SPARQL Query Mediation over RDF Data Sources with Disparate Contexts

Xiaoqing Zheng
School of Computer Science
Fudan University
Shanghai, 201203, China
zhengxq@fudan.edu.cn

Stuart E. Madnick
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02142, USA
smadnick@mit.edu

Xitong Li
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02142, USA
xitongli@mit.edu

ABSTRACT

Many Semantic Web applications require the integration of data from distributed and autonomous RDF data sources. However, the values in the RDF triples would be frequently recorded simply as the literal, and additional contextual information such as unit and format is often omitted, relying on consistent understanding of the context. In the wider context of the Web, it is generally not safe to make this assumption. The Context Interchange strategy provides a systematic approach for mediated data access in which semantic conflicts among heterogeneous data sources are automatically detected and reconciled by a context mediator. In this paper, we show that SPARQL queries that involve multiple RDF graphs originating from different contexts can be mediated in the way using the Context Interchange (COIN) Framework.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous D*atabases* – *data translation*; H.2.4 [**Database Management**]: Systems – *query processing*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *data sharing.*

General Terms

Algorithms, Design

Keywords

Semantic heterogeneity, semantic interoperability, query mediator, data integration

1. INTRODUCTION

An increasing amount of data is published in RDF format due to the activity of the linked data community. The Web of linked data that is emerging by integrating data from different sources via URIs can be considered as a single, globally distributed dataspace [1]. With SPARQL [2], a W3C Recommendation for a query language for RDF, data from different sources can be aggregated and applications can pose complex queries over the RDF dataset, which were not possible before. The presence on the Web of such a huge distributed and autonomous RDF sources poses a great challenge when it comes to achieve semantic interoperability among heterogeneous sources and receivers.

RDF is a data model for expressing the information that needs to be processed by applications, so that it can be exchanged without loss of meaning. The data do not need to be stored in RDF but can be created on the fly from relational databases [3] or other non-RDF sources. We expect that more and more content providers will make their data available via SPARQL endpoints. However, much existing data on the Web takes the form of simple values for properties such as weights, costs, etc., and contextually dependent information such as unit and format is often omitted. Due to the openness of the Web, it is generally not safe to make the assumption that anyone accessing the value of a property will understand the units being used. For example, an U.K. site might give a height value in feet, but someone accessing that data outside the U.K. might assume that heights are given in meters. Another example is that a gallon in the U.K. (the so-called Imperial gallon) is approximately 4546 ml, while in the U.S. the "same" gallon (the so-called Winchester gallon) is 3785 ml, almost 1 liter less. The principle that such simple values are often insufficient to adequately describe these values is an important one. If the data originating from different contexts are brought together and we pose queries on the whole dataset, many semantic conflicts can happen (see a motivational example in Section 2 for more detail). Proper interpretation of RDF data would depend on information that is not explicitly provided in the RDF dataset, and hence such information may be not available to other applications that need to interpret this data.

With the above observations in mind, the goal of this paper is to illustrate the novel features of the Context Interchange mediation strategy, and to describe how the semantic conflicts in RDF data sources can be automatically detected and reconciled by query rewriting technique. Specifically, the paper makes the following contributions:

- We describe how to use the Context Interchange strategy to achieve semantic interoperability among heterogeneous RDF sources and receivers by rewriting the user SPARQL query to a mediated query. The mediated query can return the answer collated and presented in the receiver context.
- We propose a formal and logical COIN framework to model contexts, i.e., the factual statements present in a data source are true relative to the context associated with the source but not necessarily so in a different context. With the framework, the users are not burdened with the diverse data semantics in sources, all of which are declared in the context representation components and can be automatically taken into consideration by the mediator.
- A SPARQL query rewriting algorithm is described and tested with the real data from CIA Factbook[1] and DBpedia[2]. The

[1] http://www4.wiwiss.fu-berlin.de/factbook/

CONTEXT: RECEIVER

- *Currency is USD with a scale-factor of 1;*
- *Datetime is expressed in US style;*
- *Locations are expressed as city name.*

NAÏVE SPARQL QUERY

```
1:  SELECT  ?airline1  ?airline2  ?total
2:
3:  FROM NAMED <http://usairline.com/flights>
4:  FROM NAMED <http://japanairline.com/flights>
5:
6:  WHERE  {
7:    GRAPH  ?graph1
8:       {  ?airline1  fts:depDateTime  ?depDateTime1 ;
9:                     fts:arrDateTime  ?arrDateTime1 ;
10:                    fts:depCity  "Boston" ;
11:                    fts:arrCity  "Tokyo" ;
12:                    fts:price  ?price1 .  }
13:   GRAPH  ?graph2
14:      {  ?airline2  fts:depDateTime  ?depDateTime2 ;
15:                    fts:arrDateTime  ?arrDateTime2 ;
16:                    fts:depCity  "Tokyo" ;
17:                    fts:arrCity  "Shanghai" ;
18:                    fts:price  ?price2 .  }
19:
20:   FILTER ( ?arrDateTime1 < ?depDateTime2 )  .
21:   FILTER ( ?depDateTime1 >= "9:30 AM 02/09/2011" )  .
22:   FILTER ( ?arrDateTime2 <= "11:30 PM 02/10/2011" )  .
23:
24:   LET ( ?total  :=  ?price1 + ?price2 )  }
25:
26:  ORDER BY ASC(?total)
27:  LIMIT 1
```

CONTEXT: SOURCE 1

- *Currency is USD with a scale-factor of 1;*
- *Datetime is expressed in US style;*
- *Locations are expressed as IATA airport codes.*

```
# Named graph: http://usairline.com/flights
@prefix : <http://usairline.com/flights#>  .

:us339  fts:depDateTime  "12:30 PM 02/09/2011" .
:us339  fts:arrDateTime  "7:25 AM 02/10/2011"  .
:us339  fts:depCity  "BOS" .
:us339  fts:arrCity  "TYO" .
:us339  fts:price  950 .

:us512  fts:depDateTime  "9:45 AM 02/10/2011" .
:us512  fts:arrDateTime  "10:30 PM 02/10/2011" .
:us512  fts:depCity  "TYO" .
:us512  fts:arrCity  "SHA" .
:us512  fts:price  380 .
```

CONTEXT: SOURCE 2

- *Currency is JPY with a scale-factor of 1000;*
- *Datetime is expressed in xsd:dateTime type;*
- *Locations are expressed as city names.*

```
# Named graph: http://japanairline.com/flights
@prefix : <http://japanairline.com/flights#>  .

:jp241  fts:depDateTime  "2011-02-10T09:25:00Z"^^xsd:dateTime .
:jp241  fts:arrDateTime  "2011-02-10T22:05:00Z"^^xsd:dateTime .
:jp241  fts:depCity  "Tokyo" .
:jp241  fts:arrCity  "Shanghai" .
:jp241  fts:price  25 .
```

Figure 1. Example scenario

results show that the approach is promising and effective. The source code and some test cases can be downloaded from the web site http://homepage.fudan.edu.cn/zhengxq/coin/.

The rest of the paper is organized as follows. Following this introduction, a motivational example is presented to highlight the Context Interchange strategy toward semantic interoperability. The COIN framework is described by introducing the formalism in section 3. The SPARQL query mediation via query rewriting technique is explained in section 4. A preliminary implementation is introduced in section 5. Section 6 presents a brief overview of related work. The conclusions are summarized in section 7.

