services that use different quoting standards; we will use data sources and services interchangeably in the rest of the discussion. The comparison service needs to interoperate among them and allow comparisons in all quoting standards. This can be implemented in middleware, to which there exist a number of possible approaches.

### *A. Hard-wiring approaches*

There are several alternatives to "wire" up these disparate services using conversion programs. They are often termed "hardwired" because the conversion programs

[1] In even more realistic situations, there can be many more context differences involved, such as lot size (whether price is for single unit or

need to be rewritten or modified if there are changes in the underlying services. Three representative approaches are depicted in Figure 1 below.

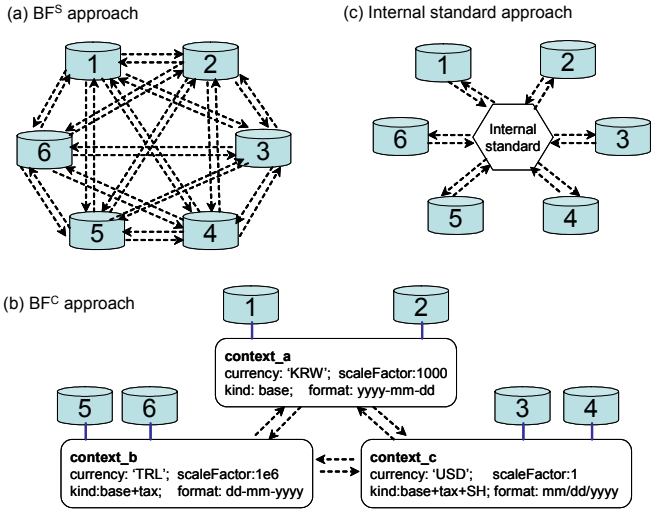



Figure 1. Conventional hard-wiring approaches

*Source-based brute-force hard-wiring ($BF^S$)*. For any pair of the N individual data sources, implement a conversion program to convert from one source to another, and another conversion program for the reverse direction. In Figure 1(a) we show 6 data sources; the conversion programs between them are depicted by dotted arrows.[2]

*Brute-force hard-wiring with shared conversions ($BF^C$)*. Implement a pair of conversion programs only between pairs of contexts that are different in at least one semantic aspect. If source $s_i$ corresponds to context $c_x$ and source $s_j$ corresponds to context $c_y$, to exchange information between $s_i$ and $s_j$, use conversion program $C_{xy}$ (or $C_{yx}$, depending on the direction of conversion.) It allows multiple sources in the same context to share the same conversion programs, thus it has the potential of reducing the number of conversion programs. In Figure 1(b) we assume that the 6 data sources only have 3 unique contexts (i.e., each context is shared by two sources.) This latter approach requires the establishment and maintenance of records for the correspondences between sources and contexts, which can be a laborious task if there are many source contexts.

Clearly, both approaches need to enumerate all combinations, with $BF^s$ on the combination of all sources and $BF^c$ on the combination of all contexts.

*Internal standard*. There are other hard-wiring approaches that require the adoption of an internal standard. Then the comparison service can translate from other standards to this internal standard and vice versa, as shown in Figure 1(c). When the data in the underlying sources do not change frequently, the Extraction-Transform-Load (*ETL*) approach can be used; ETL has been widely used in many data warehouses and data mart activities. If the data changes frequently, the comparison service can call the conversion programs on the fly to fetch fresh data upon request. This is often known as the Global Schema (*GS*) approach because the internal standard can be viewed as the global schema over all data sources.

A meaningful comparison takes place first by converting other contexts to the internal standard, then by converting from the internal standard to the target context. Although this approach can reduce the number of conversion programs needed, it is often impractical to establish and maintain a standard[3]; in addition, it also has shortcomings relative to the more customized conversion programs used in the "brute force" approaches described earlier. For example, if the internal standard for currency was US dollars and the source context currency was Korean won and the target context was Turkish lira, this approach converts the price from Korean won to US dollars and then from US dollars to Turkish lira. The "brute force" approaches would convert directly from Korean won to Turkish lira. This situation becomes even more burdensome if the source and target contexts were the same (e.g., Korean won). In the "brute force" customized conversion program approach, data would be passed without any transformation whereas the internal standard approach would still convert to and from US dollars, which involves extra processing and often loses accuracy in data.

