MIT Sloan School of Management

MIT Sloan School Working Paper 4755-09

Reconciling Semantic Heterogeneity in Web Services Composition

Xitong Li, Stuart Madnick, Hongwei Zhu, Yushun Fan

Reconciling Semantic Heterogeneity in Web Services Composition

Xitong Li
Stuart Madnick
Hongwei Zhu
Yushun Fan

Working Paper CISL# 2009-08

September 2009

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

RECONCILING SEMANTIC HETEROGENEITY IN WEB SERVICES COMPOSITION

Completed Research Paper

Xitong Li
MIT Sloan School of Management
Cambridge, MA 02142, USA
xitongli@mit.edu
Tsinghua University
Beijing 100084, P. R. China
lxt04@mails.tsinghua.edu.cn

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

Hongwei Zhu
Old Dominion University
Norfolk, VA 23529, USA
hzhu@odu.edu

Yushun Fan
Tsinghua University
Beijing 100084, P.R. China
fanyus@tsinghua.edu.cn

Abstract

Service Oriented Computing (SOC) is a popular computing paradigm for the development of distributed Web applications. Service composition, a key element of SOC, is severely hampered by various types of semantic heterogeneity among the services. In this paper, we address the various semantic differences from the context perspective and use a lightweight ontology to describe the concepts and their specializations. Atomic conversions between the contexts are implemented using XPath functions and external services. The correspondences between the syntactic service descriptions and the semantic concepts are established using a flexible, standard-compliant mechanism. Given the naive BPEL composition ignoring semantic differences, our reconciliation approach can automatically determine and reconcile the semantic differences. The mediated BPEL composition incorporates necessary conversions to convert the data exchanged between different services. Our solution has the desirable properties (e.g., adaptability, extensibility and scalability) and can significantly alleviate the reconciliation efforts for Web services composition.

Keywords: Web service, service composition, semantic heterogeneity, ontology, context

Introduction

Service Oriented Computing (SOC) has become an increasingly important computing paradigm to develop and integrate distributed enterprise IT applications (Papazoglou et al. 2007; Zhang et al. 2007). As a technology of choice of SOC, Web services, also simply called services, are accessible software components/applications that can be invoked via open-standard Internet protocols (Yu et al. 2008). Web services composition addresses the situation in which a business need cannot be accomplished by a single pre-existing service, whereas a composite service consisting of multiple component services working together could satisfy the need. While the interface of a single (component or composite) service is described in the Web Service Description Language (WSDL) (Christensen et al. 2001), the workflow logic of a composite service is usually defined in the Business Process Execution Language (BPEL) (Alves et al. 2007), a standard for specifying the process of messages exchanged between the services.

A successful service composition must ensure semantic interoperability so that data can be exchanged meaningfully among the involved services. Unfortunately, semantic interoperability is severely hampered by the pervasive heterogeneity among independently-developed services. For example, a gallon in the U.S. (the so-called U.S. gallon) is approximately 3785 ml, while the "same" gallon in the U.K. (the so-called Imperial gallon) is 4546 ml, almost a liter more. So when we learn that a particular car model has a fuel tank capacity of 15 gallons by querying a Web service (say from the U.K.), and learn about the gas mileage of 30 miles per gallon for the model by querying another Web service (say from the U.S.), we still need to know how to interpret the exchanged data (i.e., 15 gallons) between the two services to compute the distance the car can go with a full tank of gas. Thus, additional information is still needed to correctly utilize the exchanged data. The challenge of semantic heterogeneity grows when composing multiple services developed by independent providers that are distributed throughout the world and have disparate assumptions of data interpretation. The basic Web services standards (e.g., WSDL, BPEL) generally ignore data semantics, rendering semantic interoperability far from reality. Several initiatives, e.g., OWL-S (Martin et al. 2007), WSMF/WSMO (Lausen et al. 2005) and METEOR-S (Patil et al. 2004), have proposed languages and frameworks to explicitly add semantics into service descriptions. Despite the foundations provided by these efforts, effective methods still need to be developed for reconciling semantic heterogeneity in Web services composition.

In this paper, we present a solution to automatic determination and reconciliation of semantic heterogeneity in Web services composition, such as inconsistent data naming, representation, precision, unit and scaling. The solution is inspired by the Context Interchange (COIN) strategy for semantic interoperability among multiple data sources (Bressan et al. 2000; Goh et al. 1999) and the preliminary work of applying the strategy (Li et al. 2009a; b; Mrissa et al. 2007) to Web services composition. The solution involves the use of a lightweight ontology, known as a COIN lightweight ontology, which defines a common vocabulary capturing only generic concepts shared by the involved services. The COIN lightweight ontology also defines multiple contexts capturing different specializations of the generic concepts which are actually used by the various services. Atomic conversions reconciling certain aspects of the differences need to be provided. Further, the WSDL descriptions of the involved services need to be annotated to establish correspondences between the data elements of WSDL descriptions and the concepts of the ontology. In this paper, we assume the service composition is specified using BPEL – in fact, our solution can be applied with any other composition specification languages. We call the BPEL composition, which ignores semantic heterogeneity, the naive BPEL. With the above descriptions in place, the reconciliation approach can automatically determine semantic conflicts in the naive BPEL and incorporate appropriate conversions into the composition. The mediated BPEL composition, now without any semantic conflict, is produced as the output of the reconciliation approach.

The rest of the paper is organized as follows. In the second section, we provide a motivating example of service composition with semantic conflicts. The third and fourth sections introduce the COIN lightweight ontology and conversions which form the foundation of our solution. In the fifth section, we present a standard-compliant mechanism for semantic and context annotation so that syntactic WSDL descriptions are elevated to the semantic level. The sixth section describes the automatic approach to determine and reconcile semantic conflicts within service composition. The seventh section demonstrates a proof-of-concept tool and evaluates the solution. Then, we review and compare the related work. Finally, we conclude this paper and suggest directions of future work.

Examples of Semantic Conflicts

Lack of explicitly representing data semantics in Web services descriptions makes it difficult to use them to build service composition. For example, a request for Total Assets of i2 Technology (ticker symbol: ITWO) by invoking a Web service from a financial data provider Xignite (http://www.xignite.com/) returned data shown in Figure 1.

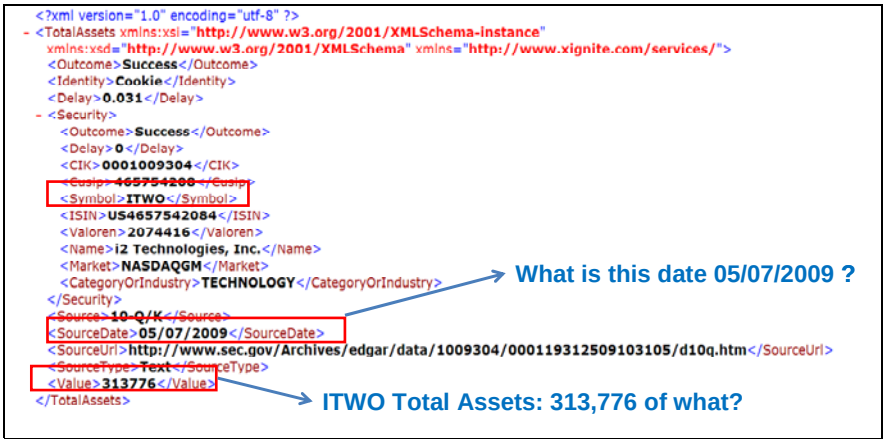

Figure 1. Total Assets Returned from a Real World Web Service

Should the date "05/07/2009" be interpreted as May 7th, 2009 or July 5th, 2009? Further, the same service returned 68853 as the total assets for Microsoft (ticker symbol: MSFT). Is it possible that ITWO has more than four times total assets than MSFT? Further investigation shows that the numeric value for ITWO is in thousands and that for MSFT is in millions, both in "$". But does the symbol "$" mean US dollar, Canadian dollar, or HK dollar?

Using one service is not easy, and composing multiple services is even more challenging. Consider a scenario that a U.K. developer wants to develop a new Web service, *OpeningPriceMarketCap* (denoted as *CS* for short for *C*omposite *S*ervice), to obtain the opening stock price and market capitalization of a U.S. company on its first trading day. *CS* is intended to be used by a U.K. analyst to study the U.S. stock market. The developer decides to implement the service by composing three existing services: *StockIPOWS*, *OpeningPriceWS* and *DailyMarketCap*, denoted as *S1*, *S2* and *S3* respectively. *S1* has the operation *getDateofIPO* that provides the IPO date of a company traded in the U.S. by using the company's ticker symbol. The operation *getOpeningPrice* of *S2* provides the opening stock price of a company on its first trading day. The operation *getDailyMarketCap* of *S3* provides the daily market capitalization of a company on a given date. The signatures of the four involved services (i.e., *CS*, *S1*, *S2* and *S3*) are summarized in Table 1. For simplicity, we do not show the verbose WSDL descriptions and assume the low-level messages of these services have compatible data types (e.g., string, double).