2. MOTIVATIONAL EXAMPLE

Consider the scenario of finding cheap airfare on the Web shown in Figure 1, deliberately kept simple for didactical reasons. In this paper, examples assume the namespace prefix bindings given in Appendix A unless otherwise stated. Data on scheduled-service flights are available in two autonomously administered data sources. We assume that the flights are described by the terms from shared vocabularies to highlight the data-level conflicts. Suppose a user looks for a one-way ticket from Boston to Shanghai with one stop in Tokyo. She wants to leave Boston after 9:30 a.m., Feb 9th, 2011 and arrive in Shanghai before 11:30 p.m., Feb 10th, 2011. This query can be formulated on the schema of the two sources as the naïve query shown in Figure 1. The query will return the empty answer without any mediation if it is executed over the given dataset.

The query, however, does not take into account the fact that both sources and receivers may operate with different contexts, i.e., they may have different assumptions on how the property values should be interpreted. Specifically, the user operates with city names and US style datetimes, while the locations are recorded using IATA airport codes in the source 1 and the source 2 assumes xsd:dateTime format. It requires that certain constraints typed by the user should be transformed properly to comply with assumptions in the contexts of data sources (for example, from "Boston" to "BOS"; from "9:30 AM 02/09/2011" to "2011-02-09T09:30:00Z"). Besides, she works with US dollars with a scale-factor of 1, whereas the source 2 reports all ticket prices in Japanese Yen with a scale-factor of 1000, which shows that the data might vary in two or more aspects (in that case, currency and scale). So there must be more than one conversion of the data. Even if these specific differences were carefully dealt with by writing a new query with appropriate datetime formats, currencies and city codes for each individual source (which might be a significant challenge for the user, especially if unfamiliar with the details of each of the multiple sources involved), the result still

[2] http://dbpedia.org/

would be misleading. For the source 1, the graph pattern of naïve query needs to be rewritten to the following one:

```
   GRAPH  ?graph1
      {  ?airline1  fts:depDateTime  ?depDateTime1;
                    fts:arrDateTime  ?arrDateTime1;
                    fts:depCity  "BOS";
                    fts:arrCity  "TYO";
                    fts:price  ?price1 .  }
     GRAPH  ?graph2
      {  ?airline2  fts:depDateTime  ?depDateTime2;
                    fts:arrDateTime  ?arrDateTime2;
                    fts:depCity  "TYO";
                    fts:arrCity  "SHA";
                    fts:price  ?price2 .  }
```

But the result below still is incorrect because it is not cheapest airfare:

airline1	airline2	total
us339	us512	1330.00

The above result is a solution corresponding to the way in which the query's pattern matches the RDF data, all from the source 1. For the source 2, the result returned is empty because there is only one flight from "Tokyo" to "Shanghai". SPARQL, however, can be used to express queries across diverse data sources, and a SPARQL query can match different parts of the query pattern against different graphs. In the example, it does not work by rewriting the naïve query into two separate queries, one for executing over source 1 and one for source 2, and combining the results. A solution is required to make these context conversions in dynamic way, depending on which data sources is involved. Further, the answers should be transformed so that they conform to the context of the user.

In the Context Interchange system, the semantics of data of those present in a source, or of those expected by a receiver can be explicitly represented in the form of a *context theory* and a set of *context statements* with reference to a *context ontology* (see Section 3). Queries submitted to the system would be rewritten into the mediated queries by a *Context Mediator*, in which the semantic conflicts between the sources and the receiver would be automatically recognized and reconciled.

The naïve query can be transformed to the mediated query by the rewriting algorithm described in the Section 4. This mediated query considers all potential conflicts between the sources and the receiver when matching, restricting and testing values. Moreover, the answers returned can be transformed so that they conform to the context of the receiver. The mediated query, when executed, returns the "correct" answer below, which helps the user to make the right choice.

airline1	airline2	total
us339	jp241	1255.59

Exchange rate: 100 USD = 8181 JPY

In the COIN system, *query mediation* and *query answering* are separated as shown in the above example. Since the mediated queries encode all the necessary data transformations, they can be executed by existing query engines such as ARQ[3], AllegroGraph[4], OpenRDF Sesame[5], taking advantage of their sophisticated query optimizers. We have shown only one user in this scenario. However, other users with different contexts could also issue their queries and get the results in their own contexts by simply declaring or choosing their contexts. Similarly, other RDF data sources can be added to the system with the declaration of their contexts, and queries over multiple sources with different contexts could be mediated in the similar way.

3. CONTEXT REPRESENTATION

The purpose of knowledge representation in COIN is to provide a formal way of making explicit disparate assumptions about data. Knowledge about the source and user contexts is declared under a formal, logical COIN framework consisting of the following four core components (see Figure 2):

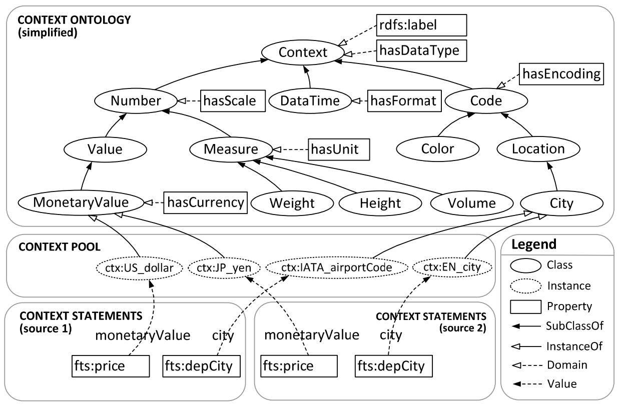

Figure 2. An illustration of the COIN framework

- $\mathcal{O}$ = <$\mathcal{C}$, $\mathcal{P}$>, the *Context Ontology*, is a description of generic concepts $\mathcal{C}$ that would be interpreted differently across data sources and receivers (e.g. "*MonetaryValue*"), and the conflict dimensions (or *modifiers*) that are defined as the properties $\mathcal{P}$ for these concepts (e.g. "*hasCurrency*").
- $\mathcal{A}$, the *Context Pool*, is a set of instantiations of the context ontology. Value assignments are made for each modifier to explicate the meaning of a concept in a data source or receiver (e.g. ctx:JP_yen coin:scale "1000"^^xsd:integer).
- $\mathcal{M}$, the *Context Mappings*, defined a set of *context statements* that associate the sources or receivers with their contexts. A context statement can identify the correspondences between a property used in the sources and a context instance defined in $\mathcal{A}$ (e.g. fts:price coin:monetaryValue ctx:JP_yen).
- $\mathcal{F}$, the *Conversion Function Bindings*, specified which functions should be used to make data transforms. *Conversion functions* are defined to achieve conversions between different contexts. For each modifier at least one conversion function will be defined to transform a value in one (source) context into a corresponding value in another (target) context (e.g. fn:cvt_currency("JPY", "USD", 25000), which converts 25000 Japanese Yen to the equivalent US dollars).

In the remaining subsections, we describe each of the components in turn with examples. RDF and OWL have been used to describe context knowledge in the framework. The adoption of RDF and OWL provides us with greater flexibility in representing, reusing and exchanging data semantics captured in different contexts.

[3] http://jena.sourceforge.net/ARQ/
[4] http://www.franz.com/
[5] http://www.openrdf.org/

3.1 Context Ontology

Context ontology is a collection of generic concepts (or classes in the OWL language), which provides a common type system for describing data semantics exchanged between disparate systems. A context ontology corresponding to the motivational example in Section 2 can be seen in Figure 2. Concepts, denoted by $\mathcal{C}$, are depicted by ellipses, and the "*Context*" is the special concept from which all other concepts inherit. Concepts may have properties, called *modifiers* and denoted by $\mathcal{P}$, which serve as annotations that make explicit the semantics of data in different contexts.