All these approaches require the conversions to be pre-programmed, thus lack adaptability to accommodate changes. As we will see in a later section, they also lack scalability to handle a large number of heterogeneous services and extensibility to the addition of new services.

### B. The Context Interchange (COIN) Approach

#### 1) The COIN Framework

The COIN framework consists of a deductive object-oriented data model for knowledge representation, a general purpose mediation service module that detects semantic differences between sources and receivers and generates mediated query to reconcile them, and a query processor that optimizes and executes the mediated query to retrieve and transform data into user context (see Figure 2).

Knowledge representation consists of three components. An *ontology* is used to capture common concepts and their relationships such as one concept being a property (i.e., attribute) or a sub-concept (i.e., *is_a* relationship) of another. Each concept may have modifiers as a special kind

price is for packets of 6 or 12) and other context sensitive attributes to the products, such as weight and size dimension.

[2] In some cases, two sources (e.g., 5 and 6) might have the same context, so no conversion is needed. But it is still necessary to manually determine this and be on the alert for changes in the future, so the manual work effort is not null even in such cases.

[3] Reaching such enterprise-wide agreement, especially within large diverse organizations, can be extremely difficult and time-consuming and is often unsuccessful. Further discussion of these issues is beyond the scope of this paper.

of property to explicitly represent the meta-attributes of the concept that can vary by source and receiver. We call the declarative specifications of modifier values *context*. Conversions between contexts are also declaratively defined in the *context definitions*. The semantic *mappings* establish the correspondence between data elements in the sources and the concepts in the ontology. These components are expressed in the object-oriented deductive language F-Logic [10], which can be translated into Horn logic expressions that we use internally, or Web Ontology Language (OWL) and RuleML intended for the Semantic Web.

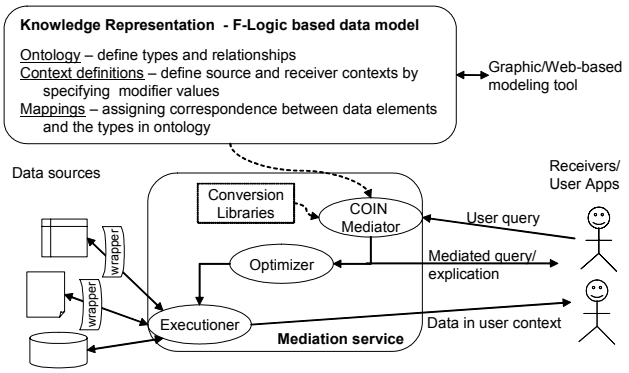



Figure 2. Architecture of COIN System

The core component in the mediation service module is the COIN mediator implemented in abductive constraint logic programming [9], where constraints are concurrently solved using Constraint Handling Rules (CHR) [5]. It takes a user query and produces a set of mediated queries (MQs) that resolve semantic differences. This happens by first translating the user query into a Datalog query and using the encoded knowledge to derive the MQs that incorporate necessary conversions from source contexts to receiver context. The query processor optimizes the MQs using a simple cost model and the information on source capabilities, obtains the data, performs the conversions, and returns final datasets to the user.

Within the COIN framework, the users are not burdened by the diverse and changing semantics in data sources, all of which are captured in the knowledge representation component and are automatically taken into account by the mediator. Adding or removing a data source is accomplished by adding and removing declarations in the knowledge base, which does not affect the mediator and the query processor at all. There are other ontology based approaches with varying capabilities [11], detailed discussion of which is out of the scope of this paper.

*2) Online Comparison Service Using COIN*

The COIN approach achieves flexibility and scalability by declaratively defining the conversions for each individual modifier and automatically composing the overall conversion program at run time. We will use the comparison shopping example to demonstrate the COIN approach.

