A Context-Based Approach to Reconciling Data Interpretation Conflicts in Web Services Composition

Xitong Li
Stuart Madnick
Hongwei Zhu

Working Paper CISL# 2013-03

March 2013

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E62-422
Massachusetts Institute of Technology
Cambridge, MA 02142

A Context-Based Approach to Reconciling Data Interpretation Conflicts in Web Services Composition

Xitong Li

Sloan School of Management, Massachusetts Institute of Technology, USA; email: xitongli@mit.edu

Stuart Madnick

Sloan School of Management & School of Engineering, Massachusetts Institute of Technology, USA; email: smadnick@mit.edu

Hongwei Zhu

Manning School of Business, University of Massachusetts Lowell, USA; email: hongwei_zhu@uml.edu.

Abstract

Web services composition is often hampered by various types of data misinterpretation problems. In this paper, we present a comprehensive classification of the data misinterpretation problems. To address them, we develop an approach to automatic detection and reconciliation of data interpretation conflicts in Web services composition. The approach uses a lightweight ontology augmented with modifiers, contexts, and atomic conversions between the contexts, implemented using XPath functions and external services. The WSDL descriptions of Web services are annotated to establish correspondences to the ontology and contexts. Given the naive Business Process Execution Language (BPEL) specification of the desired Web services composition with possible data interpretation conflicts, the reconciliation approach can automatically detect the conflicts and produce the corresponding mediated BPEL by incorporating appropriate conversions into the composition. Finally, we develop a prototype to validate and evaluate the reconciliation approach.

1. INTRODUCTION

Service-Oriented Computing (SOC) has become an increasingly important computing paradigm to develop and integrate distributed enterprise IT systems (Papazoglou et al. 2007). As a technology of choice for SOC, Web services, also simply called services, are accessible software components 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 Web Service Description Language (WSDL) (Christensen et al. 2001), the workflow logic of a composite service is usually defined in Business Process Execution Language (BPEL) (Alves et al. 2007), a standard from the Organization for the Advancement of Structured Information Standards (OASIS) for specifying the process of messages exchanged between Web services.

A successful service composition must ensure semantic interoperability so that data can be exchanged unambiguously among the involved services. Unfortunately, semantic interoperability is often hampered by data misinterpretation 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. Apparently, additional information is still needed to correctly utilize the exchanged data. The challenge of data misinterpretation 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 data misinterpretation in Web services composition.

In this paper, we first present several real-world examples[1] of Web services and service composition with data misinterpretation problems. Those examples clearly demonstrate in reality how data misinterpretation affects the use of Web services and hampers their composition. Then, we develop a comprehensive classification of the various data misinterpretation problems that we have observed in the practice of Web services composition. The classification helps identify the scope of the problem domain. To address the challenging problems, we describe our approach to automatic detection and reconciliation of data interpretation conflicts in Web services composition. The approach 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 works of applying the strategy (Li et al. 2009a; Li et al. 2009b; Mrissa et al. 2007) to Web services composition. The approach uses a lightweight ontology to define a common vocabulary capturing only generic concepts shared by the involved services. The lightweight ontology also defines multiple contexts capturing different specializations (which are actually used by the involved services) of the generic concepts. 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 ignoring data misinterpretation the *naive BPEL*. With the above descriptions in place, the reconciliation approach can automatically detect data interpretation conflicts in the naive BPEL and produce the corresponding *mediated BPEL* by incorporating appropriate conversions into the composition. The mediated BPEL composition, now without any data interpretation conflict, is the output of the reconciliation approach and can be successfully deployed and executed.

[1] Some of them are simplified from real-world Web services.

We make three contributions that, to the best of our knowledge, have not appeared elsewhere:

First, we provide a set of new algorithms to automatically analyze data flows of service composition processes and reconcile data misinterpretation problems in the composition processes. The approach can significantly alleviate the reconciliation efforts and accelerate the development of Web services composition. Although the approach is demonstrated with BPEL composition only, it is a generalizable approach and can be easily adapted to analyze the data flow of a process specified in many other process modeling languages, such as process algebra, UML Activity Diagram and the Business Process Modeling Notation (BPMN). Thus, the approach can address semantic reconciliation in a broad context of Business Process Integration (BPI) (Becker et al. 2003) and workflow management (van der Aalst and Kumar 2003).

Second, we extend the W3C standard SAWSDL so that the extended SAWSDL can be used to annotate context information in WSDL descriptions. Specifically, we design two methods for context annotation to alleviate the complexity of handling the evolving data semantics of Web services. The extension for context annotation complies with SAWSDL so that the annotation task can be performed using any existing SAWSDL-aware tools, e.g., Radiant (Verma and Sheth 2007). Thus, this mechanism facilitates the annotation task and makes our approach practical, accessible and flexible.