Table 1. Signatures of Involved Web Services in the Composition

Service	Operation	Input	Output
CS	*getOpeningPriceMarketCap*	tickerSymbol	openingPrice, openingMarketCap
S1	*getDateofIPO*	tickerSymbol	dateofQuote, tickerSymbol
S2	*getOpeningPrice*	tickerSymbol	openingPrice
S3	*getDailyMarketCap*	dateofQuote, tickerSymbol	dailyMarketCap

It appears that *CS* can be accomplished by a composition of *S1*, *S2* and *S3*. Specifically, the input *tickerSymbol* of *CS* needs to be transferred to *S1* and *S2*, respectively. The output *openingPrice* of *CS* is obtained from the output *openingPrice* of *S2*. The output *openingMarketCap* of *CS* can be achieved by feeding the output of *S1* to the input of *S3* and delivering the output of *S3* to *CS*. According to this plan, the developer defines the workflow logic of the composition using a typical BPEL tool, ActiveVOS BPEL Designer (http://www.activevos.com/). The BPEL composition process is graphically illustrated in Figure 2. Note that ActiveVOS BPEL Designer alerts no error in the composition process. However, since these four services are developed by independent providers, they have different assumptions of data interpretation, as summarized in Table 2. Usually, these assumptions are not explicitly represented in the WSDL descriptions. Further, existing BPEL tools cannot detect the conflicting assumptions and fail to alert these conflicts in the composition (see Figure 2), because the differences of data interpretation, also known as semantic conflicts, exist at the data instance level. If not reconciled, these semantic conflicts would result in severe errors and failures during the execution of the composition.

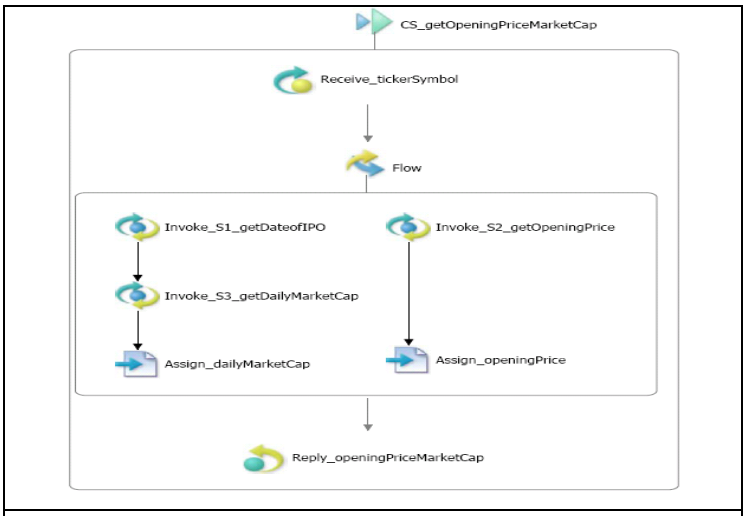

Figure 2. BPEL Composition with Semantic Conflicts

Table 2. Different Assumptions of Data Interpretation

Service	Date format	Currency	Scale factor
CS	NULL	GBP	1
S1	dd-mm-yyyy	NULL	NULL
S2	NULL	USD	1
S3	mm/dd/yyyy	USD	1000

A brute-force solution to address these semantic conflicts is to manually construct and insert ad-hoc conversions to transform the output of one service to the input of another service. Every time an involved service in the composition is changed or upgraded, the composition developers, however, have to manually specify new custom conversions and insert them into the composition process. As a result, the brute-force solution potentially makes the number of manually identified custom conversions very large and difficult to maintain over time. A survey (Seligman et al. 2002) shows that approximately 70% of the costs of integration projects are spent on identifying semantic differences and developing custom code to reconcile these differences. Our solution can significantly reduce the cost of semantic reconciliation for Web services composition.

COIN Lightweight Ontology

An alternative solution to semantic interoperability is to use a common ontology to support the transformation of the data exchanged among various services. An ontology is a collection of concepts and the relationships between these concepts. In practice there are various ontologies ranging from lightweight, rather informal, to heavyweight, more formal ontologies (Wache et al. 2001). To combine their strengths and avoid their weaknesses, we adopt a lightweight ontology, which requires a small set of generic concepts among the involved services and can structure their respective assumptions for interpreting the generic concepts by means of `contexts`.

Figure 3 presents a graphical representation of the COIN lightweight ontology for the composition example. Concepts are depicted by round rectangles and *basic* is the special concept from which all other concepts inherit. Like traditional ontologies, the COIN lightweight ontology has two classic relationships: `is_a` and `attribute`. For instance, concept *openingPrice* is a type of *stockMoneyValue*. An attribute is a binary relationship between a pair of concepts. For example, attribute *dateOf* indicates that each instance of concept *stockMoneyValue* reflects the money value of a stock on a certain date. In practice, it is frequently straightforward to identify generic concepts among multiple independent services. For example, *S3* has the output data *dailyMarketCap* and *CS* has an output data *openingMarketCap*. Both of them correspond to a generic concept *marketCapital*. However, *S3* provides the data instances of *dailyMarketCap* using currency "USD" and scale factor "1000", while *CS* interprets and furnishes the data instances of *openingMarketCap* according to currency "GBP" and scale factor "1". To accommodate the

different data interpretations, the construct `modifier` is introduced to allow multiple variations (i.e., specializations) to be associated with different services. In other words, `modifier` is used to capture additional information that affects the data interpretations of the generic concepts. A generic concept can have multiple modifiers, each of which indicates an orthogonal dimension of the variations. Also, a modifier can be inherited by a sub-concept from its ancestor concepts.

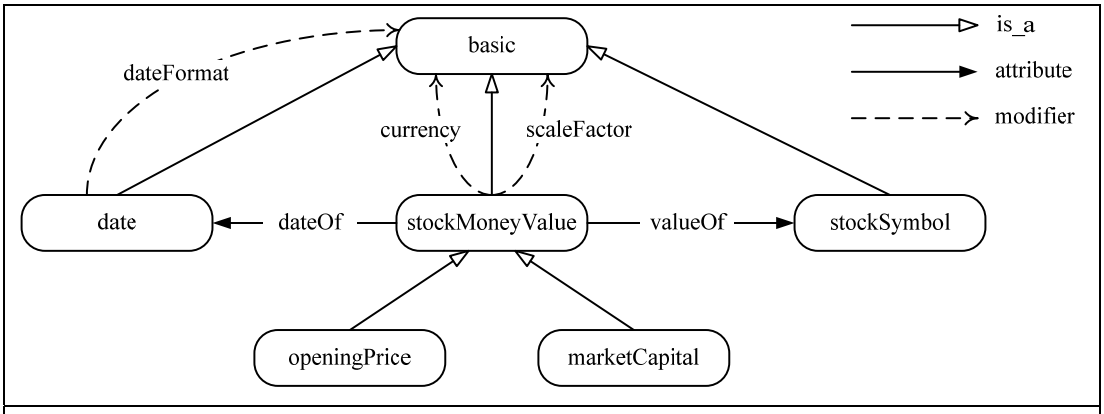

Figure 3. COIN Lightweight Ontology Shared by Involved Services of the Composition

Table 3. Context Definition of Involved Services in the Composition	
Service	**Context**
CS	*ctxt0* = {<*dateFormat*, NULL>, <*currency*, GBP>, <*scaleFactor*, 1>}
S1	*ctxt1* = {<*dateFormat*, dd-mm-yyyy>, <*currency*, NULL>, <*scaleFactor*, NULL>}
S2	*ctxt2* = {<*dateFormat*, NULL>, <*currency*, USD>, <*scaleFactor*, 1>}
S3	*ctxt3* = {<*dateFormat*, mm/dd/yyyy>, <*currency*, USD>, <*scaleFactor*, 1000>}

Modifiers are depicted by dashed arrows in Figure 3. For example, concept *stockMoneyValue* has two modifiers, *currency* and *scaleFactor*, which indicates that its data instances need to be interpreted according to two dimensions: money currency and scale factor, respectively. Also, concept *date* has modifier *dateFormat* that indicates its data instances can be interpreted by different date formats. The actual interpretation of a generic concept depends on modifier values. For instance, *CS* interprets concept *openingMarketCap* using currency "GBP". Thus, the value of modifier *currency* is "GBP" in case of *CS*. According to Table 2, the modifier value of *currency* is "USD" in case of *S2* and *S3*. That means that different services may need to assign different values to the modifiers. In our work, the different value assignments to a collection of modifiers are referred to as different `contexts`, and in a certain context each modifier is assigned by a specific modifier value. Specifically, a `context` is conceptually a set of assignments of all the modifiers of the COIN ontology and can be described by a set of <modifier, value> pairs. Further, each service involved in the composition may be associated with a context which corresponds to its assumption of data interpretation. For example, the different assumptions in Table 2 can be described using four contexts associated with the four services involved in the composition, as shown in Table 3. As a result, semantic differences among these services can be treated as context differences.