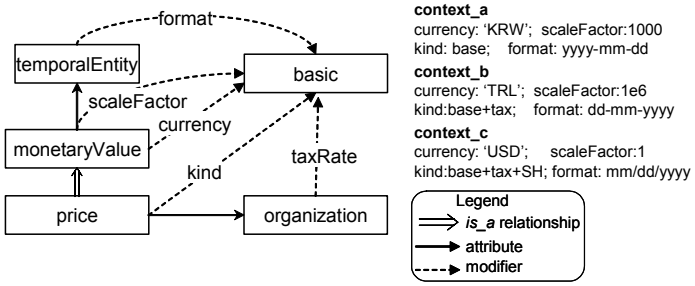



Figure 3. Excerpt of Ontology and Context for Dynamic Comparison Service

Figure 3 shows an excerpt of the ontology on the left, and an intuitive context description of three example contexts on the right. Here *price* is a sub-concept of *monetaryValue*, which has a *temporalEntity* attribute to allow for time dependent currency conversions. By inheritance in the data model, *price* has all attributes and modifiers that its parent *monetaryValue* has.

Although we show the syntax in F-Logic that is used internal to COIN in the following discussion, a typical data administrator[4] would use the user-friendly graphical front-end (as depicted in top right of Figure 2). We call the collection of these formulas as the knowledge base. For example, the following formula in the knowledge base states that in *context_a*, the currency for *monetaryValue* is KRW:

$$\forall X : monetaryValue \; \exists Y : basic \vdash X[currency(context\_a) \rightarrow Y] \wedge Y[value(context\_a) \rightarrow 'KRW'].$$

In COIN, conversions are not defined between sources, but only on individual modifiers between different modifier values. In the example, conversions are defined for converting between different currencies, scale factors, date formats, and notions of price. In some cases, a general-purpose conversion can be used between any arbitrary modifier values, i.e., variables are used in lieu of constants in conversion specification. For example, the following function can convert from an arbitrary currency $C_f$ in context *C1* to another arbitrary currency $C_t$ in context *C2* by using an external service called *olsen*[5]:

$$x : monetaryValue \vdash x[cvt(currency, C2)@C1, u \rightarrow v] \leftarrow x[currency(C1) \rightarrow C_f] \wedge x[currency(C2) \rightarrow C_t] \wedge x[tempAttr \rightarrow T] \wedge olsen\_(A, B, R, D) \wedge C_f = A \wedge C_t^{C2} = B \wedge T^{C2} = D \wedge R[value(C2) \rightarrow r] \wedge v = u * r.$$

Specifications and conversions for other modifiers can be expressed similarly.

When a consumer with a known context performs a

[4] Note that actual users need never see any of this – they merely issue queries and get the results returned to them in their context. It is the data administrators that define the individual contexts.

[5] Olsen is actually a web service at www.oanda.com that provides the exchange rate between any two currencies for any date. Using the Cameleon web wrapping technology, we are able to treat olsen as if it were a relational database.

comparison, the mediator intercepts the query, compares the context differences between the involved sources and the consumer, and automatically composes a comprehensive conversion using the conversions defined for relevant modifiers. That is, the conversions are composed dynamically according to the consumer context and the data sources accessed[6].

The key features of COIN approach can be illustrated using the following demonstration. In addition to the three contexts shown in Figure 3, let us consider two other contexts:

| **context_e** | **context_f** |
|---|---|
| currency: 'TRL'; scaleFactor:1000 kind:base+tax; format: yyyy-mm-dd | currency: 'USD'; scaleFactor:1 kind:base+tax; format: mm/dd/yyyy |

where for *context_e* one can think of a source in Turkey that uses scale factor and date format that are different from that of *context_b*, and for *context_f* one can think of an American, just arriving in Turkey, is subject to local tax but is used to comparing in unit of USD and with a date format commonly used in the U.S.

To simplify explication, we show how queries from different receivers to one data source are mediated by COIN to reconcile semantic differences. The source is src_turkey(Product, Vendor, QuoteDate, Price), which subscribes to *context_b* and lists vendor prices by product and quote date.