Contexts are the actual specializations of the concepts subject to multiple meanings across sources and receivers. For sources, the contexts are defined as the specializations used for the underlying data values. For receivers, on the other hand, the contexts are defined as the specializations assumed in viewing the data values. These specializations could be the representation of date or the number scale being used.

The modifiers, as properties, will be inherited by the sub-concept relations from its ancestors. A concept can have multiple modifiers, each of which indicates an orthogonal dimension of the variations. For example, the concept "*MonetaryValue*" has four modifiers, two of which are "*hasCurrency*" introduced by the "*MonetaryValue*" and "*hasScale*" inherited from the "*Number*", which indicates that its instances could be interpreted according to two dimensions: scale factor and money currency. All the concepts and their properties of the context ontology are defined in the namespace bound to the prefix "coin".

3.2 Context Pool

Context pool contains a set of instantiations of the concepts in the context ontology. As we mentioned above, modifiers are special properties that affect the interpretation of data values. The context ontology defines what types of modifiers apply to which concepts. A *context instance* or *individual* is defined by a set of RDF statements that determine the values of modifiers. Such statements are defined over the domain $\mathcal{A} \times \mathcal{P} \times (\mathcal{I} \cup \mathcal{L})$, where $\mathcal{I}$ is a set of the IRIs, and $\mathcal{L}$ is a set of the RDF literals. If $\{s, p, o\}$ is a statement about a context instance s, and s belongs to a concept $C \in \mathcal{C}$, the property $p \in \mathcal{P}$ should be able to apply to the class C, and a value o must be in the range of the property p.

For example, a context instance "ctx:US_dollar" can be described with the following statements:

```
ctx:US_dollar  coin:hasDataType  xsd:long  .
ctx:US_dollar  coin:hasScale  "1"^^xsd:integer  .
ctx:US_dollar  coin:hasCurrency  "USD"  .
```

These modifier assignments explicitly indicate that any data value associated with the "ctx:US_dollar" is in US dollars with a scale-factor of 1 and is represented by the typed literal "xsd:long". All the context instances are defined in the namespace bound to the prefix "ctx". The objects (xsd:long, "1", and "USD") in the above RDF statements are called *modifier values*.

We can declare new context instances or reuse which are already defined in the pool. For each concept of the context ontology that is interpreted differently by sources and receivers, modifiers are introduced to explicate those differences. The advantage of this approach is that it allows conflicts between sources and receivers to be introduced gradually as they are discovered. Many conflicts emerge later in the integration process as more sources and users are incorporated in the system. If all sources and receivers hold a common meaning for a given concept, no modifier is required at that time. When that situation changes at a later time, modifiers can be introduced to handle the variations.

3.3 Context Mappings

Context mappings provide the articulation of the data semantics for the sources and receivers, which are often implicit in the given contexts. For each concept in the context ontology, a predicate with the same name as the concept was defined (for example, the predicate coin:monetaryValue needs to be defined for the concept "*MonetaryValue*"). These predicates are used to associate the properties used in the sources with the corresponding context instances in order to make explicit the data semantics of the values of the properties. The statements of the context mappings are defined over the domain $(\mathcal{R} \cup \mathcal{D}) \times \mathcal{T} \times \mathcal{A}$, where $\mathcal{R}$ is a set of the IRIs that represent the properties appeared in the data sources, $\mathcal{D}$ is a set of IRIs used to identify specific applications, and $\mathcal{T}$ is a set of predefined vocabulary that have the same names as the concepts defined in the context ontology, but begin with a lower case letter. $\mathcal{T}$ is used to indicate the types of the context instances assigned in the context mapping statements.

For example, the fact that the values of the "fts:price" property are reported in US dollars using a scale-factor of 1 is made explicit by the following statement:

```
fts:price  coin:monetaryValue  ctx:US_dollar  .
```

The "ctx:US_dollar" is a context instance of the "MonetaryValue" concept, and was defined in the example of Section 3.2. If the data is structured in the form: *subject-predicate*-[ ]-*predicate-object*, using an intermediate blank node, as the example similar to the following definition:

```
item  hasPrice  _:blanknode
_:blanknode  price  "100"
_:blanknode  currency  "USD "
```

In that case, the context mapping will be defined by associating the "price" property with the appropriate context instance to make explicit the data semantics of the value "100" of that property. For the sources, the context mappings are defined by attaching the context instances to the properties, which could be considered as some extensions to the RDF Schema vocabulary. The extensions support the description of context information indicating how the values of a given property are interpreted.

We cannot assume that the users have intimate knowledge of the data sources being accessed since this assumption is generally non-tenable when the number of systems involved is large and when changes are frequent. The users should remain isolated from underlying semantic heterogeneity, i.e., they are not required to be sufficiently familiar with the properties in different schemas (so as to construct a query). There are some graphic tools (Gruff[6] for example) to assist the users to create SPARQL queries. The COIN enables the users to identify their contexts independent of the data sources. Receiver's contexts can be declared by assigning the instances in $\mathcal{A}$ to $\mathcal{D}$ with the aid of vocabulary in $\mathcal{T}$. For example, a user could declare that she use US dollars by the following statement:

```
:flight  coin:monetaryValue  ctx:US_dollar  .
```

[6] http://www.franz.com/agraph/gruff/

The ":flight" is used to indicate an application domain, so a meaningful name is recommended. A user is allowed to make different context definitions for different applications.

All context statements for a data source or a receiver should be made in a separate namespace. The correspondences between the sources or receivers and their context definitions then need to be further identified. We assume that all context statements about the source 1 were described at <http://coin.mit.edu/sources/usairline>. The following triple asserts that the context of the source 1 is defined in the RDF graph <http://coin.mit.edu/ sources/usairline>, where the source 1 is identified by <http:// usairline.com/flights>:

<http://usairline.com/flights> coin:hasContext <http://coin.mit.edu/sources/usairline> .

Such RDF statements, called *context bindings*, are defined over the domain $\mathcal{I} \times$ coin:hasContext $\times$ $\mathcal{I}$. The context bindings, denoted by $\mathcal{B}$, will be added in the mediated query to retrieve the context mappings of the data sources and/or receivers (more about these in the section 4).

3.4 Conversion Function Bindings

The preceding statements are not yet sufficient for resolving conflicts of data present in disparate contexts, since we have to define how values in one (source) context are to be reported in another (target) context with different assumptions (i.e., modifier values). This is accomplished via the introduction of conversion functions that are defined for each modifier between two different modifier values. A general representation of conversion functions is shown as follows:

fn:cvt_modifier(*mvs*, *mvt*, *vs*)

where *mvs* and *mvt* are two distinct values of the modifier in the source and target contexts respectively. The function returns the equivalent value *vt* that complies with assumptions in the target context for the source value *vs*. For example, a scale conversion fn:cvt_scale could be defined by multiplying a given value with the appropriate ratio as shown below:

fn:cvt_scale: $vt = vs * mvs / mvt$

Note that the conversion function will return *vt* directly if *mvs* or *mvt* is undefined.