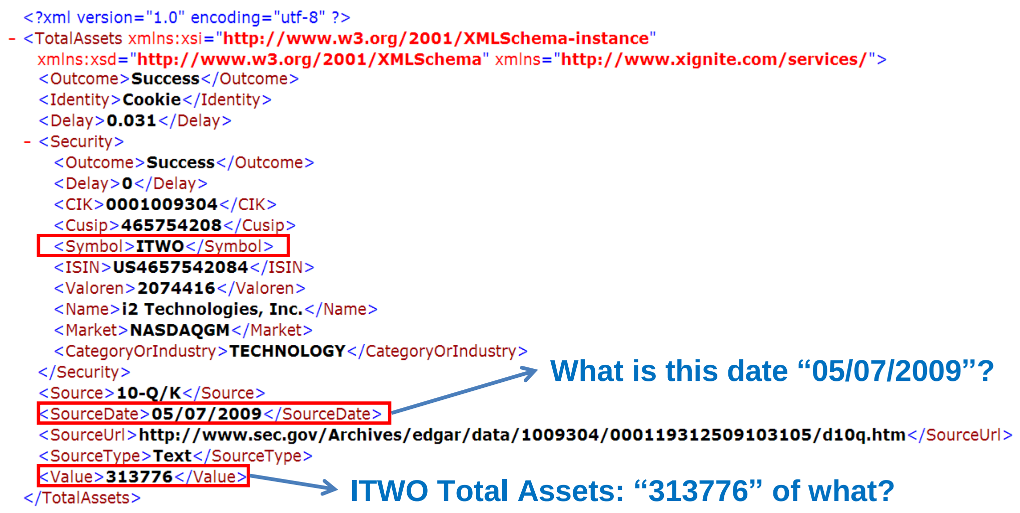
Third, as part of this work, we develop and describe a working prototype – the Context Mediation Tool (CMT). By using the working prototype in a number of examples, we demonstrate the feasibility and applicability of our approach.

The reconciliation approach, as qualitatively and quantitatively evaluated in this paper, has the desirable properties of software development methodology (e.g., adaptability, extensibility and scalability) and can significantly alleviate the reconciliation efforts for Web services composition. Thus, the approach facilitates the application of SOC to develop Web-based information systems. This paper contributes to the literature on Service-Oriented Computing (Papazoglou et al. 2007), Business Process Integration (BPI) (Becker et al. 2003) and workflow management (van der Aalst and Kumar 2003). The rest of the paper is organized as follows. Section 2 describes the challenges of data misinterpretation problems when using and composing Web services. Section 3 and Section 4 present the reconciliation approach and the prototype. Section 5 presents the results of the validation and evaluation. Section 6 discusses the related work. Finally, Section 7 concludes the paper.

2. CHALLENGES OF DATA MISINTERPRETATION PROBLEMS

2.1 Motivating Examples of Web Services

2.1.1. Example 1: A Problematic Web Service. *Xignite, Inc.*, an established U.S. Web services provider, has published a service named *XigniteEdgar* which consumes the stock ticker symbol of a company and returns its total assets. When requested using "ITWO" for i2 Technology, *XigniteEdgar* returns the data as shown in Figure 1. The returned total assets of i2 Technology is associated with the date "05/07/2009". But should the users interpret the date as May 7th, 2009 or July 5th, 2009? How should the total assets of "313776" be interpreted? When invoked with "MSFT" for Microsoft, *XigniteEdgar* returns "68853" as Microsoft's total assets. Is it possible that i2 Technology's total assets are more than four times of Microsoft? Manual investigation shows the numeric figure for i2 Technology is in thousands, whereas that for Microsoft is in millions. If these assumptions of data interpretation were not explicitly clarified, users may incorrectly use *XigniteEdgar*, perhaps causing financial losses.



```
<?xml version="1.0" encoding="utf-8" ?>
- <TotalAssets xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.xignite.com/services/">
   <Outcome>Success</Outcome>
   <Identity>Cookie</Identity>
   <Delay>0.031</Delay>
 - <Security>
     <Outcome>Success</Outcome>
     <Delay>0</Delay>
     <CIK>0001009304</CIK>
     <Cusip>465754208</Cusip>
     <Symbol>ITWO</Symbol>
     <ISIN>US4657542084</ISIN>
     <Valoren>2074416</Valoren>
     <Name>i2 Technologies, Inc.</Name>
     <Market>NASDAQGM</Market>
     <CategoryOrIndustry>TECHNOLOGY</CategoryOrIndustry>
   </Security>
   <Source>10-Q/K</Source>
   <SourceDate>05/07/2009</SourceDate>
   <SourceUrl>http://www.sec.gov/Archives/edgar/data/1009304/000119312509103105/d10q.htm</SourceUrl>
   <SourceType>Text</SourceType>
   <Value>313776</Value>
</TotalAssets>
```

What is this date “05/07/2009”?
ITWO Total Assets: “313776” of what?