When the receiver context is the same as the source context, COIN determines that there is no semantic difference and will not perform any conversion. Figure 4 is the screen shot of the demo for this case; the Stage section in the middles allows us to step through the process of how a receiver query in the SQL box is transformed and executed by COIN. The empty Conflict Detection table indicates that there is no semantic difference; after Execution stage, data meaningful to the receiver is returned, which is shown at the bottom of Figure 4.[7]

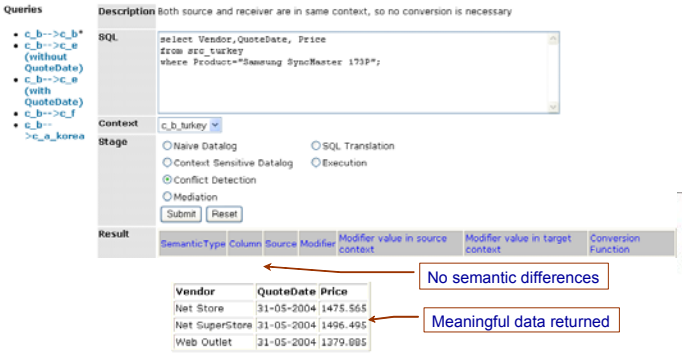



Figure 4. COIN Demo for the Case that Has No Semantic Difference

The conversions composed by COIN are made efficient by not including any sub-conversions that are not relevant to the user query. For example, *conext_e* is different from *context_b* in scale factor and date format. The conversion program implemented by hardwiring approaches would have code for reconciling both semantic differences. In contrast, when the query does not ask for QuoteDate, the conversion composed by COIN only contains the sub-conversion for scale factor; see Figure 5(a). Obviously, if the receiver also requests QuoteDate information, sub-conversions for both semantic aspects are included in the composed conversion, as shown in Figure 5(b). The mediated queries are shown in Datalog. Also note that the specification and the sub-conversion for scale factor of price is correctly inherited from its parent type *monetaryValue* in the ontology.

(a) Select Vendor, Price From src_turkey Where Product="Samsung SyncMaster 173P";

| SemanticType | Column | Source | Modifier | Modifier value in source context | Modifier value in target context | Conversion Function |
|---|---|---|---|---|---|---|
| price | Price | src_turkey(Samsung SyncMaster 173P, Name, QuoteDate, Price) | scaleFactor | c_b_turkey : 1000000 | c_e : 1000 | V5 is V4 / V3, V2 is V1 * V5 |

```
answer('V4', 'V3'):-
      src_turkey("Samsung SyncMaster 173P", 'V4', 'V2', 'V1'),
      'V3' is 'V1' * 1000.0.
```

Conversion for scale factor

(b) Select Vendor, QuoteDate, Price From src_turkey Where Product="Samsung SyncMaster 173P";

| SemanticType | Column | Source | Modifier | Modifier value in source context | Modifier value in target context | Conversion Function |
|---|---|---|---|---|---|---|
| price | Price | src_turkey(Samsung SyncMaster 173P, Name, QuoteDate, Price) | scaleFactor | c_b_turkey : 1000000 | c_e : 1000 | V5 is V4 / V3, V2 is V1 * V5 |
| temporalEntity | QuoteDate | src_turkey(Samsung SyncMaster 173P, Name, QuoteDate, Price) | format | c_b_turkey : European Style - | c_e : ISO Style - | datexform(V4, V3, V2, V1) |

```
answer('V5', 'V4', 'V3'):-
      src_turkey("Samsung SyncMaster 173P", 'V5', 'V2', 'V1'),
      datexform('V2', "European Style -", 'V4', "ISO Style -"),
      'V3' is 'V1' * 1000.0.
```

Conversion for scale factor

Conversion for date format

Figure 5. COIN only Introduces Relevant Conversions

COIN also dynamically detects and reconciles semantic differences between the auxiliary sources introduced by sub-conversions and the receiver. There are three semantic differences between *context_b* and *context_f* – currency, scale factor, and date format. When the receiver chooses not to retrieve QuoteDate, date format conversion is still included in the composed conversion (see Figure 6; the mediated query in Datalog is overlaid in the middle). This is because in the sub-conversion for currency we use the *olsen* data service, whose date format is different from the source; COIN detects this difference and automatically introduces appropriate sub-conversion *datexform*.