Conversions between Different Contexts

Context differences, once detected, need to be reconciled using conversions for converting the exchanged data from the source value *vs* to the target value *vt*. In our work, a conversion is defined for each modifier between two different modifier values. Below is a general representation of the conversions, where *C* is the generic concept having a modifier *m*, *mvs* and *mvt* are two different values of *m* in the source context *ctxt_s* and the target context *ctxt_t*, respectively. In fact, *mvs*, *mvt* can be derived by querying the context definition according to *ctxt_s*, *ctxt_t* (see Table 3).

$$cvt(C, m, ctxt\_s, ctxt\_t, mvs, mvt, vs, vt)$$

The conversions defined with modifiers are called atomic conversions. Since there exist three modifiers in the exemplified COIN ontology (see Figure 3 and Table 3), three atomic conversions need to be defined, i.e., $cvt_{\text{dateFormat}}$, cvt_{currency} and $cvt_{\text{scaleFactor}}$.

Our solution is agnostic about the actual implementation of the atomic conversions. In practice, depending on its complexity, an atomic conversion can be implemented using an XPath function[1] or an external (e.g., third-party) service. If possible, XPath functions are recommended due to the consideration of execution efficiency. For example, the atomic conversion $cvt_{\text{dateFormat}}$ for converting the date format from "dd-mm-yyyy" to "mm/dd/yyyy" can be implemented using the following XPath function:

```
cvt_dateFormat: Vt = concat(substring-before(substring-after(Vs,"-"),"-"),"/",
    substring-before(Vs,"-"),"/",substring-after(substring-after(Vs,"-"),"-"))
```

Also, the atomic conversion $cvt_{\text{scaleFactor}}$, which converts a number value from the scale factor *mvs* to *mvt*, can be implemented using the following XPath function[2]:

```
cvt_scaleFactor: Vt = Vs * mvs div mvt
```

In complex cases, the conversions have to be implemented by invoking external (e.g., third-party) services, such as by using Web wrapper services (Madnick et al. 2000). For example, it is needed to invoke an external currency exchange service *CurrencyExchange* (denoted as *S4* for short) which consumes the source and target currencies *mvs*, *mvt* and a money value *vs* and converts to another money value *vt*. Thus, *S4* can be used to implement the atomic conversion cvt_{currency}. In general, the WSDL template, which developers can use to discover and document an appropriate external service for the conversions, is presented in Figure 4. In our work we adopt WSDL 1.1 instead of WSDL 2.0, because current BPEL specification (Alves et al. 2007) only supports WSDL 1.1.

```
<wsdl:portType name="cvtPT">
   <wsdl:operation name="cvtOP">
      <wsdl:input message="msgType_s" name="msgName_s"/>
      <wsdl:output message="msgType_t" name="msgName_t"/>
   </wsdl:operation>
</wsdl:portType>
```

Figure 4. WSDL Template of External Web Services

It is worth noting that $cvt_{\text{scaleFactor}}$ and cvt_{currency} are defined as parameterized conversions: the source and target modifier values *mvs*, *mvt* are used as parameters of the conversions. A parameterized conversion can be applied to handle any pair of different modifier values *mvs* and *mvt* (i.e., a dimension of the context differences), not only a specific one. For example, cvt_{currency} can be used to convert money value between any pair of currencies. Using parameterized conversions can largely reduce the number of predefined atomic conversions and significantly enhance the scalability of our reconciliation solution.

In addition, atomic conversions can be used to construct composite conversions. In the motivating composition example, the market capitalization value in context *ctxt3* from *S3* is transferred to the value in context *ctxt0* for *CS*. As shown in Table 3, the differences of both currency and scale factor need to be reconciled. In brute-force solutions, the conversion for this reconciliation is frequently specified in a straightforward but manual way. According to the COIN ontology, two modifiers (i.e., *scaleFactor* and *currency*) are considered and our solution can automatically construct the composite conversion by applying the two atomic conversions $cvt_{\text{scaleFactor}}$ and cvt_{currency} successively. The algorithm of conversion composition can be found in (Zhu and Madnick 2006). Compared to the brute-force integration approaches, the mechanism of constructing composite conversions consisting of parameterized atomic conversions significantly enhances the adaptability and scalability of our reconciliation solution (Gannon et al. 2009; Zhu and Madnick 2006).

Semantic and Context Annotation

Web services are described using the WSDL specification at a syntactic level, rather than a semantic level. To facilitate semantic interoperability, semantic annotation is widely used to establish correspondences between the

[1] The BPEL specification and most BPEL engines (e.g., ActiveBPEL) support XPath 1.0.
[2] Note that this is a general purpose conversion function that works for any values of *mvs* and *mvt*.

data elements of WSDL descriptions and the concepts of an ontological model (Patil et al. 2004; Sivashanmugam et al. 2003). The annotations can be done using the W3C standard, Semantic Annotation for WSDL and XML Schema (SAWSDL) (Farrell and Lausen 2007). SAWSDL allows any language for expressing an ontological model and enables developers to annotate the syntactic WSDL descriptions with pointers to the concepts (identified via URIs) of the ontological model (Kopecký et al. 2007; Verma and Sheth 2007). We use SAWSDL to annotate the WSDL descriptions so that the syntactic descriptions are lifted to a semantic level.

SAWSDL provides an extension attribute `modelReference` for specifying the correspondence between WSDL components (e.g., data/element types, input and output messages) and the concepts of an ontology. Herein, we introduce two alternative ways of using the `modelReference` attribute to annotate the WSDL descriptions to the COIN lightweight ontology: (1) Global context annotation: we allow the <wsdl:definitions> element of the WSDL specification to have the `modelReference` attribute and use its value to indicate that all data elements of a WSDL description subscribe to a certain context identified via the URI value; (2) Local context annotation: for any data element, in addition to the URI value indicating the corresponding ontological concept, we allow the `modelReference` attribute to have an additional URI value to indicate the context of the data element.