In some cases, ancillary data sources may be used for defining appropriate functions. For instance, currency conversions need to be supported by external data sources that provide the exchange rate between two different currencies. Atomic conversions can be composited to construct composite conversions. As we mentioned, each modifier captures one aspect of interpretation that may vary across contexts. After invoking an atomic conversion, the source value is transformed to a (intermediate) context that is the same as the target context in terms of this aspect; by invoking the atomic conversions consecutively, the value is converted through a series of intermediate contexts, each having one aspect being different from the target context; it reaches the target context in the end when no aspect is different from the target context. Thus, in the case of "*MonetaryValue*" that has two modifiers, *currency* and *scale*, we may have:

fn:cvt_scale(1000, 1, fn:cvt_currency("JPY", "USD", 25))

Hence, if the function for currency returns the value 0.30559, it will be rewritten to 305.59 by the scale conversion function. All the COIN components for the motivational example are given in Appendix B except the context ontology that has already shown in Figure 2.

Conversion function bindings, denoted by $\mathcal{F}$, are defined by a set of RDF triples that determine which functions can be used to make necessary data transforms for the modifiers of the concepts. Such statements are defined over the domain $\mathcal{C} \times \mathcal{P} \times \mathcal{N}$, where $\mathcal{N}$ is a set of IRIs used to identify and retrieve conversion functions.

4. SPARQL QUERY MEDIATION

The goal of the COIN framework is to provide a formal, logical basis that allows for the automatic mediation of queries such as those described in Section 2. The *semantic conflicts* would happen when the RDF literals typed by a receiver are attempted to match against source graphs or the RDF terms from different graphs are compared. In this section, we describe the process of rewriting a naïve SPARQL query (i.e. query ignoring semantic differences between sources and/or receivers) to a mediated query with all the semantic conflicts reconciled and the query results transformed appropriately according to user expectation.

4.1 Well-Formed Query

Given a naïve SPARQL query, context mediation is bootstrapped by transforming this user query into a logically equivalent query. The mediation process starts by converting the naïve query into its well-formed query that must satisfy the following requirements:

- All the data sources are introduced as named graphs by using the FROM NAMED clause.
- All the graph patterns (i.e., basic or group graph patterns) are defined inside the scope of GRAPH keywords.
- There is no such variable that is used in two different GRAPH clauses.

The translation to meet the first two requirements is obviously a trivial exercise. Note that the GRAPH keyword could be followed by a variable or an IRI. If a variable is provided, it is not possible to know in advance which named graph will be matched since the variable will range over the IRIs of all the named graphs in the query's RDF dataset. Query variables in SPARQL queries have global scope and use of a given variable name anywhere in a query identifies the same variable. However, a variable shared by different named graphs might be bound to the logically-equivalent term, but with different representations in different contexts. So the last requirement is necessary and can always be guaranteed by renaming variables and adding the corresponding FILTERs to the WHERE clause. For example, if a ?var variable is used across two different GRAPH clauses, one variable would be renamed ?nvar, and a FILTER(sameTerm (?var ,?nvar)) constraint will be created.

4.2 Semantic Conflicts Detection

The semantic conflicts are detected by a context mediator through the comparison of context statements corresponding to the sources and receivers engaged in query patterns. The algorithm 1 shows how to detect the potential conflicts among heterogeneous sources and receivers for given a pair of properties, where a three-place notation $Triple_g(s, p, o)$ is used to represent a typical RDF triple $<s, p, o>$ and if the subscript g is given it indicates that the triple is defined under the named graph g. The subscript g could be an IRI or a variable. In this paper, $Triple_g(s, p, o)$ is also used to denote a triple pattern.

Note that the properties P_1 and P_2 are not required to be associated with the same concept in $\mathcal{C}$, which makes it possible to express the constraints like FILTER (?price < ?weight). If the properties of ?price and ?weight have a common modifier,

Algorithm 1. Detection of semantic conflicts between sources and/or receivers for a given property

Input:	$\mathcal{K}$:	COIN components, $\mathcal{K} = <\mathcal{O}, \mathcal{A}, \mathcal{M}, \mathcal{F}>$ including $\mathcal{B}$
	S_1 :	a data source IRI (could be a variable)
	S_2 :	another data source IRI (could be a variable) or an IRI used to identify a receiver
	P_1 :	a property IRI used in S_1
	P_2 :	a property IRI used in S_2 or an IRI used to identify a application domain if S_2 is a receiver identifier
Output:	$\mathcal{SC}$:	a set of semantic conflicts detected

```
1   $\mathcal{SC} = \emptyset$
2:  $def_1$ = { $def$ | $Triple(S_1$, coin:hasContext, $def) \in \mathcal{B}$ }
3:  $def_2$ = { $def$ | $Triple(S_2$, coin:hasContext, $def) \in \mathcal{B}$ }
4:  $ctx_1$ = { $ctx$ | $Triple_{def1}(P_1, concept, ctx) \in \mathcal{M}$ }
5:  $C_1$ = { $concept$ | $Triple_{def1}(P_1, concept, ctx_1) \in \mathcal{M} \wedge concept \in \mathcal{T}$ }
6:  if  $S_2$ is a data source identifier  then
7:      $ctx_2$ = { $ctx$ | $Triple_{def2}(P_2, concept, ctx) \in \mathcal{M}$ }
8:      $C_2$ = { $concept$ | $Triple_{def2}(P_2, concept, ctx_2) \in \mathcal{M}$
                                 $\wedge\ concept \in \mathcal{T}$ }
9:  else $ctx_2$ = { $ctx$ | $Triple_{def2}(P_2, C_1, ctx) \in \mathcal{M}$ }
10:     $C_2 = C_1$
11: for each  $modifier \in$ { $p$ | $Triple(ctx_1, p, mv) \in \mathcal{A} \wedge p \in \mathcal{P}$ }  do
12:     $mv_1$ = { $mv$ | $Triple(ctx_1, modifier, mv) \in \mathcal{A}$ }
13:     $mv_2$ = { $mv$ | $Triple(ctx_2, modifier, mv) \in \mathcal{A}$ }
14:     if  $mv_1 \neq mv_2$ and $mv_1 \neq$ NULL and $mv_2 \neq$ NULL  then
15:         $\mathcal{SC} = \mathcal{SC} \cup$ { $S_1, S_2, P_1, P_2, C_1, C_2, modifier, mv_1, mv_2$ }
16: return  $\mathcal{SC}$
```

"*hasScale*" for example, and different modifier values, this conflict will also be detected and be used to construct the conversion function (see Section 4.3). Recall that the users are not required to be sufficiently familiar with the underlying schemas of data sources, and their contexts are declared by assigning the context instances to the concepts $\mathcal{C}$ via $\mathcal{T}$. If S_2 identifies a receiver, the context instance ctx_2 could be retrieved directly by looking up $\mathcal{M}$ for the concept C_1 as shown in line 9 of the algorithm 1.