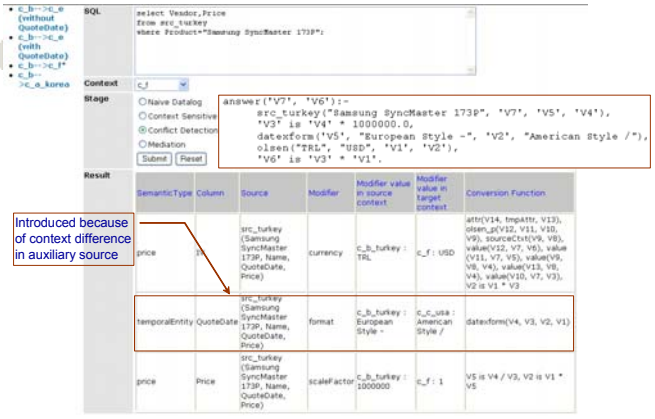
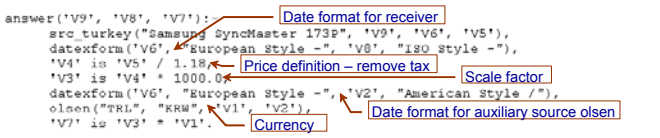



Figure 6. Reconciliation of Semantic Difference Introduced in Sub-Conversion

[6] Generated conversion programs could be saved for re-use, if desired.

[7] This is the same as the actual data values in the data source, since there were no conversions performed.

Finally, in Figure 7, we show the mediated query in both Datalog and SQL when a receiver in *context_a* queries for vendor prices and quote date from src_turkey. Note that 18% VAT included in the source is correctly removed because the receiver wants base price; two date format conversions are included, one for the receiver, the other for the auxiliary source *olsen*; currency and scale factor are also converted into the receiver context.




```
answer('V9', 'V8', 'V7'):-
      src_turkey("Samsung SyncMaster 173P", 'V9', 'V6', 'V5'),
      datexform('V6', "European Style -", 'V0', "ISO Style -"),
      'V4' is 'V5' / 1.18,
      'V3' is 'V4' * 1000.0,
      datexform('V6', "European Style -", 'V2', "American Style /"),
      olsen("TRL", "KRW", 'V1', 'V2'),
      'V7' is 'V3' * 'V1'.
```

```
select src_turkey.Vendor, datexform.date2, (((src_turkey.Price/1.18)*1000.0)*olsen.rate)
from   (select 'Samsung SyncMaster 173P', Vendor, QuoteDate, Price
        from   src_turkey
        where  Product='Samsung SyncMaster 173P') src_turkey,
       (select date1, 'European Style -', date2, 'ISO Style -'
        from   datexform
        where  format1='European Style -'
        and    format2='ISO Style -') datexform,
       (select date1, 'European Style -', date2, 'American Style /'
        from   datexform
        where  format1='European Style -'
        and    format2='American Style /') datexform2,
       (select 'TRL', 'KRW', rate, ratedate
        from   olsen
        where  exchanged='TRL'
        and    expressed='KRW') olsen
where  datexform2.date2 = olsen.ratedate
and    src_turkey.QuoteDate = datexform.date1
and    datexform.date1 = datexform2.date1
```

Figure 7. Mediated Query in Datalog and SQL for Receiver in *context_a*

## IV. Analysis of Adaptability, Extensibility, and Scalability

### A. Adaptability and Extensibility Analysis

*Adaptibility* refers to the capability of accommodating changes, such as semantic changes within a data source (e.g., when a French company charges its prices from French Francs to Euros). *Extensibility* refers to the capability of adding or removing data sources with minimal effort. We use the term *flexibility* to collectively refer to the two properties.

The $BF^S$ approach has the least flexibility. With $N$ sources, a change in any source would affect $2(N\text{-}1)$ conversion programs, i.e., $N$-1 conversion programs converting from the changing source to the other sources and vice versa. Adding or removing a source has similar effects.