Global context annotation affects the entire WSDL description and allows the developers to succinctly declare the context for all elements of the WSDL description. Local context annotation provides a mechanism for certain elements to have their contexts different from the globally declared context. In case a small number of elements in a WSDL description have contexts different from that of the other elements, this `overriding` capability devised in our solution can be useful to simplify the annotation task.

```
<wsdl:definitions targetNamespace="http://openingPriceMarketCap.coin.mit" …
                  xmlns:stkCoin="http://coin.mit.edu/ontologies/stockOntology#"
                  xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
                  sawsdl:modelReference="stkCoin#ctxt3" >
  <wsdl:types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
            targetNamespace="http://openingPriceMarketCap.coin.mit">
    <element name="ticekerQuoteDate">
      <complexType>
        <sequence>
          <element name="tickerSymbol" type="xsd:string"
                   sawsdl:modelReference="stkCoin#stockSymbol" />
          <element name="dateofQuote" type="xsd:string"
                   sawsdl:modelReference="stkCoin#date stkCoin#ctxt3" />
        </sequence>
      </complexType>
    </element>
    <element name="dailyMarketCap" type="xsd:double"
             sawsdl:modelReference="stkCoin#marketCapital stkCoin#ctxt3" />
   </schema>
  </wsdl:types>
  <wsdl:message name="getDailyMarketCapResponse">
    <wsdl:part element="impl:dailyMarketCap" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="getDailyMarketCapRequest">
    <wsdl:part element="impl:ticekerQuoteDate" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="DailyMarketCap">
    <wsdl:operation name="getDailyMarketCap">
      <wsdl:input message="impl:getDailyMarketCapRequest" name="getDailyMarketCapRequest"/>
      <wsdl:output message="impl:getDailyMarketCapResponse" name="getDailyMarketCapResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figure 5. Annotated WSDL Description of *S3* Using Global and Local Context Annotations

Figure 5 shows the annotated WSDL description of *S3* in which the annotations are highlighted in bold. Each leaf data element of *S3* has the `modelReference` attribute to point to its corresponding concept in the COIN ontology. For example, the elements *tickerSymbol* and *dateofQuote* correspond to the concepts *stockSymbol* and

date, respectively. Since *S3* use context *ctxt3* (see Table 3), the `modelReference` attribute of the element <*wsdl:definitions*> has the value "*stkCoin#ctxt3*" which is the URI of context *ctxt3* defined in the COIN ontology. As shown in Figure 5, the `modelReference` attribute of a data element has one value, or two values separated by a whitespace[3]. In case of only one value, it is the URI of the concept to which the data element corresponds. In case of two values, the former value is the URI of the concept and the latter is the URI of the context in which the data element is interpreted. It is worth noting that both global and local context annotations comply with the SAWSDL standard. For illustration purposes, both the global and local context annotations are used in Figure 5, though the local annotation in Figure 5 is unnecessary - it does not override the global context for a different one.

If business needs were to change over time and we later needed to shift the date format of *S3* from "mm/dd/yyyy" to "dd-mm-yyyy", the only thing we need to do is to update the context of the *dateofQuote* element of *S3* to context *ctxt1* (see Table 3) by means of the local context annotation. Then, our solution would automatically determine and reconcile possible semantic differences resulting from the date format change. Thus, the global and local context annotations promote the flexibility of our solution to handle the evolving semantics of services.

Reconciliation Approach

In the domain of Web services composition, context conflicts probably occur when a piece of data from the source service with one context is transferred to and consumed by the target service with another context. Figure 6 shows the typical scenario where a context conflict occurs in the composition. As shown in Figure 6, there exists a data transfer where the data *data_s* from service *WS_s* is transferred and consumed as data *data_t* of service *WS_t*. Using context annotation, both *data_s* and *data_t* are annotated to concept *C* having a modifier *m*. Also, *WS_s* and *WS_t* are annotated with two contexts *ctxt_s*, *ctxt_t*, respectively. According to the context definition of the COIN ontology, *data_s* and *data_t* are interpreted differently by *WS_s* and *WS_t* if the modifier value of *m* in *ctxt_s* is *mvs* different from the value *mvt* of *m* in *ctxt_t*. As a result, a context conflict occurs within the data transfer.

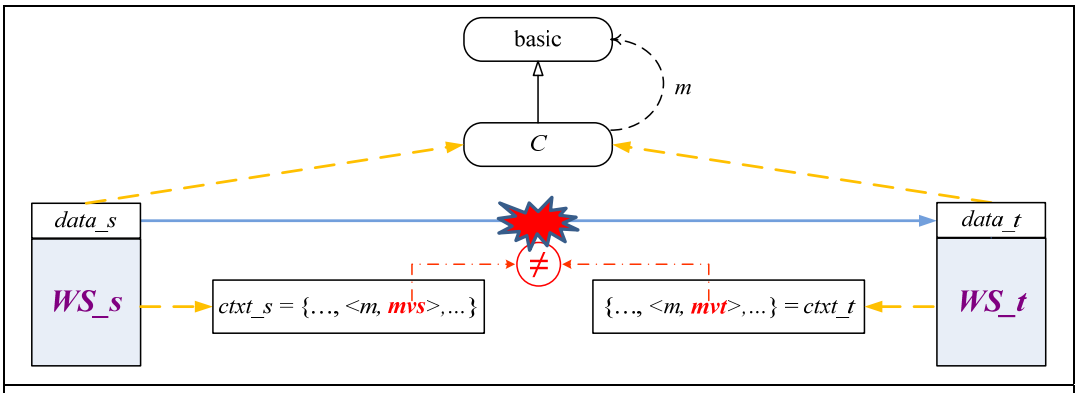

Figure 6. Scenario of Context Conflict in Web Services Composition

Herein, we introduce the approach to determine and reconcile context conflicts within Web services composition. The reconciliation approach consists of three procedures that can be automatically performed. In the following sections, we assume the needed context annotation is specified in the WSDL descriptions and the process/workflow logics of service composition are defined in BPEL. Note that our solution need not extend or annotate the BPEL specification. Also, the solution can be easily adapted for many other process modeling languages.

Identifying Data Transfers

Recall that the BPEL composition which defines the process of service composition ignoring context conflicts is called the naive BPEL. Since context conflicts occur within data transfers, it is needed to analyze the data flow of the naive BPEL and identify all the data transfers. Each data transfer can be represented using the following form, where *ws_s*/*ws_t* and *data_s*/*data_t* are the source/target service and data element involved in the data transfer, and *type* is the type of the data transfer, which can be either explicit or implicit.

[3] The SAWSDL specification allows the `modelReference` attribute having multiple values separated by whitespaces.

dataTrans(*type, data_s, ws_s, data_t, ws_t*)

Each explicit data transfer involves two variables and can be easily identified according to the <*assign*> activity which is used to copy the data of the source variable to the target variable. As shown in Figure 2, there are two <*assign*> activities within the composition example: one is to transfer the data *dailyMarketCap*, and the other is to transfer the data *openingPrice*. Thus, two explicit data transfers are identified.

Each implicit data transfer involves one variable and can be identified in case the source and target interaction activities manipulating the variable are different. The BPEL specification provides four types of interaction activities, that is, <*receive*>, <*reply*>, <*invoke*>, and <*onMessage*> contained in <*pick*>. For a certain variable *var*, its source interaction activity may be <*receive*>, <*onMessage*>, or <*invoke*> in case *var* is the output variable. Its target interaction may be <*reply*>, or <*invoke*> in case *var* is the input variable. By examining each variable in the composition, all implicit data transfers in the BPEL composition can be identified.

```
Input: BPEL process proc;
Output: The set of explicit data transfers EDT = {edt},
        The set of implicit data transfers IDT = {idt};

1.  Set EDT = ∅, IDT = ∅
2.  For each <assign> activity asn in proc
3.     var_s ← getSourceVariable(asn)
4.     var_t ← getTargetVariable(asn)
5.     act_s ← getSourceInteractionActivity(proc, asn)
6.     act_t ← getTargetInteractionActivity(proc, asn)
7.     edt ← getDataTransfer(var_s, var_t, act_s, act_t)
8.     EDT ← EDT ∪ {edt}
9.  For each variable var in proc
10.    L_var ← getInteractionActivitySeries(proc, var)
11.       For each source activity act_s1 in L_var
12.          act_s2 ← getNextSourceActivity(L_var, act_s1)
13.          T_var ← getTargetActivitySeries(L_var, act_s1, act_s2)
14.          For each target activity act_t in T_var
15.             idt ← getDataTransfer(var, act_s1, act_t)
16.             IDT ← IDT ∪ {idt}
17. Return EDT, IDT
```

Figure 7. Algorithm 1: Algorithm for Identifying Explicit and Implicit Data Transfers

Figure 7 gives the algorithm (Algorithm 1) for identifying explicit and implicit data transfers. Using Algorithm 1, three implicit and two explicit data transfers are identified within the composition example, as shown in Table 4. Instead of explicitly using the <*assign*> activity, the output of invoking *S1* is directly transferred and consumed as the input of invoking *S3* through variable *tickerQuoteDate*. An implicit data transfer is thus identified, where the source and target interaction activities are the invocation of *S1*, *S3*, respectively. As shown in Figure 2, the composition example involves <*receive*>, <*reply*> and <*invoke*>, except <*onMessage*>.

Table 4. Data Transfers of the Composition Example	
dt1	*dataTrans* (*implicit*, *tickerSymbol*, *CS*, *tickerSymbol*, *S1*)
dt2	*dataTrans* (*implicit*, *tickerSymbol*, *CS*, *tickerSymbol*, *S2*)
dt3	*dataTrans* (*implicit*, *tickerQuoteDate*, *S1*, *tickerQuoteDate*, *S3*)
dt4	*dataTrans* (*explicit*, *openingPrice*, *S2*, *openingPrice*, *CS*)
dt5	*dataTrans* (*explicit*, *dailyMarketCap*, *S3*, *openingMarketCap*, *CS*)

Determining Context Conflicts

When a data transfer is identified, the annotated WSDL descriptions of its source and target services (denoted as *ws_s* and *ws_t*, respectively) can be derived through <*partnerLinkType*> of the BPEL composition. According to the context annotation, the concept *C* corresponding to the transferred data is obtained. Also, if the source data *data_s* and the target data *data_t* are annotated with contexts, their contexts are denoted as *ctxt_s*, *ctxt_t*, respectively. In order to determine possible context conflicts, all modifiers of concept *C* need to be examined. In case a certain

modifier *m* has different values *mvs*, *mvt* in *ctxt_s* and *ctxt_t*, respectively, a context conflict is thus determined. The scenario of determining context conflicts is illustrated in Figure 6. For example, *dt3* (see Table 4) is an implicit data transfer involving variable *tickerQuoteDate* which contains two data elements *dateofQuote* and *tickerSymbol*. In the WSDL descriptions of *S1* and *S3*, *dateofQuote* is annotated to concept *date* of the COIN ontology. The concept *date* has a modifier *dateFormat* with different values in the contexts of *S1* and *S3*: "dd-mm-yyyy" for *S1* and "mm/dd/yyyy" for *S3* (see Table 3). As a result, a context conflict occurs when *dateofQuote* is transferred from *S1* to *S3* through the data transfer *dt3*.

Each context conflict can be represented using the following form, where *dt* is the data transfer in which the context conflict occurs. $[(m_i, mvs_i, mvt_i)]_{i=\{1,\ldots,n\}}$ depicts the array of 1 to n modifiers with different values in *ctxt_s* and *ctxt_t*.

$$ctxtConflict(dt, C, ctxt\_s, ctxt\_t, [(m_i, mvs_i, mvt_i)]_{i=\{1,\ldots,n\}})$$

Figure 8 gives the algorithm (Algorithm 2) to automate the determination procedure. Using Algorithm 2, three context conflicts within the naive BPEL composition are determined as shown in Table 5.