The algorithm 1 can only be used to statically detect the semantic conflicts between sources and/or the receiver. Sometimes it is impossible to know in advance which sources the query pattern will be matched until the query is executed, not to mention the contexts of the sources. For example, the following group graph pattern in the query of the motivational example can be matched against both source 1 and 2 (assuming the city names, "Shanghai" and "Tokyo", are automatically transformed so that they conform to the contexts of the corresponding sources).

```
GRAPH  ?graph2
   {  ?airline2  fts:depDateTime  ?depDateTime2 ;
                 fts:arrDateTime  ?arrDateTime2 ;
                 fts:depCity  "Tokyo" ;
                 fts:arrCity  "Shanghai" ;
                 fts:price  ?price2 .  }
```

The trick is that we are able to access the graph name by making a variable (i.e., ?graph2 used in the above pattern) bound to IRIs of the graph being matched, and then use the graph name to obtain its context definition via context bindings (see Section 3.3). SPARQL queries can be used to find all the potential semantic conflicts instead of the algorithm 1, and the algorithm can be trivially translated to the equivalent SPARQL query shown in Figure 3. If this query is applied to the source 1 and 2 for the property fts:price, two semantic conflicts would be detected as shown in Table 1. Note that we just need to detect the conflicts that are required to make necessary conversions for queries.

```
SELECT ?concept1 ?concept2 ?modifier ?mv1 ?mv2
WHERE {
  S1 coin:hasContext ?def1
  S2 coin:hasContext ?def2
  GRAPH ?def1 { P1 ?concept1 ?ctx1 }
  GRAPH ?def2 { P2 ?concept2 ?ctx2 }
  ?ctx1 ?modifier ?mv1 .
  ?ctx2 ?modifier ?mv2 .
  FILTER(!sameTerm(?mv1, ?mv2)) } .
```

Figure 3. The equivalent SPARQL query for Algorithm 1

Table 1. The semantic conflicts detected in the motivational example for the property fts:price

modifier	mv_1	mv_2
coin:hasScale	1	1000
coin:hasCurrency	USD	JPY

For the same reason, in some cases it is not possible to know in advance how to interpret the value of a property because we cannot know in advance the sources of the value. However, in most cases, we are able to know with which concept the property was associated by looking up the context mappings. With the concept name, we can get all the modifiers applied to the concept, access the values of the modifiers, and then use them to construct the appropriate conversion functions. An OPTIONAL keyword will be used when we cannot know in a static way whether certain modifiers or modifier values are defined in the context statements. Notice that if a concept has no modifier, there is no conversion function defined for the concept, so the values are assumed not to vary across any context.

Algorithm 2. Conversion function construction

Input:	$\mathcal{K}$:	COIN components, $\mathcal{K} = < \mathcal{O}, \mathcal{A}, \mathcal{M}, \mathcal{F}>$ including $\mathcal{B}$
	$\mathcal{SC}$:	a set of semantic conflicts detected by the algorithm 1
	MV :	a data value or a variable in source context
	D :	a boolean variable to indicate conversion direction
Output:	CF :	a composite conversion (default value is NULL)

```
1:  for each { $S_1, S_2, P_1, P_2, C_1, C_2, modifier, mv_1, mv_2$ } $\in \mathcal{SC}$  do
2:      if   $CF$ = = NULL   then
3:           $function$ = { $f$ | $Triple(C, modifier, f) \in \mathcal{F} \wedge C \in \mathcal{C} \wedge$
                  ( $C = C_1 \vee C$ is a closest super-concept of $C_1$ ) }
4:           if   $D$ = = TRUE
5:                $CF = function(mv_1, mv_2, MV)$
6:           else $CF = function(mv_2, mv_1, MV)$
7:      else $function$ = { $f$ | $Triple(C, modifier, f) \in \mathcal{F} \wedge C \in \mathcal{C} \wedge$
                  ( $C = C_1 \vee C$ is a closest super-concept of $C_1$ ) }
8:           if   $D$ = = TRUE
9:                $CF = function(mv_1, mv_2, CF)$
10:          else $CF = function(mv_2, mv_1, CF)$
11: return  $CF$
```

4.3 Conversion Function Construction

The conversion functions are introduced to define how values of a given concept are transformed between different contexts. In the COIN framework, an "ontology-based" representation is adopted where conversion functions are attached to concepts in different contexts. This mechanism allows for greater sharing and reuse of semantic encoding. For example, the same concept may appear many times in different properties (e.g., consider the concept "*MonetaryValue*"). Rather than writing a function conversion for each property that redundantly describes how different reporting currencies are resolved, we can simply associate the conversion function with the concept "*MonetaryValue*".

In addition, when a property involves two or more conflicts, a composite function can be constructed to reconcile those conflicts by the algorithm 2. It is also scalable because it can compose all necessary conversions using a small set of component conversions. The composition can be obtained as a series of invocations on the conversion function defined for each modifier pertaining to the concept in the context ontology.

The function-finding method in the algorithm 2 (line 3 and 7) at first will try to find the specific conversion function defined for the modifier of the concept. If it cannot be found, the method will try to find the function defined for the same modifier under the direct super-concept. The step will repeat until a function is retrieved or it reaches the top "*Context*" concept. For example, the function cvt_format_dateTime( ) is defined for the modifier "*hasFormat*" under the concept "*dateTime*" that has no sub-concepts, while cvt_scale( ) function is defined under the concept "*Number*" and the function can be used by all its sub-concepts such as "*Weight*", "*Height*" and "*MonetaryValue*" for scale-factor adjustment.

Note that the translation from one context to another is embedded in conversion functions present in individual context theories, and they are not part of the context ontology. This means that there is greater scope for different users to introduce conversion functions which are most appropriate for their purposes without requiring these differences to be accounted globally. For example, different currency exchange system would be used in different countries or for different purposes.

The modifiers of the concept are called orthogonal if the value derived from its composite conversion is not affected by the order in which the component conversions are invoked. For example, the *currency* and *scale* modifiers of the "*MonetaryValue*" concept are orthogonal. We will get the same value either by converting currency first followed by scale-factor adjustment or by adjusting scale-factor first followed by currency conversion. For any pair of modifiers that are not orthogonal, the component conversions are required to be invoked in a particular order to return right value. The order depends on how the component conversion is specified. An in-depth discussion on this issue can be found in [6].

4.4 Query Rewriting

The context mediator uses the algorithm 3 to undertake the role of detecting and reconciling potential conflicts at the time a query is submitted. The following parts in a query might be rewritten to the corresponding forms that all semantic conflicts, when detected, are resolved:

- SELECT: the answers returned should be further transformed so that they conform to the context of the receiver.
- WHERE: the constants should be transformed to comply with assumptions in the source contexts.
- EXPRESSION and FUNCTION: one of two arguments might be transformed so that the two arguments conform to the same context.

First, the context mappings $\mathcal{M}$ and context pool $\mathcal{A}$ will be added into the RDF dataset as named graphs (line 4). A mediated query is executed against the RDF dataset that comprises one or more data sources and the two graphs generated from $\mathcal{M}$ and $\mathcal{A}$. In the