Fig. 1. A problematic Web service with ambiguous data interpretation.

2.1.2. Example 2: A Simple Composition of Two Component Services. Let’s consider a simple composition scenario with only two services in which a Chinese developer wants to develop a composite service *ConfHotelDeals*. Its function is to consume an international conference code and return the hotel expenses in the city where the conference is held. With the purpose of exploiting reuse, the developer decides to implement *ConfHotelDeals* by composing two existing services: *ConfInfo* and *HotwireDeals*.[2] Given a conference code, the operation *queryConfInfo* of *ConfInfo* provides basic information of the conference, including start and end dates and the city where the conference is held. The operation *queryDeals* of *HotwireDeals* returns the room charges of the deals based on the city name and start/end dates. The composition process is illustrated in Figure 2. Unfortunately, these services have different assumptions about data interpretation. *ConfHotelDeals* is intended to return the monetary expenses in Chinese yuan (“RMB”) and the hotel expense includes the value-added taxes. *ConfInfo* provides the dates in “dd-mm-yyyy”. *HotwireDeals* assumes dates are in “mm/dd/yyyy” and returns the hotel deals in US dollars (“USD”) without value-added taxes. If the data misinterpretation problems were not properly resolved, conflicts would happen in the composition process (as noted in Figure 2 by little “explosions”) and the composite service *ConfHotelDeals* would not work correctly.

2.1.3. Example 3: Composition Example of Multiple Services. Now let’s consider a somewhat complicated scenario that a U.K. developer wants to develop a new Web service, *OpeningPriceMarketCap* (denoted as *CS* for *C*omposite *S*ervice), to obtain the opening stock price and market capitalization of a U.S. company on its <u>first</u> trading day. *CS* is intended for a U.K. analyst to monitor 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.

[2] HotwireDeals originates from Hotwire.com, available at http://developer.hotwire.com/docs/Hotel_Deals_API.

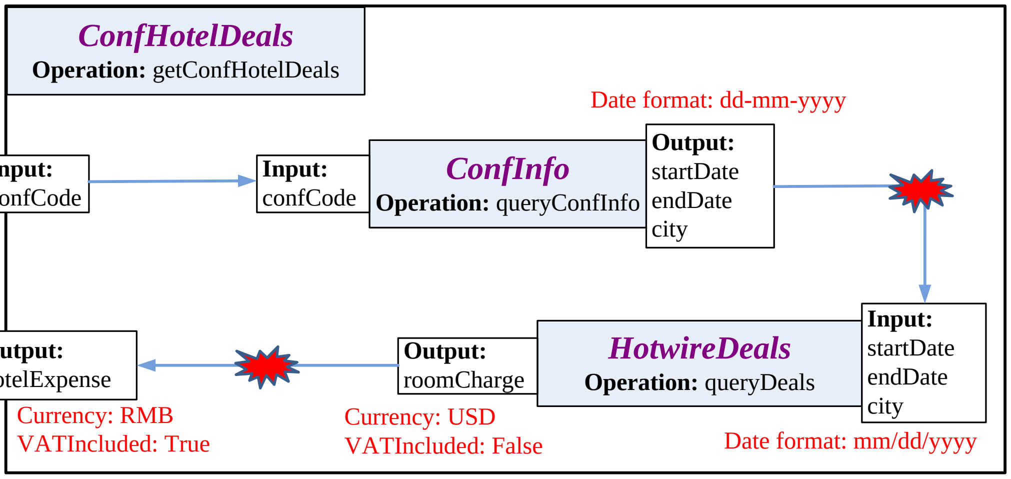


Fig. 2. Example 2: simple composition of two component services.