This problem is somewhat reduced in the other hard-wiring approaches. With $BF^C$, if a source changes its context to coincide with another existing context, only the mapping needs to be updated; if it changes to new context, $2(n\text{-}1)$ conversion programs need to be updated,. Adding or removing a source also has similar effects. The ETL and GS internal standard approaches are better because of their hub-and-spoke architecture. But both require re-programming to handle changes, which can be tedious and error-prone. All hard-wiring approaches require the reconciliation of all semantic differences to be pre-determined and implemented in conversion programs. As a result, they lack flexibility.

In contrast, the ontology and context based COIN approach overcomes this problem. COIN has several distinctive features:

- It only requires that the individual contexts and individual conversions between a modifier's values (e.g., how to convert between currencies) be described declaratively in the knowledge base. Thus it is flexible to accommodate changes because updating the knowledge base is much simpler than rewriting conversion programs (e.g., it is merely necessary to indicate that a source now reports in Euros instead of French Francs).
- The customized conversion between any pair of sources (as many conversion programs as are needed) is composed automatically by the mediator using conversions of the relevant modifiers.
- COIN is able to compose all the conversion in $BF^S$, but without the burden of someone having to manually create and keep up-to-date all the pair-wise conversion programs.
- The COIN approach also avoids the multiple or unnecessary conversions that arise from the internal standard approaches since the conversion programs that it generates only includes the minimally required conversions, including no conversions for certain (or all) modifiers, if that is appropriate.

As we will see from the next section, the COIN approach significantly reduces the number of pre-defined conversions so that it can scale well when a large number of sources need to exchange information.

### B. Scalability Analysis

In order to accomplish meaningful price comparison, pair-wise exchange of information amongst data sources is essential, i.e., price in any sources can be compared with that in any other sources, and vice versa. Again, using the service analogy, every service needs to understand the data from all other services, which is achieved by performing conversions between the different standards used by heterogeneous services. Our scalability analysis will be focused on the number of conversions needed in each approach.

**Theorem 1 - Scalability of $BF^S$**. With $N$ data sources, the number of conversions for $BF^S$ is $O(N^2)$.

*Proof*: Each source needs to perform translations with the other $N$-1 sources; there are $N$ sources, thus a total of $N(N\text{-}1)$ translations need to be in place to ensure pair-wise information exchange, which is $O(N^2)$.

**Theorem 2 - Scalability of $BF^C$**. With $n$ distinct contexts among $N$ data sources, the number of conversions for $BF^C$ is $O(n^2)$.

*Proof*: similar to proof for Theorem 1.

Note the number of conversions for $BF^C$ is quadratic with the number of distinct contexts, not the number of sources. So when multiple sources share the same context, the conversions can be shared.

**Theorem 3 - Scalability of ETL and GS Internal Standards**. With $N$ data sources, the number of conversions for ETL or GS is $O(N)$.

*Proof*: For each source there is a conversion to the internal standard and another conversion from the internal standard to the source. There are $N$ sources, so the total number of conversions is $2N = O(N)$.

**Theorem 4 - Scalability of COIN**. With $N$ data sources and an ontology that has $m$ modifiers with each having $n_i$ unique values, $i \in [1,m]$, the number of conversions for COIN is $O(mn_k^2)$, where $n_k = \max\{n_i \mid i \in [1,m]\}$; when $m$ is fixed, the number of conversions defined in COIN is $O(n_k^2)$

*Proof*: As seen earlier, conversions in COIN are defined for each modifier, not between pair-wise sources. Thus the number of conversions depends only on the variety of contexts, i.e., number of modifiers in the ontology and the number of distinct values of each modifier. In worst case, the number of conversions to be defined is $\sum_{i=1}^{m} n_i(n_i - 1)$, where $n_i$ is the number of unique values of the $i^{th}$ modifier in the ontology, which is not to be confused with the number of sources; $m$ is the number of modifiers. This is because in worst case for each modifier, we need to write a conversion from a value to all the other values and vice versa, so the total number of conversions for the $i^{th}$ modifier is $n_i(n_i\text{-}1)$. Let $n_k$=max($n_1$, …, $n_m$). When both $m$ and $n_k$ approach infinity, $\sum_{i=1}^{m} n_i(n_i - 1) = O(mn_k^2)$; for $\forall m, \sum_{i=1}^{m} n_i(n_i - 1) = O(n_k^2)$, as $n_k \to \infty$.