Algorithm 3. SPARQL query rewriting

```
Input:   $\mathcal{K}$ :  COIN components, $\mathcal{K} = <\mathcal{O}, \mathcal{A}, \mathcal{M}, \mathcal{F}>$ including $\mathcal{B}$
         $WQ$ : a well-formed SPARQL query
         $r$ :  an IRI used to identify a receiver
         $a$ :  an IRI used to identify a application domain
Output:  $MQ$ : an mediated SPARQL query

1:  $MQ = WQ$
2:  $pattern = \emptyset$
3:  parse the query $WQ$ and extract its triple patterns into $bgp$
4:  $\mathcal{M}$ and $\mathcal{A}$ are included into the dataset of $MQ$ as named graphs
5:  for each $var \in \{ v \mid v$ is a RESULT variable $\wedge$
                $Triple_g(s, p, v) \in bgp \}$ do
6:      if $\mathcal{SC}$ = Algorithm1($\mathcal{K}, g, r, p, a$) $\wedge \mathcal{SC} \neq \emptyset$ then
7:          $pattern = pattern \cup$ PatternCreator($g, r, p, a, \mathcal{SC}$) / $bgp$
8:          $CF$ = Algorithm2($\mathcal{K}, \mathcal{SC}, v$, TRUE)
9:          adds the assignment LET ( $nvar := CF$ ) into $MQ$ and
            replace $var$ with $nvar$ in the SELECT clause, where $nvar$
            is a new variable never used before.
            /* if the variable $v$ is introduced by a LET clause, the
            context conversion will be done in the LET clause */
10: for each $cont = \{ c \mid c$ is a constant $\wedge$ $Triple_g(s, p, c) \in bgp \}$ do
11:     if $\mathcal{SC}$ = Algorithm1($\mathcal{K}, g, r, p, a$) $\wedge \mathcal{SC} \neq \emptyset$ then
12:         $pattern = pattern \cup$ PatternCreator($g, r, p, a, \mathcal{SC}$) / $bgp$
13:         $CF$ = Algorithm2($\mathcal{K}, \mathcal{SC}, cont$, FALSE)
14:         adds the FILTER( $nvar := CF$ ) into $MQ$ and replace
            $cont$ with $nvar$ in $Triple_g(s, p, c)$, where $nvar$ is a new
            variable never used before.
15: for each EXPRESSION ( $expr_1$ op $expr_2$ ) or
             FUNCTION( $expr_1, expr_2$ ) do
16:     if $expr_1$ is a constant $\wedge$ $expr_2$ is a variable $\wedge$
            $Triple_g(s, p, expr_2) \in bgp$ $\wedge$
            $\mathcal{SC}$ = Algorithm1($\mathcal{K}, g, r, p, a$) $\wedge \mathcal{SC} \neq \emptyset$ then
17:         $pattern = pattern \cup$ PatternCreator($g, r, p, a, \mathcal{SC}$) / $bgp$
18:         $CF$ = Algorithm2($\mathcal{K}, \mathcal{SC}, expr_1$, FALSE)
19:         replace $expr_1$ with $CF$ in the expression or function
20:     if $expr_2$ is a constant $\wedge$ $expr_1$ is a variable $\wedge$
            $Triple_g(s, p, expr_1) \in bgp$ $\wedge$
            $\mathcal{SC}$ = Algorithm1($\mathcal{K}, g, p, r, a$) $\wedge \mathcal{SC} \neq \emptyset$ then
21:         $pattern = pattern \cup$ PatternCreator($g, r, p, a, \mathcal{SC}$) / $bgp$
22:         $CF$ = Algorithm2($\mathcal{K}, \mathcal{SC}, expr_2$, FALSE)
23:         replace $expr_2$ with $CF$ in the expression or function
24:     if $expr_1$ is a variable $\wedge$ $expr_2$ is a variable $\wedge$
            $Triple_{g1}(s_1, p_1, expr_1) \in bgp$ $\wedge$
            $Triple_{g2}(s_2, p_2, expr_2) \in bgp$ $\wedge$
            $\mathcal{SC}$ = Algorithm1($\mathcal{K}, g_1, g_2, p_1, p_2$) $\wedge \mathcal{SC} \neq \emptyset$ then
25:         $pattern = pattern \cup$
                    PatternCreator($g_1, g_2, p_1, p_2, \mathcal{SC}$) / $bgp$
26:         $CF$ = Algorithm2($\mathcal{K}, \mathcal{SC}, expr_1$, TRUE)
27:         replace $expr_1$ with $CF$ in the expression or function
28: adds $pattern$ into $MQ$
29: return $MQ$
```

```
Function PatternCreator($S_1, S_2, P_1, P_2, \mathcal{SC}$)
1:  $pattern = \emptyset$
2:  $pattern = \{$ $Triple(S_1$, coin:hasContext, $def_1)$ ,
                $Triple(S_2$, coin:hasContext, $def_2)$ ,
                $Triple_{def1}(P_1, C_1, ctx_1)$ ,
                $Triple_{def2}(P_2, C_2, ctx_2)$ $\}$
3:  for each $\{ S_1, S_2, P_1, P_2, C_1, C_2, modifier, mv_1, mv_2 \} \in \mathcal{SC}$
4:      $pattern = \{$ $pattern$ ,
                   $Triple(ctx_1, modifier, mv_1)$ ,
                   $Triple(ctx_2, modifier, mv_2)$ $\}$
5:  return $pattern$
```

current SPARQL core standard, SELECT queries only project out variables bound in the query and there is no way to return the values of expressions over result bindings. The mediated queries need the ability to project expressions rather than just variables because the results need to be transformed to comply with assumptions in the receiver context. An example is returning the total cost of two air tickets. LET assignments are used to enable transformation on the results by using the conversion functions (line 9). For the same reason, we also use FILTER clauses in the line 14 to transform the constants in the queries to comply with assumptions in the sources contexts.

Considering the naïve query in the motivational example, the first triple pattern encountered that needs to be processed is the one:

```
   GRAPH ?graph1  { ?airline1  fts:depCity  "Boston" . }
```

The line 10-14 of the algorithm 3 will transform this triple into the following patterns.

```
i:    GRAPH  ?graph1 { ?airline1  fts:depCity  ?depCity . }
ii:   GRAPH  <http://coin.mit.edu/bindings/flight>
iii:    {  ?graph1  coin:hasContext  ?def1  .
iv:        coin:receiver  coin:hasContext  ?recv .  }
v:    GRAPH  ?def1 {  fts:depCity  coin:city  ?ctx1 . }
vi:   GRAPH  ?recv {  :flight  coin:city  ?ctx2 . }
vii:  GRAPH  <http://coin.mit.edu/contexts>
viii: {  ?ctx1  coin:hasEncoding  ?mv1 .
ix:       ?ctx2  coin:hasEncoding  ?mv2 . }
x:    FILTER ( ?depCity = fn:cvt_encoding_city(?mv2, ?mv1,
                                               "Boston") ) .
```

A new variable ?depCity was generated and replaced "Boston" at the line i. The context statements of the data sources can be accessed dynamically via the variables ?graph1 that will bound to IRIs of the sources being matched (line iii). The pattern of the line iv was used to retrieve the receiver's context definition. The two context instances of the concept "*City*" will be obtained by the line v-vi. One was associated with the property fts:depCity of the data sources; the other was referenced in the receiver's context statements. The modifier values of those two context instances can be retrieved by the line viii-xi, which will be taken as inputs to the conversion function fn:cvt_encoding_city. In the FILTER clause of the line x, the English city name "Boston" might be transformed to comply with the contexts of the data sources. If the variable ?graph is bound to the IRI of the source 1 (USA airlines), "Boston" will be rewritten into "BOS". If ?graph is bound to the IRI of the source 2 (Japan airlines), "Boston" will stay unchanged.

For the result variables of the naïve query, only the variable ?total needs to be processed because it is the sum of the prices of the two tickets, ?price1 and ?price2 that occur as objects in the triple patterns. The following fragment of the mediated query will be generated by the line 6-9 of the algorithm 3 for the variable ?total.

```
i:    GRAPH  <http://coin.mit.edu/bindings/flight>
ii:     {  ?graph1  coin:hasContext  ?def1  .
iii:        ?graph2  coin:hasContext  ?def2  .
iv:         coin:receiver  coin:hasContext  ?recv .  }
v:    GRAPH  ?def1 {  fts:price  coin:monetaryValue  ?ctxp1 . }
vi:   GRAPH  ?def2 {  fts:price  coin:monetaryValue  ?ctxp2 . }
vii:  GRAPH  ?recv {  :flight  coin:monetaryValue  ?ctxr }
viii: GRAPH  <http://coin.mit.edu/contexts>
ix:     {  ?ctxp1  coin:hasScale  ?mvs1 .
x:          ?ctxp1  coin:hasCurrency  ?mvc1 .
xi:         ?ctxp2  coin:hasScale  ?mvs2 .
xii:        ?ctxp2  coin:hasCurrency  ?mvc2 .
xiii:       ?ctxr  coin:hasScale  ?mvsr .
xiv:        ?ctxr  coin:hasCurrency  ?mvcr .  }
xv:  LET ( ?total :=  fn:cvt_currency(?mvc1, ?mvcr,
                      fn:cvt_scale(?mvs1, ?mvsr, ?price1))  +
                      fn:cvt_currency(?mvc2, ?mvcr,
                      fn:cvt_scale(?mvs2, ?mvsr, ?price2)) )
```

The line i-iv was generated to retrieve the context definitions of the two possible data sources (one is ?graph1 in which ?price1 will be retrieved; the other is ?graph2 that will provide ?price2) and the receiver. Note that the duplicate patterns will be deleted from the mediated query when the patterns are merged into the previously-generated ones. The context instances and their modifier values will be obtained by the line v-xiv. To rewrite the LET assignment (line xv), the algorithm 2 will be called twice, each for ?price1 and ?price2, to make the sum of the two prices comply with assumptions in the receiver context.