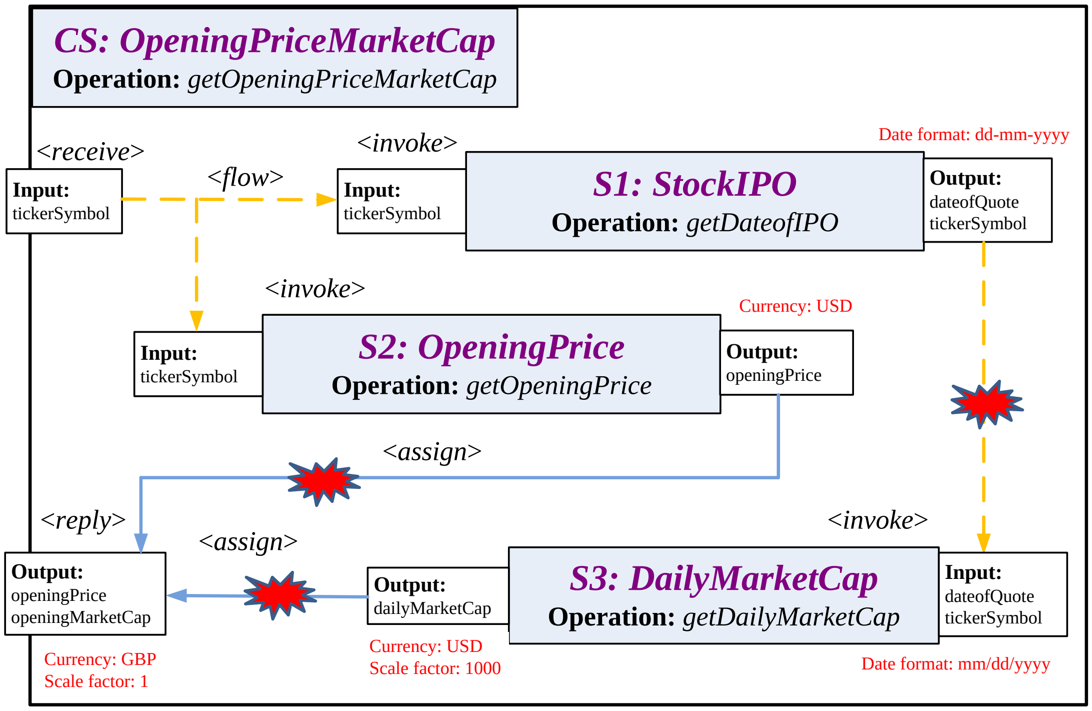


Fig. 3. Example 3: composition of multiple services.

In principle, *CS* can be accomplished by a composition of *S1*, *S2* and *S3*. Specifically, the input *tickerSymbol* of *CS* needs to be transferred to both *S1* and *S2*. 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, such as ActiveVOS BPEL Designer.[3] The BPEL composition is graphically illustrated in Figure 3, where BPEL activities (e.g., <receive>, <invoke>) are enclosed in angle brackets. Since these four services are developed by independent providers, they have

[3] http://www.activevos.com/

different assumptions about data interpretation in terms of data format, currency, and scale factors, as summarized in Table 1.

Table 1. Different Assumptions of Data Interpretation

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

Note that usually these assumptions are not explicitly represented in WSDL descriptions. As a result, existing BPEL tools (e.g., ActiveVOS BPEL Designer) cannot detect these conflicting assumptions and fail to alert data misinterpretation problems in the composition because the interpretation conflicts exist at the data instance level. If not reconciled, the data interpretation conflicts would result in severe errors and failures during the execution of the composition. This composition example (i.e., Example 3) will be used as the "walk-through" example in the rest of the paper.

2.2 Classification of Data Misinterpretation Problems

We classify data misinterpretation problems into *representational*, *conceptual* and *temporal* categories, as summarized in Table 2. The purpose of the classification is to help readers understand the problem scope of our solution and meanwhile draw the boundary of our study. Note that there exist a number of classification frameworks in the literature (Nagarajan et al., 2006; Sheth et al., 2005; Halevy, 2005). Those existing classifications tend to cover a broader range of semantic heterogeneity issues, some of which can be addressed by our approach (e.g., scale factors, currency), while others are not the focus of this paper, such as structural/schematic differences. The classification presented here exclusively focuses on data interpretation conflicts that may occur in Web services.

2.2.1. Representational. Different organizations may use different representations for a certain concept, which can result in representational misinterpretation problems. Five subcategories can be further identified at this level: *format*, *encoding*, *unit of measure*, *scale factor*, and *precision*. Format differences occur because there often exist multiple format standards, such as for representing date, time, geographic coordinates, and even numbers (e.g., "1,234.56" in USA would be represented as "1.234,56" in Europe). Encoding differences may be the most frequent cause of representational misinterpretation, because there are often multiple coding standards. For example, the frequently used coding standards for countries include the FIPS 2-character alpha codes, the ISO3166 2-character alpha codes, 3-character alpha codes, and 3-digit numeric codes. Also, IATA and ICAO are two standards for airport codes. Data misinterpretation problem can occur in the presence of different encoding standards (e.g., country code "BG" can stand for Bulgaria or Bangladesh, depending on whether the standard is ISO or FIPS). Besides the format and encoding differences, numeric figures are usually represented using different units of measure, scale factors, and precisions. For example, financial services use different currencies to report the data to consumers who prefer to use their local currencies. Scientific services may use different units of measure to record the data (e.g., meter or feet).