However, in this example, and in many practical cases, the conversion functions can be parameterized to convert between all values of a modifier. For instance, the currency conversion given in Section 3 can convert between any two currencies using the external relation *olsen*. The conversion functions for scale factors and date formats are also of this nature. Thus, only 3 of these parameterized conversion functions are necessary for converting between contexts that differ in currency, scale factor, and/or date format. The COIN approach can take advantage of these general functions because the overall conversion program between any two contexts is automatically generated. The hard-wiring approaches can not benefit from this because all the pair-wise conversions still need to be programmed, even if most of the programs merely make calls to general functions using different parameters.

In worst case, the COIN approach needs 6 conversions for the three price definitions, e.g., from base price to price with taxes, and vice versa, and the conversion between the other two pairs. We observe that these conversions are based on a set of simple equations, e.g., add tax to base price to yield tax included price. Most of the conversion functions are invertible, e.g., the inverse of the former function is obtained by subtracting tax from tax included price to yield base price. When multiple equations exist for conversions between different modifier values, COIN composes the conversions using its symbolic equations solver [3, 4] to reduce the number of conversion declarations needed. In the example, we have three definitions for price: (A) base price, (B) tax included price, and (C) tax and shipping & handling included price. This is modeled by using a modifier that has three unique values for *price* concept in the ontology. With known equational relationships among the three price definitions, and two conversions (1) from base_price to base_price+tax (i.e., A to B), (2) from base_price+tax to base_price + tax + shipping & handling (i.e., B to C), the COIN mediator can compute the other four conversions automatically (A to C and the three inverses). Thus the number of conversion definitions for a modifier can be reduced from $n(n\text{-}1)$ to $n\text{-}1$, where $n$ is the number of unique values of the modifier. For price in the example, $n$=3, so we only need 2 conversions. Thus we have the following corollary:

**Corollary 1 – Scalability of COIN**. When conversions can be parameterized, COIN requires $m$ conversions. Otherwise, if the conversions are invertible functions, COIN needs $\sum_{i=1}^{m}(n_i - 1)$ conversions.

Furthermore, declaring the contexts can be simplified since contexts can be inherited with optional overriding in COIN. This significantly reduces the number of necessary declarations. For example, we can define a context $k$ for a country because most vendors in the same country share the same context. If a vendor in the country differs from the other vendors only with regard to price definition, we can define its context as $k'$ and specify the price definition for the vendor; by declaring $k'$ as a sub-context of $k$, $k'$ inherits all the other context definitions context $k$. This keeps the size of the knowledge base compact while the number of sources grows. As we mentioned earlier, subtypes in the ontology inherit the modifiers and the conversion definitions of their parent types, which also helps keep the number of conversion definitions small.

TABLE 2. NUMBER OF CONVERSIONS TO ACHIEVE SEMANTIC INTEROPERABILITY AMONG 10,000 SOURCES

| Approach | General case | In the example |
|---|---|---|
| $BF^S$ | $N(N\text{-}1)$, $N$:= number of sources and receivers | ~100 million |
| $BF^C$ | $n(n\text{-}1)$, $n$:= number of unique contexts | ~ 13 million |
| ETL/GS | $2N$, $N$:= number of sources and receivers | 20,000 |
| COIN | 1) Worst case: $\sum_{i=1}^{m} n_i(n_i - 1)$, $n_i$:= number of unique values of $i^{th}$ modifier, $m$ := number of modifiers in ontology<br>2) $\sum_{i=1}^{m}(n_i - 1)$, when equational relationships exist<br>3) $m$, if all conversions can be parameterized | 1) worst: ~10,000<br>2) actual number: 5 (3 general conversions plus 2 for price) |

Table 2 summarizes the complexity of different approaches in terms of the number of conversions that need

to be specified. Even in the worst case, the COIN approach requires several orders of magnitude less conversions than the brute-force approaches. In most practical cases, the actual number is far less than all the hard-wiring approaches.