```
Input: BPEL process proc, the set of data transfers DT = {dt},
       The set of annotated WSDL description WS = {ws}, COIN ontology onto;
Output: The set of context conflicts CC = {cc};

1. Set CC = ∅
2. For each data transfer dt in DT
3.    ws_s ← getSourceService(dt, proc, WS)
4.    ws_t ← getTargetService(dt, proc, WS)
5.    data_s ← getSourceDataElement(ws_s, dt)
6.    data_t ← getTargetDataElement(ws_t, dt)
7.    c ← getConcept(ws_s, data_s)
8.    ctxt_s ← getContext(ws_s, data_s)
9.    ctxt_t ← getContext(ws_t, data_t)
10.   For each modifier m of c in onto
11.      mvs ← getModifierValue(c, m, ctxt_s)
12.      mvt ← getModifierValue(c, m, ctxt_t)
13.         If mvs ≠ mvt
14.         Then cc ← getContextConflict(C, m, ctxt_s, ctxt_t, mvs, mvt)
15.              CC ← CC ∪ {cc}
16. Return DT
```

Figure 8. Algorithm 2: Algorithm for Determining Context Conflicts

Table 5. Context Conflicts of the Composition Example

cc1	*ctxtConflict* (*dt3*, *date*, *ctxt1*, *ctxt3*, [(*dateFormat*, "dd-mm-yyyy", "mm/dd/yyyy")])
cc2	*ctxtConflict* (*dt4*, *openingPrice*, *ctxt2*, *ctxt0*, [(*currency*, "USD", "GBP")])
cc3	*ctxtConflict* (*dt5*, *dailyMarketCap*, *ctxt3*, *ctxt0*, [(scaleFactor, "1000", "1"); (currency, "USD", "GBP")])

Incorporating Conversions

Once a context conflict is determined within a data transfer, it is needed to identify an appropriate conversion to reconcile the conflict. The appropriate conversion is either a predefined atomic conversion or a composite conversion consisting of several atomic conversions. For reconciliation, the identified conversion is incorporated into the data transfer to convert the exchanged data in the source context to the target context.

In case the determined context conflict occurs in an implicit data transfer, the data transfer needs to be made explicit in order to incorporate the conversion. Suppose *var* is the variable involved in the implicit data transfer. To make the data transfer explicit, it is needed to create a new variable named *var_t* which has the same element type with *var*, and to insert a <*assign*> activity into the data transfer for copying *var* to *var_t*. As discussed above, data transfer *dt3* is an implicit data transfer where a context conflict of date format occurs. To make *dt3* explicit, a new variable

tickerQuoteDate_t is declared using the same element type of variable *tickerQuoteDate*. Since *tickerQuoteDate* has two data elements *dateofQuote* and *tickerSymbol*, the <*assign*> activity inserted into *dt3* has two <*copy*> elements for copying *dateofQuote* and *tickerSymbol* of *tickerQuoteDate* to that of *tickerQuoteDate_t*. Then, the input variable of the invocation of *S3* is changed from variable *tickerQuoteDate* to variable *tickerQuoteDate_t*. After this step, all data transfers with context conflicts are explicit.

In case a context conflict involves only one modifier, it can be reconciled using a predefined atomic conversion that is identified according to its information. For example, the context conflict *cc1*, as shown in Table 5, involves modifier *dateFormat* of concept *date*. It is thus easy to identify the atomic conversion $cvt_{\text{dateFormat}}$ that can reconcile *cc1*. The conversion $cvt_{\text{dateFormat}}$ is applied through substituting the input *vs* of the XPath function as data element *dateofQuote*. Also, the context conflict *cc2* involves modifier *currency* of concept *openingPrice*, which can be reconciled using the atomic conversion cvt_{currency}. As discussed before, cvt_{currency} is implemented by the external currency exchange service *S4*, rather than using XPath function. Thus, a <*invoke*> activity is inserted in the data transfer *dt4* of *cc2* in order to convert *openingPrice* in "USD" from *S2* to the equivalent price in "GBP", an output data of *CS*. Necessary <assign> activities are also inserted to explicitly transfer the exchanged data.

```
Input: BPEL process proc, the set of annotated WSDL description WS = {ws},
       the set of context conflicts CC = {cc},
       the set of predefined atomic conversions CVT = {cvt};
Output: Mediated BPEL process mediatedProc;