Table 2. Classification of Data Misinterpretation Problems

Categories		Explanations / Examples
Representational	Format	Different format standards for date, time, geographic coordinate, etc. Example: "05/07/2009" vs. "2009-05-07"
	Encoding	Different codes for country, airport, ticker symbol, etc. Example: Male/Female vs. M/F vs. H/D[4] vs. 0/1
	Unit of measure	Different units of currency, length, weight, etc. Example: 10 "USD" vs. 10 "EUR"
	Scale factor	Different scale factors of numeric figures Example: 10 "Billion"[5] vs. 10 "Million"
	Precision	Different precisions of numeric figures Example: "5.8126" vs. "5.81"
Conceptual	Subtle differences in conceptual extension	Different interpretations about whether or not a specific entity should be included Example: does the reported retail "price" include value-added taxes or not?
Temporal	Representational and conceptual data interpretation may change over time	Prices listed in Turkey are implicitly in Turkish liras (TRL) before 2005 but in Turkish New Lira (TRY) after January 1, 2005.

2.2.2. Conceptual. The same term and representation is often used to refer to similar but slightly different data concepts. This category of misinterpretation usually occurs when the extension of the concept has different assumptions of the interpretation, such as whether or not a specific entity is included by the concept. For example, a retail price reported by European services usually includes the value-added taxes, while retail prices reported by US services, especially for purchases to be done in a store, usually do not include the value-added taxes.[6] An even more challenging problem in this category is referred to as "Corporate Householding" (Madnick et al. 2003) which refers to misinterpretation of corporate household data. For example, the answer to "What were the total sales of IBM" varies depending on whether the sales of majority owned subsidiaries of IBM should be included or not. The answers can be very different due to different reporting rules adopted in different countries or for different purposes. Besides the entity aggregation issue, the conceptual extension of the inter-entity relationship may also have different interpretations. For instance, in order to answer the question "How much did MIT purchase from IBM in the last fiscal year?", we need to clarify whether the purchasing relationship between MIT and IBM should be interpreted as *direct purchasing* (i.e., purchased directly from IBM) or *indirect purchasing* through other channels (e.g., third-party brokers, distributors, retailers). In some cases, only the direct purchasing from IBM to MIT are considered, whereas in other cases indirect purchasing through other channels also needs to be included (Madnick and Zhu 2006).

2.2.3. Temporal. Most of the above-mentioned possibilities of data interpretation may change over time (Zhu and Madnick 2009). For example, a Turkish auction service may have listed prices in millions of Turkish liras (TRL),[7] but after the Turkish New Lira (TRY) was introduced on January 1, 2005, it may start to list prices in unit of Turkish New Lira. Also, an accounting service may or may not

[4] In France.

[5] Of course, these categories can be nested – for example, there can be different meanings of scale factor, such as "Billion" means one thousand million in USA but it used to mean one million million in the UK.

[6] Usually called "sales taxes" in the USA

[7] About one million TRL equaled one US dollar.

aggregate the earnings of Merrill Lynch into that of Bank of America which acquired the former in September 2008. Considering the highly dynamic and distributed environment of Web services, these data misinterpretation problems resulting from the temporal evolvement would become very challenging. Due to length limit, we will not address the temporal issues in this paper, but our approach can be extended to resolve them.

2.3 Deficiency of Existing Approaches

To address the abovementioned problems, we must identify the data interpretation conflicts that may occur in naive BPEL composition and rewrite it to reconcile the identified conflicts. The existing approaches usually perform the identification and reconciliation of interpretation conflicts in a manual way. As depicted in the upper half of Figure 4, after the naive BPEL is produced, a manual inspection of potential conflicts is conducted. Once an interpretation conflict is detected, the naive BPEL is modified by inserting an ad-hoc conversion to transform the output of the upstream service to the needed input of the downstream one. These steps (as indicated as "Identify conflicts" and "Rewrite") are continued iteratively until a valid BPEL is produced. The ad-hoc, "brute-force" approaches tend to produce "spaghetti" code that is difficult to debug and maintain. In summary, the brute-force approaches suffer from the following deficiencies: 1) It is error-prone to manually inspect the naive BPEL, especially when the composition involves a large number of data elements as well as Web services and has complicated workflow logic. Also, it is error-prone to manually define customized conversion code and insert it to the composition; 2) It is difficult to reuse the conversion code, as it usually defined and inserted in the composition in an ad-hoc way; and 3) Every time an involved service is changed (or removed) or a new service is added, the *Identifying conflicts* and *Rewrite* steps need to be manually performed again and new custom conversions may need to be inserted in the composition. As a result, the brute-force approaches potentially make the number of custom conversions very large and difficult to maintain over time.