We have implemented the COIN approach in a global comparison aggregator that accesses sources from a half dozen countries [12]. When we later extended it to over a dozen countries, we only needed to update the knowledge base with assertions regarding the countries to be added and no new conversions were needed. The extended prototype application accesses over 100 data sources worldwide. This demonstrates the flexibility of the COIN approach. In [8], we discuss the use of COIN for Straight-Through-Processing in financial services.

Recent research [13, 14] extended COIN to represent and reason about semantic changes over time. For example, when comparing historic stock prices in different exchanges, some of them changed reporting currency. With the formalism and the mediation engine, these temporal changes can be captured and the semantic differences at different times (in addition to between different sources) can be automatically recognized and reconciled at run time. With these advanced features and its flexibility and scalability, COIN is ideal for implementing dynamic online services.

## V. Conclusion

Dynamic online services interact with versatile and evolving business processes and data sources, at the same time need to meet various user requirements. The technology must be flexible and scalable in reconciling semantic differences amongst these information exchange entities. In this paper, we described the COIN approach to this challenge. Our analysis shows that the COIN approach can efficiently handle large number of semantic conflicts and is flexible and scalable to meet the evolving requirements.

## Acknowledgement

This work has been supported, in part, by MITRE Corp., the MIT-MUST project, the Singapore-MIT Alliance, and Suruga Bank.

## References

[1] T. Berners-Lee, J Hendler, O. Lassila, "The Semantic Web", Scientific American, 284(5), 34-43, May 2001.

[2] S. Bressan, C. Goh, N. Levina, S. Madnick, A. Shah, M. Siegel, "Context Knowledge Representation and Reasoning in the Context Interchange System", Applied Intelligence: The International Journal of Artificial Intelligence, Neutral Networks, and Complex Problem-Solving Technologies, 12(2), pp. 165-179, 2000.

[3] A. Firat, S.E. Madnick, B. Grosof, "Financial Information Integration In the Presence of Equational Ontological Conflicts", Proceedings of the Workshop on Information Technology and Systems (WITS), Barcelona, Spain, December 14-15, 2002, pp. 211-216.

[4] A. Firat, "Information Integration using Contextual Knowledge and Ontology Merging," PhD Thesis, MIT, 2003.

[5] T. Frühwirth, "Theory and Practice of Constraint Handling Rules," J. of Logic Programming, 37, 95-138, 1998.

[6] C.H. Goh, "Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems", PhD Thesis, MIT, 1997.

[7] C.H. Goh, S. Bressan, S. Madnick, M. Siegel, "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information", ACM Trans. on Information Systems (TOIS), 13(3), 270-293, July 1999.

[8] S. Jayasena, S. Bressan, S. Madnick, "Financial Information Mediation: A case study of standards integration for Electronic Bill Presentment and Payment using the COIN mediation technology," Proceedings of the VLDB Workshop on Technologies for E-Services, Toronto, Canada, August 30 - September 1, 2004 .

[9] A.C. Kakas, A. Michael, and C. Mourlas, "ACLP: Integrating Abduction and Constraint Solving," Journal of Logic Programming, 44, pp. 129-177, 2000.

[10] M. Kiffer, G. Laussen, J. Wu, "Logic Foundations of Object-Oriented and Frame-based Languages", J. ACM, 42(4), pp. 741-843, 1995.

[11] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hubner, "Ontology-Based Integration of Information – A Survey of Existing Approaches", Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing, Seattle, USA, 4 – 5 August, 2001.

[12] H. Zhu, S.E. Madnick, M.D. Siegel, "Global Comparison Aggregation Services", WEB 2002, Barcelona, Spain.

[13] H. Zhu, S.E. Madnick, M.D. Siegel, "Reasoning about Temporal Context using Ontology and Abductive Constraint Logic Programming", Proceedings of Principles and Practice of Semantic Web Reasoning (PPSWR'04), pp90-101, St. Malo, France, September 2004.

[14] 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.