1. mediatedProc = proc
2. For each context conflict cc in CC
3.    dt ← getDataTransfer(cc)
4.    If isImplicit(dt) == 'TRUE'
5.      Then var ← getVariable(dt), var_t ← declareNewVariable(var),
6.           insertAssign(mediatedProc, dt, var, var_t)
7.    AMV = [(m_i, mvs_i, mvt_i)] ← getArrayOfModifierValues(cc)
8.    If |AMV| == "1"
9.       Then cvt ← getAtomicConversion(cc, m, CVT)
10.           insertConversion(mediatedProc, cvt)
11.   Else
12.      For each (m_i, mvs_i, mvt_i) in AMV
13.         cvt_i ← getAtomicConversion(cc, m_i, CVT)
14.         insertConversion(mediatedProc, cvt_i)
15. Return mediatedProc
```

Figure 9. Algorithm 3: Algorithm for Incorporating Conversions

When a certain context conflict involves two or more modifiers, no predefined atomic conversion can reconcile the context conflict, as each atomic conversion is defined with only one modifier. In these complex cases, the context conflict can still be reconciled using the composition of multiple atomic conversions, each of which is defined with one of the modifiers involved in the context conflict. For example, the context conflict *cc3* involves two modifiers *scaleFactor* and *currency* of concept *marketCapital*. Among the predefined atomic conversions, modifier *scaleFactor* and *currency* correspond to $cvt_{\text{scaleFactor}}$, cvt_{currency}, respectively. Therefore, *cc3* can be reconciled using the composition of the two atomic conversions, successively applying $cvt_{\text{scaleFactor}}$ and cvt_{currency}. Specifically, the output data *dailyMarketCap* from *S3* is first converted by $cvt_{\text{scaleFactor}}$ from the scale factor "1000" to "1", and then converted by cvt_{currency} from the currency "USD" to the equivalent amount in "GBP". After the two-step composite conversion consisting of $cvt_{\text{scaleFactor}}$ and cvt_{currency}, the exchanged data is converted and transferred to the output data *openingMarketCap* of *CS*. Figure 9 gives the algorithm (Algorithm 3) to automate the procedure of incorporating conversions to reconcile the determined context conflicts.

Implementation and Evaluation

A proof-of-concept prototype, `Context Mediation Tool` (CMT), has been implemented to validate the reconciliation solution. The COIN lightweight ontology with structured contexts is defined using the COIN Model Application Editor[4] which is a Web-based tool for creating and editing COIN ontologies and contexts in RDF/OWL. Atomic conversions between the contexts are defined in a specification file. The WSDL descriptions of the

[4] http://interchange.mit.edu/appEditor/TextInterface.aspx?location=MIT

composite and component services (i.e., *CS* and *S1* ~ *S3*) are annotated with the `modelReference` attribute of the SAWSDL standard to the COIN lightweight ontology. To facilitate the annotation task, we have developed the context annotation tool *Radiant4Context* which is an extension to Radiant[5], an open-source Eclipse plug-in for semantic annotation. We assume that the U.K. developer, being unaware of semantic differences among the services, created the naive BPEL process of the motivating composition example.

To perform the reconciliation approach, we use CMT to create a mediation project and import all these documents. The reasoning engine implemented within CMT first uses Algorithm 1 (see Figure 7) to identify the three implicit and two explicit data transfers within the naive BPEL composition example. Then, CMT uses Algorithm 2 to determine the three context conflicts. Finally, three predefined[6] atomic conversions $cvt_{\text{dateFormat}}$, $cvt_{\text{scaleFactor}}$ and cvt_{currency} are required and incorporated into the corresponding data transfers to reconcile the three context conflicts.

CMT has three working areas for the mediation work, as shown in Figure 10. The first working area requires the user to import the involved documents of the composition into the mediation project. To monitor the results of different mediation steps, the second working area of CMT, `Mediation Stage`, allows the user to choose one of the four consecutive stages, including *Naive BPEL Process*, *Data Transfers*, *Context Conflicts*, and *Mediated BPEL Process*. These stages reveal the intermediate and final results that the approach produces while handling semantic heterogeneity among services involved in the composition. For example, Figure 10 shows the stage *Context Conflicts* where the three context conflicts within the motivating composition example and corresponding atomic conversions required for the reconciliation are identified. At the *stage Mediated BPEL Process*, CMT produces the mediated BPEL composition with incorporated conversions.

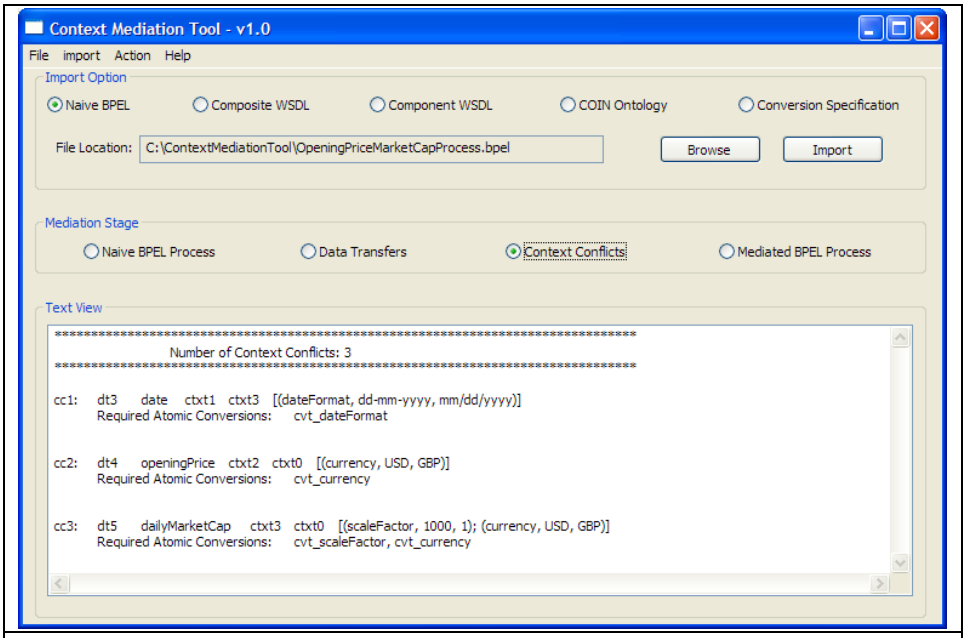

Figure 10. Snapshot of CMT at Stage *Context Conflicts*

For validation purposes, we imported the mediated BPEL composition into the typical BPEL tool ActiveVOS BPEL Designer. We provided several sample data values for the input of *CS* and the output of the invoked services (i.e., *S1* ~ *S3* and *S4*). We utilized the simulation feature of ActiveVOS BPEL Designer to simulate the execution of the mediated BPEL composition, as shown in Figure 11. The execution results indicated that: a) the mediated BPEL process normally completed; b) all the three context conflicts were successfully reconciled, that is, appropriate conversions were properly performed to convert date formats, scale factors and currencies; and c) *CS* produced the expected output: *openingPrice* and *openingMarketCap*.

[5] http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1
[6] We recommend that libraries of such atom conversions be established that can be reused for future compositions.

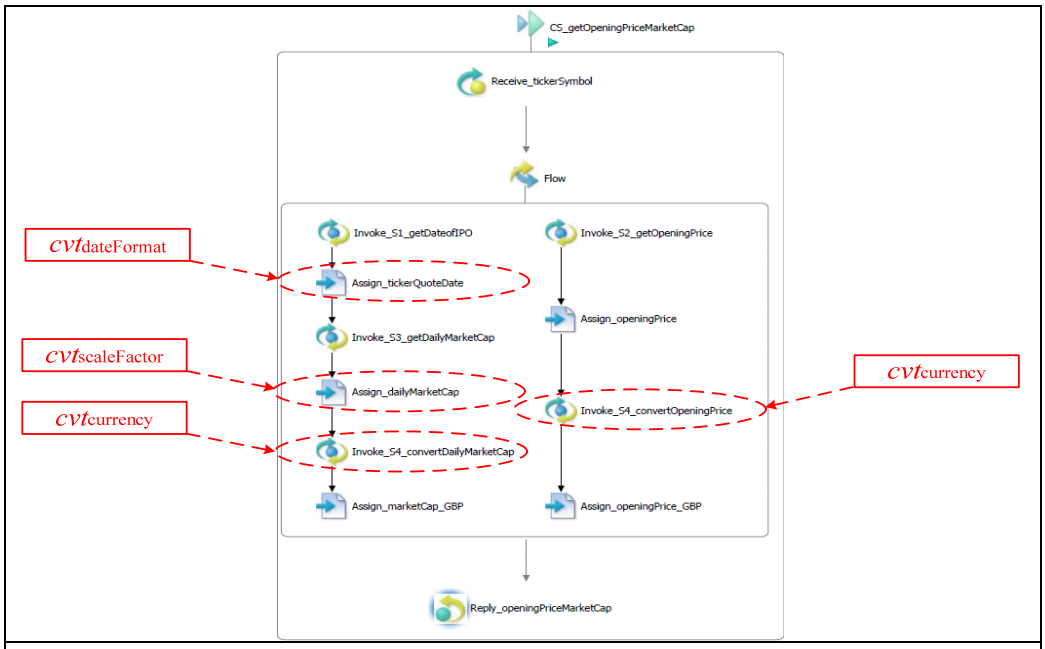

Figure 11. Mediated BPEL Composition with Incorporated Conversions

Due to the limited space, we have shown only one composition example through this paper. In fact, our reconciliation solution can handle more general and complex composition situations, such as situations that involve inconsistent data representation, precision, unit and scaling. Further, our solution can deal with all the four types of interaction activities supported by the BPEL specification (e.g., <*receive*>, <*reply*>, <*invoke*> and <*onMessage*>) and complex workflow constructs for the BPEL composition, including sequence, parallel, choice and iteration.

To evaluate the benefits of our reconciliation solution, we take the evaluation framework that utilizes an analytical approach to indirectly measure human efforts with respect to the number of conversions to be manually specified and maintained over time (Gannon et al. 2009). The measurements of the framework are as follows:

- *Adaptability*: number of conversions to be updated when data semantics of the involved services change.
- *Extensibility*: number of conversions to be added (or removed) when a service is added (or removed).
- *Scalability*: number of conversions needed for the reconciliation among the involved services.

Let us assume that the CS developer later wanted to serve diverse users that require any combination of 10 different currencies and 4 scale factors (i.e., 1, 1K, 1M, 1B). To convert the output *dailyMarketCap* of *S3* to the output *openingMarketCap* of *CS*, it would be most likely for composition developers to manually specify 39 (=10×4 - 1) custom conversions if they used a brute-force approach. An even worse case would arise if currencies and scale factors of *CS*, *S2* and *S3* changed over time independently. Actually, such situations frequently happen in reality, because the implementations of Web services always evolve in the fast-changing global business environment. Comparatively, it is only needed to define 2 parameterized atomic conversions (i.e., $cvt_{\text{scaleFactor}}$ and cvt_{currency}) by using our reconciliation solution, regardless of the changing currencies and scale factors. Therefore, our solution holds the adaptability and extensibility, if the involved services were to be updated or services were to be added (or removed) in the composition. Also, our solution has the scalability, as it significantly reduces the number of conversions to be predefined. Due to limited spaces, the detailed evaluation about the adaptability, extensibility and scalability of our solution will be presented in a future paper.

Related Work and Discussion

The basic Web services standards (e.g., WSDL, BPEL) generally ignore data semantics, rendering semantic composition and interoperability far from reality. It has become an active research area for applying Semantic Web technologies to Web services, referred to as Semantic Web Services (SWSs) (Burstein et al. 2005; McIlraith et al. 2001; Sycara et al. 2003). OWL-S (Martin et al. 2007), WSMF/WSMO (Fensel and Bussler 2002; Lausen et al. 2005) and METEOR-S (Patil et al. 2004; Sivashanmugam et al. 2003) are three major initiatives that have developed