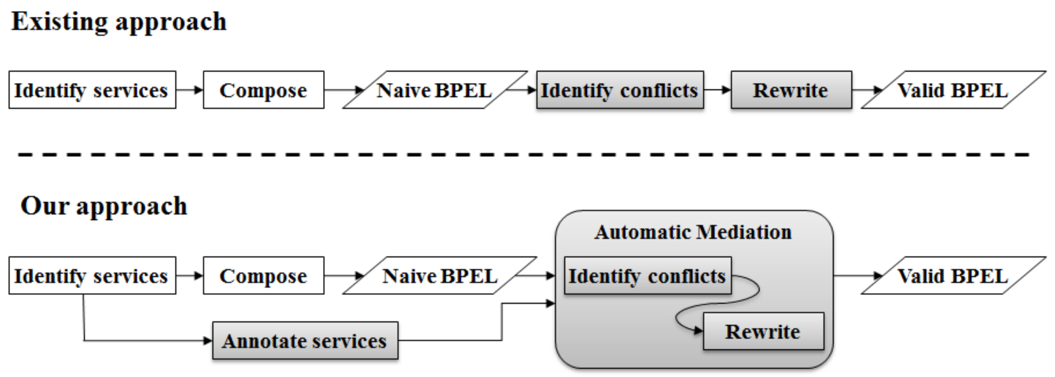


Fig. 4. Comparison of existing approach and our proposed approach.

The situation could become even worse when the number of services involved in the composition is large and the involved services are highly dynamic. For example, the recent SOA implementation of a Texas health and human resource system consists of over a hundred Web services and more than 20 composite services.[8] According to a recent Application Integration Survey, data integration accounts for about 40% of software development costs.[9] Another survey conducted in 2002 reveals

[8] Source from the email communication between the authors and SourcePulse.com, a software services firm.

[9] http://www.slideshare.net/mlbrodie/powerlimits-of-relational-technology

that approximately 70% of the integration costs were spent on identifying interpretation differences and developing custom code to reconcile these differences (Seligman et al. 2002). Therefore, it is important to develop a systematic and disciplined approach to addressing the various data misinterpretation problems for Web services composition.

We have developed an improved approach to rectify these deficiencies. Our approach automates the "*Identify conflicts*" and "*Rewrite*" steps as an intelligent mediation step (see the lower half of Figure 4). By using the proposed approach, developers do not need to read the naive BPEL to identify the conflicts or to decide where the conversions need to be inserted. We provide a tool that fully automates the mediation step and produces the valid BPEL.

Note that our approach requires the services in the composition be annotated to explicitly capture the assumptions that affect the interpretations of data. Although semantic annotation is a new step, it allows for the separation of declarative semantic descriptions from the programming code. It also enables automatic identification and reconciliation of semantic conflicts. As we will show in Section 5.2.2, this separation offers tremendous benefits to our approach.

3. CONTEXT-BASED APPROACH

In this section, we describe our context-based approach to reconciling data interpretation conflicts in Web services composition. The approach consists of methods for representing semantic assumptions and mediation algorithms for identifying conflicts and rewriting the BPEL to reconcile the identified conflicts. The lightweight ontology (Zhu and Madnick 2007) is used to facilitate semantic annotation.

3.1 Representation of Ontology and Contexts

3.1.1. Lightweight Ontology. Ontology is a collection of concepts and the relationships between these concepts. Ontologies are often used for Web query processing (Storey et al. 2008), Web services composition (Mrissa et al. 2007), and data reliability assessment (Krishnan et al. 2005). In practice, there are various types of ontologies ranging from lightweight, rather informal, to heavyweight, more formal ones (Wache et al. 2001). Lightweight ontologies are simple and easy to create and maintain since they only include the high-level concepts. On the other hand, they do not directly provide all the depth and details of a typical formal ontology. In contrast, formal ontologies are often relatively complex and difficult to create (Zhu and Madnick 2007).

To combine the strengths and avoid weaknesses of these ontology approaches, we adopt an augmented lightweight ontology approach that allows us to automatically derive a fully specified ontology from concisely described high-level concepts and contexts. By "lightweight", we mean the ontology only requires generic concepts used by the involved services and the hierarchical relationships between the concepts. The different assumptions of the services for interpreting the generic concepts are represented as contexts using the vocabulary and structure offered by the ontology.

Figure 5 presents a graphical representation of the lightweight ontology for Example 3 (see Section 2.1.3). Concepts are depicted by round rectangles and *basic* is the special concept from which all other concepts inherit. Like traditional ontologies, the lightweight ontology has two 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 the *date* concept is the "date of" attribute of concept *stockMoneyValue*. In practice, it is frequently straightforward to identify generic concepts among multiple independent services. For example, *S3* has an output *dailyMarketCap* and *CS* has an output *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* using 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 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.