5. IMPLEMENTATION

We use Jena[7] to implement COIN SPARQL framework to provide a demonstration of the feasibility of Context Interchange strategy. As shown in Figure 4, queries submitted to the system will be intercepted by a *Context Mediator*, which rewrites the user query into a mediated query and pass it to ARQ, the SPARQL query engine for Jena. COIN also can easily be plugged in to other existing SPARQL query engines to take advantage of their state of the art query optimization and execution techniques. We tested the solution with the real data from CIA Factbook and DBpedia.

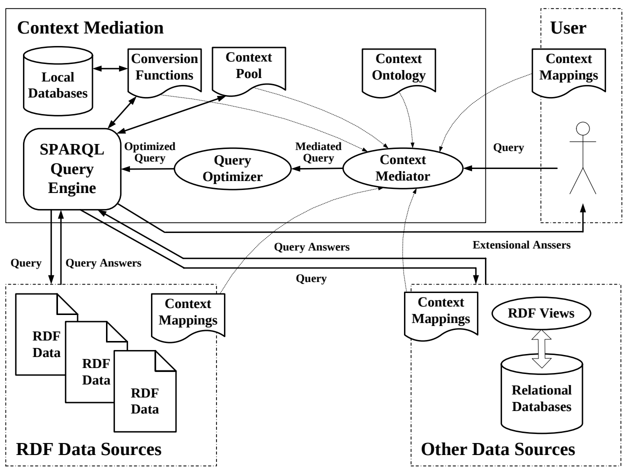

Figure 4. Architecture of a COIN SPARQL system

OWL is used to represent the context ontology, and the pool and mapping components are described by using RDF. All conversion functions now are implemented as FILTER (extension) functions, which use the advanced feature of ARQ that goes beyond the core SPARQL specification. The conversion functions extend the abstract class FunctionBase3 that takes three parameters. The query engine can dynamically call the functions based on the function URIs. This is done by either registering it or using the fake java: URI scheme.

Conversion functions are defined as parametric functions that can express the conversion between all possible modifier value pairs, not only a specific one. For example, fn:cvt_ currency can be used to make currency conversion among any different currencies. For "*hasFormat*" and "*hasEncoding*" modifiers, the value at first is

[7] http://jena.sourceforge.net/

translated from the source context into an internal representation and then into the target context. A datetime value, for example, is firstly transformed into xsd:dateTime by fn:cvt_format_dateTime before being transformed again in order to comply with the target context. Notice that the number of conversion functions is not generally proportional to the number of sources and receivers in the system. The COIN framework facilitates the maximum reuse of existing conversion functions, and thus the number of newly-introduced conversion functions would diminish rapidly with the addition of each source or receiver. A more in-depth analysis of scalability can be found in [18].

With the COIN framework, the users are not burdened with the diverse data semantics in sources, all of which are declared in the context representation components and can be automatically taken into consideration by the mediator. A receiver in any context is able to issue queries over any set of data sources in other contexts as if they were in the same context. For example, it is easy for users to modify their context definition. Continue the motivational example in Section 2. Here the receiver wants to use CNY instead of USD. After the modification is made, the query results will immediately be reported in CNY as follows.

airline1	**airline2**	**total**
us339	jp241	8172.32

Exchange rate: 100 USD = 651.56 CNY; 100 JPY = 7.93 CNY.

The sources are not required to make any change or commit to any criteria under the COIN framework; they only need to record data semantics declaratively. Adding or removing a data source is accomplished by adding or removing the corresponding context declarations, which does not require any changes to the mediator or query processor. Conversion functions are defined for each modifier between distinct modifier values, not between pair-wise sources. Thus only a small number of atomic or component conversions need to be defined, which are used by the mediator to compose necessary composite conversions in a dynamic way to reconcile all semantic differences involved in a query. In many practical cases, an atomic function can be parameterized for the modifier to convert from any given context to any other context. The COIN strategy requires significantly less conversions than traditional global or interchange standardization approaches [18], and therefore is easy to be maintained.

6. RELATED WORK

We can roughly distinguish between two types of data conflicts in data integration: schematic conflicts and semantic conflicts. The first has been well-documented in the database and Semantic Web literature. A language, called CQuery was described in [10] to represent the domain knowledge in the form of a vocabulary or ontology as semantic metadata, and use the ontology to overcome heterogeneity among different data sources. Gracia et al. proposed a technique to perform the integration of ontology terms in order to cluster the most similar ones in [12]. Networked Graphs was proposed as a means for describing RDF graphs that are partially derived from other graphs using a declarative view mechanism. The relationships between graphs are described in a declarative way using SPARQL queries in [13]. Some surveys of schematic integration can be found in [14], [15], and [17]. Our approach is different because we have chosen to focus on the semantics of data level as opposed to the conflicts at schematic level. To the best of our knowledge, very little existing work has addressed the semantic conflicts at data level among RDF data sources.

The context interchange strategy is mediator-based approach for achieving semantic interoperability among heterogeneous sources and receivers. As realizations of the strategy, COIN [4], [7] and a recent extension [9], [19], are working prototypes that implement the Context Interchange strategy. COIN uses FOL/Prolog as the representation and implementation language for the application ontology in the context mediation. Various sample applications have been implemented to illustrate its ability to solve semantic interoperability problems in areas such as financial services, weather information, and airfare aggregation and comparison. Our goal in this paper is to illustrate how to extend COIN strategy to solve context conflicts in the emerging linked data by SPARQL query rewriting.

It is worth noting some interesting work which is complementary to our approach. A system, called Sesame, as well as its index structure was presented for query processing and optimization in distributed RDF repositories in [11]. Glimm and Krötzsch extend the SPARQL query language by defining the semantics of queries under the entailment regimes of RDF, RDFS, and OWL [16]. An approach is proposed to discover data that might be relevant for answering a query during the query execution itself [5]. The discovery is driven by following RDF links between data sources based on URIs in the query and in partial results.

7. CONCLUSIONS

The "Web of linked data" can be understood as a single, globally distributed dataspace, and SPARQL queries can be executed over this huge dataspace. But semantic heterogeneity widely exists among the RDF data sources originating from different contexts, and severely hampers their integration and interoperability. This paper describes how to use the Context Interchange strategy to achieve semantic interoperability among heterogeneous RDF sources and receivers by rewriting the user query to a mediated query. The semantic conflicts can be automatically detected and reconciled by the context mediator using the context definitions associated with both the data sources and the data receivers. The mediated query, when executed, returns the answer collated and presented in the receiver context. The core components of COIN are represented by using the Semantic Web constructs such as RDF and OWL, which make them possible to be exchanged and processed by other applications.