languages and frameworks to explicitly add semantics into the Web services descriptions. Despite the ontological foundations provided by these efforts, it is still necessary to develop effective approaches to semantic composition and interoperability towards the vision of SWSs.

For Web services composition, Colombo framework (Berardi et al. 2005) combines the key elements of SWSs and can automatically synthesize composite Web services. In (Rao et al. 2006), Web services are externally described in DAML-S and the process model of the composite service is generated using a Linear Logic theorem prover. The INFRAWEBS approach (Agre and Marinova 2007) can find an appropriate service composition which is guided by the algorithm for run-time decomposition of the user goal into sub-goals. Most of the efforts in this thread focus on automatically constructing the workflow logic of service composition by means of semantics/ontologies, but ignore semantic heterogeneity among the services that could severely hamper their composition.

In the literature only a few approaches have been developed to handle semantic heterogeneity in Web services composition. The approach discussed in (Nagarajan et al. 2006; Nagarajan et al. 2007) deals with schematic heterogeneity of the messages exchanged between a pair of services, rather than a composition consisting of multiple services. The approach requires each service to be annotated and mapped to an ontology that serves as the global schema. It is significantly more time-consuming to construct and maintain this type of global schema than the lightweight ontology used in our approach which only needs a small set of generic concepts. More importantly, the mappings are created manually to include necessary conversion in their approach. In contrast, the actually mappings in our approach are automatically composed using a small number of atomic conversions.

The data-level heterogeneity between Web services is investigated in (Spencer and Liu 2004) and data transformation rules are constructed to convert the data of the exchanged messages from one service to the other. But the approach focuses on dealing with a pair of interoperating services, rather than a composition of multiple services. The work in (Gagne et al. 2006; Sabbouh et al. 2008) proposes a set of mapping relations to establish the direct correspondence between the messages of two services. The approach is also restricted to simple composition scenarios in which only two services are integrated. Furthermore, those approaches that construct conversions (e.g., mappings, rules) between a pair of services will result in the gradually increasing number of conversions over time.

To the best of our knowledge, the work in (Mrissa et al. 2006a; b; Mrissa et al. 2007), which also draws on the original COIN strategy, is most related to this paper. However, our solution is significantly distinct from their work in multiple aspects. (1) Their work ignores considering the composite service whose context may be different from any component service, while our solution can address both composite and component services. (2) They embed context definition in WSDL descriptions using a non-standard extension. As a result, their approach suffers from the proliferation of redundant context descriptions when multiple services share the same context. In contrast, we avoid such problem by specifying modifier values in the ontology definition separate from WSDL descriptions and use the flexible, standard-compliant mechanism for annotating WSDL descriptions using SAWSDL. (3) Only external services are considered in their work to reconcile context differences, while both XPath functions and external services are proposed in our reconciliation solution. In certain cases, it is applicable and more efficient to use XPath functions as conversions, such as date formats in different styles and numbers in different scale factors. (4) Only context conflicts between the <*invoke*> activities in the BPEL composition are considered in their work, while context conflicts between all interaction activities (e.g., <*receive*>, <*reply*>, <*invoke*> and <*onMessage*>) can be handled using our solution. (5) Their approach requires a priori specification of the external services to be invoked to reconcile context conflicts, thus they do not have the automatic conflict detection and conversion composition capability, and they miss the opportunity to reuse predefined conversion. In our work we define a parameterized atomic conversion for each modifier and use reasoning algorithms to automatically generate composite conversions consisting of atomic conversions to handle complex context differences. Thus, the number of predefined conversions is largely reduced. As discussed in (Gannon et al. 2009), the mechanism of the conversion composition significantly enhances the adaptability, extensibility, and scalability of the COIN-based solution.

In addition to the literature on Web services, it is worth noting an interesting work (Sun et al. 2006) from the domain of process/workflow management. Their work develops data-flow specification for detecting data-flow anomalies within a process/workflow, including missing data, redundant data and potential data conflicts. However, their work provides no automatic approach that can be used to produce the data-flow specification. Also, semantic heterogeneity of the data exchanged is not considered. It may be able to adapt Algorithm 1 (see Figure 7) to automatically construct data-flow specification so that potential data-flow anomalies can be also addressed.

In summary, this paper makes three important contributions:

First, this paper provides a solution to automatic determination and reconciliation of semantic heterogeneity in Web services composition. The solution uses the COIN lightweight ontology which, once constructed, can be reused in further composition/integration situations. More importantly, we use reasoning algorithms for the automatic composition of atomic conversions to handle complex context differences and reduce the number of conversions to be manually predefined. Also, certain atomic conversions can be parameterized to further reduce the number of predefined conversions. Thus, our solution can significantly alleviate the reconciliation efforts and accelerate the development of Web services composition.

Second, this paper describes a standard-compliant mechanism for annotating WSDL descriptions using SAWSDL. This mechanism allows the annotation task to be performed using any existing SAWSDL-aware tools, e.g., Radiant[7]. Also, two alternative ways of context annotation (i.e., the global and local methods) are provided to alleviate the complexity of handling the evolving data semantics of Web services. Thus, this mechanism facilitates the annotation task and makes our solution practical, accessible and flexible.

Third, this paper presents a generalizable approach to automatically analyze data flows of the composition processes, so that semantic differences between the exchanged data can be determined and reconciled. We show how our approach can be used with BPEL. It is easy to adapt our approach to analyze the data flow of a process specified in many other process modeling languages such as the Process Algebras, UML Activity Diagram and the Business Process Modeling Notation (BPMN), and to extend our solution to address semantic reconciliation for Business Process Integration (BPI) (Becker et al. 2003). For example, applying the solution to service composition in LOTOS, a kind of process algebra, can be found in (Li et al. 2009a; b).

Conclusion

Semantic heterogeneity widely exists among Web services and severely hampers their composition and interoperability. To this end, we adopt the *context* perspective to deal with the data-level semantic heterogeneity, including inconsistent data naming, representation, precision, scaling and unit. We describe the COIN lightweight ontology with structured contexts to define a small set of generic concepts among the services involved in the composition. The multiple specializations of the generic concepts, which are actually used by different services, are structured into different contexts so that the differences can be treated as context differences. We introduce a flexible, standard-compliant mechanism of semantic annotation to elevate the syntactic WSDL descriptions to the COIN ontology. Given the naive BPEL composition ignoring semantic differences, the reconciliation approach can automatically determine context conflicts by reasoning on the COIN ontology and produce the mediated BPEL process by incorporating necessary conversions. The incorporated conversions can be predefined atomic conversions or composite conversions that are dynamically constructed using the atomic ones. As discussed, the COIN-based reconciliation solution has desirable properties of adaptability, extensibility and scalability. In the long run, the solution can significantly alleviate the reconciliation efforts for Web services composition.