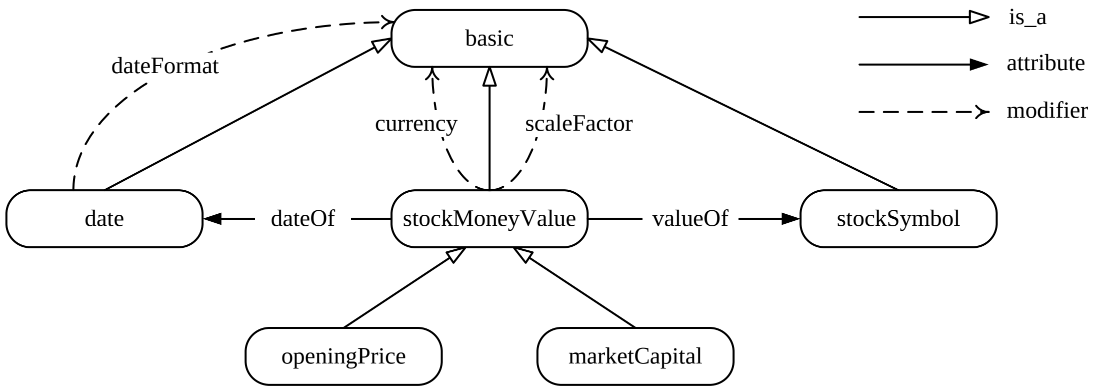


Fig. 5. Lightweight ontology shared by involved services of the composition.

Modifiers are depicted by dashed arrows in Figure 5. 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 1, the modifier value of *currency* is "USD" in case of *S2* and *S3*. That means that different services may be associated with different values assigned 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 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 1 are described using four contexts associated with the four services involved in the composition, as shown in Table 3. As a result, interpretation differences among these services can be treated as context differences.

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>]

3.1.2. Semantic and Context Annotation. Web services are usually 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 data elements of WSDL descriptions and the concepts of an ontological model (Patil et al. 2004; Sivashanmugam et al. 2003). The annotations are recommended to 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). Thus, SAWSDL is an appropriate industrial standard for us to establish the correspondence between the syntactic WSDL descriptions and the lightweight ontology.

SAWSDL provides an attribute *modelReference* for specifying the correspondence between WSDL components (e.g., data/element types, input and output messages) and the concepts of an ontology. However, SAWSDL *per se* does not provide any mechanism for context annotation that is required for resolving data misinterpretation problems in service composition. Thus, we extend SAWSDL with two annotation methods that use the modelReference attribute: (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 can be useful to simplify the annotation task.

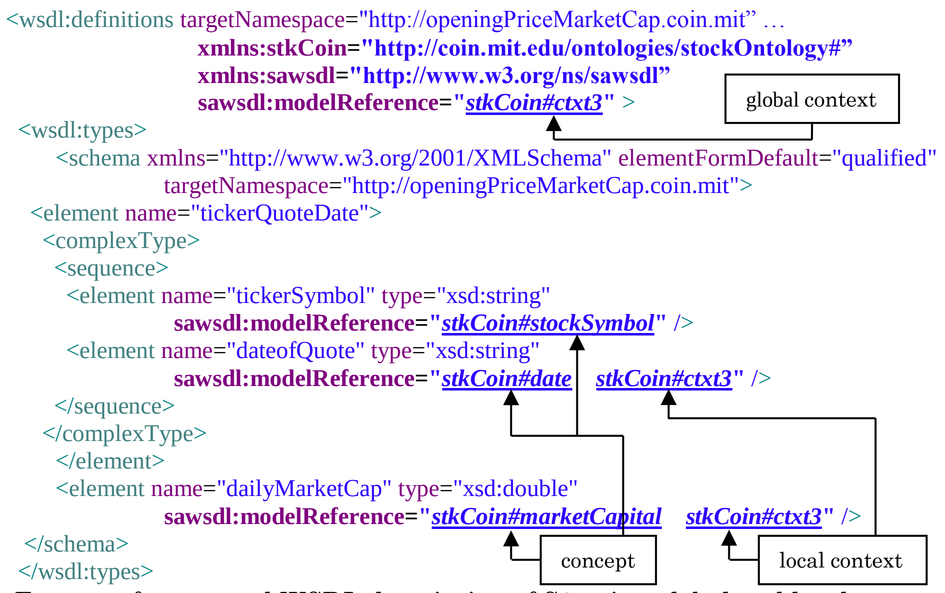

```
<wsdl:definitions targetNamespace="http://openingPriceMarketCap.coin.mit" …
                    xmlns:stkCoin="http://coin.mit.edu/ontologies/stockOntology#"
                    xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
                    sawsdl:modelReference="stkCoin#ctxt3" >
 <wsdl:types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
              targetNamespace="http://openingPriceMarketCap.coin.mit">
  <element name="tickerQuoteDate">
   <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>
```

Fig. 6. Excerpt of annotated WSDL description of S3 using global and local context annotations

Figure 6 shows the annotated part of *S3*'s WSDL description 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 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 ontology. The modelReference attribute of a data element can have one value, or two values separated by a whitespace.[10] 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. Both the global and local context annotations are used in Figure 6. Although the local annotation does not actually override the global context, we include it for illustration purposes.