The mediated query uses extension functions to transform values between different contexts, which might be likely to have limited interoperability. It might require transformation between datatype formats not supported by the core SPARQL specification, and the orderings of some application datatype also need to be defined. A promising alternative is to use nest query (i.e. to nest the results of a query within another query) to provide conversion. The current SPARQL core standard does not support nested query, but the standard is still evolving (see [8]). In the future, we plan to consider temporal semantic heterogeneities which refer to the situation where the semantics between data sources, even in the same data source, change over time. For example, the profit of a company might be represented in DEM before 1998 and in EUR since 1999. The approach partitions graph patterns of a given "naive" query into multiple GRAPH clauses, each of which uses a unique set of query variables. This solution need to add FILTER clauses that ensure the equality of values bound to newly introduced variables. These filters increase the complexity of queries and have a negative effect on performance. The focus of the paper has been on the query mediation, but an implementation with query optimization is also planned.

8. ACKNOWLEDGMENTS

The work was supported, in part, by a grant from the National Natural Science Foundation (No. 60903078).

9. REFERENCES

[1] Franklin, M.J., Halevy, A.Y., Maier, D. From databases to dataspaces: A new abstraction for information management. *SIGMOD Record*, 34, 4, (2005), 27-33.

[2] Prud'hommeaux, E., Seaborne, A. *SPARQL query language for RDF*. W3C Recommendation, 2008. Retrieved April 25, 2011, from http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[3] Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., et al. *A survey of current approaches of mapping of relational database to RDF*, 2009. Retrieved April 25, 2011, from http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SuveyReport.pdf

[4] Goh, G.H., Bressan, S., Madnick, S., Siegel, M. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, 17, 3, (1999), 270-293.

[5] Hartig, O., Bizer, C., Freytag, J.-C. Executing SPARQL queries over the Web of linked data. In *Proceedings of the International Conference on Semantic Web, (ISWC'09)*. 2009, 293-309.

[6] Zhu, H. *Effective information integration and reutilization solutions: to technological deficiency and legal uncertainty*. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2005.

[7] Zhu, H., Madnick, S. Scalable interoperability through the use of COIN lightweight ontology. In *Proceedings of the VLDB Workshop on Ontologies-Based Techniques for Databases and Information System, (ODBIS'06)*, 2006, 37-50.

[8] Kjernsmo, K., Passant, A. *SPARQL New Features and Rationale*. W3C Working Draft 2, 2009. Retrieved April 25, 2011, from http://www.w3.org/TR/sparql-features/

[9] Li, X., Madnick, S., Zhu, H., Fan, Y.S. An approach to composing Web services with context heterogeneity. In *Proceedings of the International Conference on Information System, (ICIS'09*), 2009, 695-702.

[10] Sattler, K.U., Geist, I., Schallehn, E. Concept-based querying in mediator systems. *The VLDB Journal*, 14, (2005), 97-111.

[11] Stuckenschmidt, H. Vdovjak, R. Houben, G.J., Broekstra, J., Amerfoort, A.B.V. Index structures and algorithms for querying distributed RDF repositories. In *Proceedings of the International Conference on World Wide Web, (WWW'04)*, 2004, 631-639.

[12] Gracia, J., d'Aquin, M., Mena, E. Large scale integration of senses for the Semantic Web. In *Proceedings of the International Conference on World Wide Web, (WWW'09)*, 2009, 611-620.

[13] Schenk, S., Staab, S. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the Web. In *Proceedings of the International Conference on World Wide Web, (WWW'08)*, 2008, 585-594.

[14] Noy, N.F. Semantic integration: a survey of ontology-based approaches. *SIGMOD Record*, 33, 4, (2004), 65-70.

[15] Shvaiko, P., Euzenat, J. A survey of schema-based matching approaches. *Journal of Data Semantics IV*, (2005), 146-171.

[16] Glimm, B., Krötzsch, M. SPARQL beyond subgraph matching. In *Proceedings of the International Conference on Semantic Web (ISWC'10)*, 2010.

[17] Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H. Schuster, G., Neumann, H., Hübner, S. Ontology-based integration of information – a survey of existing approaches. In *Proceedings of the International Joint Conferences on Artificial Intelligence Workshop, (IJCAI'01)*, 2001, 108-117.

[18] Gannon, T., Madnick, S., Moulton, A., Siegel, M., Sabbouh, M., Zhu, H. Framework for the analysis of the adaptability, extensibility, and scalability of semantic information integration and context mediation approach. In *Proceedings of the Hawaii International Conference on System Sciences, (HICSS'09)*, 2009, 1-11.

[19] Mihai, L., Madnick, S. Using Semantic Web tools for context interchange. In *Proceedings of the VLDB Workshop on Ontologies-Based Techniques for Databases and Information System, (ODBIS'07)*, 2007.

APPENDIX A. NAMESPACE

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX coin: <http://coin.mit.edu/ontology#>
PREFIX ctx: <http://coin.mit.edu/contexts#>
PREFIX fts: <http://www.example.org/flightschedule#>
PREFIX fn: <http://coin.mit.edu/functions#>
```

APPENDIX B. THE EXAMPLE SCENARIO

Context Pool:

```
# Default graph: http://coin.mit.edu/contexts
ctx:US_dollar  coin:hasScale  "1"^^xsd:integer  .
ctx:US_dollar  coin:hasCurrency  "USD"  .
ctx:JP_yen  coin:hasScale  "1000"^^xsd:integer  .
ctx:JP_yen  coin:hasCurrency  "JPY"  .
ctx:US_dateTime  coin:hasFormat  "US_dateTime"  .
ctx:XSD_dateTime  coin:hasFormat  "XSD_dateTime"  .
ctx:IATA_airportCode  coin:hasEncoding  "IATA_airportCode"  .
ctx:EN_city  coin:hasEncoding  "EN_city"  .
```

Context Mappings:

```
# Named graph: http://coin.mit.edu/sources/usairline (Source 1)
fts:depDateTime  coin:dateTime  ctx:US_dateTime  .
fts:arrDateTime  coin:dateTime  ctx:US_dateTime  .
fts:depCity  coin:city  ctx:IATA_airportCode  .
fts:arrCity  coin:city  ctx:IATA_airportCode  .
fts:price  coin:monetaryValue  ctx:US_dollar  .

# Named graph: http://coin.mit.edu/sources/japanairline (Source 2)
fts:depDateTime  coin:dateTime  ctx:XSD_dateTime  .
fts:arrDateTime  coin:dateTime  ctx:XSD_dateTime  .
fts:depCity  coin:city  ctx:EN_city  .
fts:arrCity  coin:city  ctx:EN_city  .
fts:price  coin:monetaryValue  ctx:JP_yen  .

# Named graph: http://coin.mit.edu/receivers/myContext (Receiver)
:flight  coin:dateTime  ctx:US_dateTime .
:flight  coin:city  ctx:EN_city .
:flight  coin:monetaryValue  ctx:US_dollar .

# Named graph: http://coin.mit.edu/bindings/flight (Context binding)
<http://usairline.com/flights>  coin:hasContext
    <http://coin.mit.edu/sources/usairline>  .
<http://japanairline.com/flights>  coin:hasContext
   <http://coin.mit.edu/sources/japanairline>  .
coin:receiver  coin:hasContext  <http://coin.mit.edu/receivers/myContext>  .
```