Semantic heterogeneity among services exists not only at the data level but also at the structural/schematic level. In the future, we plan to adapt and demonstrate our solution to handle both levels of semantic heterogeneity so that a comprehensive reconciliation solution can be achieved. To reconcile the structural differences between exchanged messages, it may be applicable to define appropriate conversions using XLST for transforming one message schema to another. Also, intelligent techniques need to be developed to facilitate the (semi-)automatic construction of the COIN ontology and contexts. In addition, we plan to extend our solution to deal with various data anomalies (e.g., missing data, redundant data, and conflicting data) in Web services composition and other application domains such as Business Process Integration and scientific workflows.

Acknowledgements

The authors thank the anonymous reviewers and editor for their insightful suggestions in improving this paper. Also, the authors thank Frank Manola, Independent Consultant, Wilmington, MA, USA, for his valuable comments in improving the work. This work was partially supported by the MIT Sloan China Management Education Project and grants from the National Natural Science Foundation of China (No. 60674080 and No. 60704027) and the National High-Tech R&D (863) Plan of China (No. 2007AA04Z150).

[7] http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1

References

Agre, G., and Marinova, Z. "An INFRAWEBS Approach to Dynamic Composition of Semantic Web Services," *Cybernetics and Information Technologies* (7:1), 2007, pp. 45-61.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., and Kartha, N. "Web services business process execution language version 2.0," *OASIS Standard*, 2007.

Becker, J., Dreiling, A., Holten, R., and Ribbert, M. "Specifying information systems for business process integration–A management perspective," *Information Systems and E-Business Management* (1:3), 2003, pp. 231-263.

Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., and Mecella, M. "Automatic composition of transition-based semantic web services with messaging," in *Proceedings of the 31st International Conference on Very Large Data Bases* (VLDB 2005), Trondheim, Norway: VLDB Endowment, 2005, pp. 613-624.

Bressan, S., Goh, C., Levina, N., Madnick, S., Shah, A., and Siegel, M. "Context Knowledge Representation and Reasoning in the Context Interchange System," Applied Intelligence (13:2), 2000, pp. 165-180.

Burstein, M., Bussler, C., Finin, T., Huhns, M.N., Paolucci, M., Sheth, A.P., Williams, S, and Zaremba, M. "A semantic Web services architecture," *IEEE Internet Computing* (9:5), 2005, pp. 72-81.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. "Web services description language (WSDL) 1.1," *W3C Recommendation*, 2001.

Farrell, J., and Lausen, H. "Semantic Annotations for WSDL and XML Schema," *W3C Recommendation*, available at http://www.w3.org/TR/2007/REC-sawsdl-20070828/ 2007.

Fensel, D., and Bussler, C. "The Web Service Modeling Framework WSMF," *Electronic Commerce Research and Applications* (1:2), 2002, pp. 113-137.

Gagne, D., Sabbouh, M., Bennett, S., and Powers, S. "Using Data Semantics to Enable Automatic Composition of Web Services," in *Proceedings of the 3rd IEEE International Conference on Services Computing* (SCC 2006), 2006, pp. 438-444.

Gannon, T., Madnick, S., Moulton, A., Siegel, M., Sabbouh, M., and Zhu, H. "Framework for the Analysis of the Adaptability, Extensibility, and Scalability of Semantic Information Integration and the Context Mediation Approach," in *Proceedings of the 42nd Hawaii International Conference on System Sciences* (HICSS 2009), Hawaii, 2009, pp. 1-11.

Goh, C.H., Bressan, S., Madnick, S., and Siegel, M. "Context interchange: new features and formalisms for the intelligent integration of information," *ACM Transactions on Information Systems* (17:3), 1999, pp. 270-293.

Kopecký, J., Vitvar, T., Bournez, C., and Farrell, J. "SAWSDL: Semantic Annotations for WSDL and XML Schema," *IEEE Internet Computing* 11(6), 2007, pp. 60-67.

Lausen, H., Polleres, A., and Roman, D. "Web Service Modeling Ontology (WSMO)," *W3C Member Submission*, 2005.

Li, X., Madnick, S., Zhu, H., and Fan, Y. "An Approach to Composing Web Services with Context Heterogeneity," in *Proceedings of the 7th International Conference on Web Services* (ICWS 2009), Los Angeles, CA, USA, 2009a, pp. 695-702.

Li, X., Madnick, S., Zhu, H., and Fan, Y. "Improving Data Quality for Web Services Composition," in *Proceedings of the 7th International Workshop on Quality in Databases* (QDB 2009), co-located with t*he 35th International Conference on Very Large Data Bases* (VLDB 2009), Lyon, France, 2009b.

Madnick, S., Firat, A., and Siegel, M. "The Caméléon Web Wrapper Engine," in *Proceedings of the VLDB Workshop on Technologies for E-Services* [SWP #4128, CISL #00-03], Cairo, Egypt, 2000, pp. 269–283.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E., and Srinivasan, N. "Bringing Semantics to Web Services with OWL-S," *World Wide Web* (10:3), 2007, pp. 243-277.

McIlraith, S.A., Son, T.C., and Zeng, H. "Semantic Web Services," IEEE Intelligent Systems (16:2), 2001, pp. 46-53.

Mrissa, M., Ghedira, C., Benslimane, D., and Maamar, Z. "Context and Semantic Composition of Web Services," in *Proceedings of the 17th International Conference on Database and Expert Systems* (DEXA 2006), Krakow, Poland, 2006a, pp. 266-275.

Mrissa, M., Ghedira, C., Benslimane, D., and Maamar, Z. "A Context Model for Semantic Mediation in Web Services Composition," in *Proceedings of the 25th International Conference on Conceptual Modeling* (ER 2006), Tucson, Arizona, USA, 2006b, pp. 12-25.

Mrissa, M., Ghedira, C., Benslimane, D., Maamar, Z., Rosenberg, F., and Dustdar, S. "A context-based mediation approach to compose semantic Web services," ACM Transactions on Internet Technology (8:1), 2007, 4.

Nagarajan, M., Verma, K., Sheth, A.P., Miller, J., and Lathem, J. "Semantic Interoperability of Web Services - Challenges and Experiences," in *Proceedings of 4th International Conference on Web Services* (ICWS 2006), Chicago, USA, 2006, pp. 373-382.

Nagarajan, M., Verma, K., Sheth, A.P., and Miller, J.A. "Ontology driven data mediation in web services," *International Journal of Web Services Research* (4:4), 2007, pp. 104-126.

Papazoglou, M.P., Traverso, P., Dustdar, S., and Leymann, F. "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer* (40:11), 2007, pp. 38-45.

Patil, A.A., Oundhakar, S.A., Sheth, A.P., and Verma, K. "Meteor-s web service annotation framework," in *Proceedings of the13th international conference on World Wide Web* (WWW 2004), 2004, pp. 553-562.

Rao, J., Küngas, P., and Matskin, M. "Composition of semantic web services using linear logic theorem proving," *Information Systems* (31:4-5), 2006, pp. 340-360.

Sabbouh, M., Higginson, J.L., Wan, C., and Bennett, S.R. "Using Mapping Relations to Semi Automatically Compose Web Services," in *Proceedings of IEEE Congress on Services - Part I*, 2008, pp. 211-218.

Seligman, L.J., Rosenthal, A., Lehner, P.E., and Smith, A. "Data Integration: Where Does the Time Go?" *IEEE Data Engineering Bulletin* (25:3), 2002, pp. 3-10.

Sivashanmugam, K., Verma, K., Sheth, A., and Miller, J. "Adding Semantics to Web Services Standards," in *Proceedings of the 1st IEEE International Conference of Web Services* (ICWS 2003), Las Vegas, Nevada, USA, 2003, pp. 395–401.

Spencer, B., and Liu, S. "Inferring Data Transformation Rules to Integrate Semantic Web Services," in *Proceedings of the 3rd International Semantic Web Conference* (ISWC 2004), Hiroshima, Japan, 2004, pp. 456-470.

Sun, S.X., Zhao, J.L., Nunamaker, J.F., and Sheng, O.R.L. "Formulating the data-flow perspective for business process management," *Information Systems Research* (17:4), 2006, pp. 374-391.

Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N. "Automated discovery, interaction and composition of Semantic Web services," *Web Semantics: Science, Services and Agents on the World Wide Web* (1:1), 2003, pp. 27-46.

Verma, K., and Sheth, A. "Semantically Annotating a Web Service," IEEE Internet Computing (11:2), 2007, pp. 83-85.

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. "Ontology-based integration of information-a survey of existing approaches," *IJCAI-01 Workshop on Ontologies and Information Sharing*, Seattle, WA, USA, 2001, pp. 108-117.

Yu, Q., Liu, X., Bouguettaya, A., and Medjahed, B. "Deploying and managing Web services: issues, solutions, and directions," *The International Journal on Very Large Data Bases* (17:3), 2008, pp. 537-572.

Zhang, J., Chang, C.K., Zhang, L.J., and Hung, P.C.K. "Toward a Service-Oriented Development Through a Case Study," *IEEE Transactions on Systems, Man and Cybernetics, Part A* (37:6), 2007, pp. 955-969.

Zhu, H., and Madnick, S. "Scalable Interoperability Through the Use of COIN Lightweight Ontology," in *Proceedings of the 2nd VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems* (ODBIS 2006), Seoul, Korea, 2006, pp. 37-50.