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 approach can automatically determine and reconcile possible interpretation differences resulting from the date format change. As a result, the global and local context annotations promote the flexibility of our solution to handle the evolving semantics of services.

3.1.3. Conversions between Different Contexts. Context differences, once detected, need to be reconciled using conversion programs to convert 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 for individual modifiers are called *atomic conversions*. At least one atomic conversion is specified for each modifier to reconcile the difference indicated by different modifier values. Since there exist three modifiers in the example ontology (see Figure 5 and Table 3), we specify three atomic conversions: $cvt_{\text{dateFormat}}$, 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[11] or an external (e.g., third-party) service. 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:

$cvt_{\text{dateFormat}}$: *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:[12]

$cvt_{\text{scaleFactor}}$: *Vt* = *Vs* * *mvs* div *mvt*

[10] SAWSDL allows the modelReference attribute to have multiple values separated by whitespaces.

[11] The BPEL specification and most BPEL engines (e.g., ActiveBPEL) support XPath 1.0.

[12] Note that this is a general purpose conversion function that works for any values of *mvs* and *mvt*.

In complex cases, the conversions may 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 *CurrencyConverter*[13] (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}.

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) and thus is not limited to 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.

3.2 Reconciliation Algorithms

In Web services composition, context conflicts can occur when a piece of data from the source service in one context is transferred to, and consumed by, the target service in another context. Figure 7 shows the typical scenario where a context conflict occurs in the composition. In Figure 7, there exists a data transfer where the data *data_s* from service *WS_s* is transferred to service *WS_t* and consumed as data *data_t*. Using context annotation, both *data_s* and *data_t* are instances of concept *C* which has a modifier *m*. Also, *WS_s* and *WS_t* are annotated with two different contexts *ctxt_s*, *ctxt_t*, respectively. As a result, according to the context definition of the ontology, *data_s* and *data_t* are interpreted differently by *WS_s* and *WS_t* if the modifier value of *m* in *ctxt_s* (i.e., *mvs*) is different from the value *mvt* of *m* in *ctxt_t*. In such a case, a context conflict occurs within the data transfer. In the following sections, we present three successive algorithms that automate the identification and reconciliation of context conflicts in the composition process. Example 3 will be used to demonstrate the algorithms.

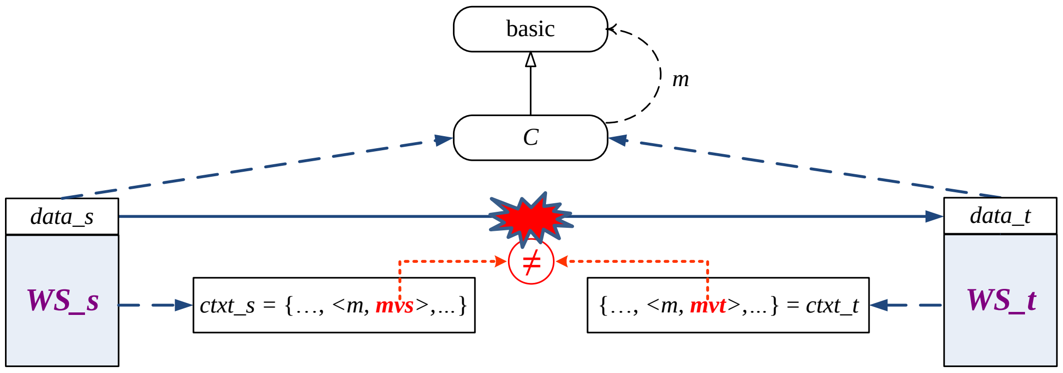


Fig. 7. Scenario of context conflict in Web services composition.

3.2.1. Identifying Data Transfers.
Recall that the BPEL composition that ignores 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* and *ws_t* are the source and target services, *data_s* and *data_t* are the

[13] *CurrencyConverter* originates from http://www.ac-markets.com/forex-resources/currency-converter.aspx External services for conversions may also need to be annotated with concepts and contexts.

data elements involved in the data transfer, and *type* indicates if the data transfer is explicit or implicit.

$$dataTrans(type, data\_s, ws\_s, data\_t, ws\_t)$$

Each explicit data transfer involves two variables and can be easily identified according to the <assign> activity which is used to copy the data from the source variable to the target variable. As shown in Figure 3, there are two <assign> activities in the composition process of Example 3: they are to transfer the data *dailyMarketCap* and *openingPrice*, respectively. Thus, two explicit data transfers are identified.

Each implicit data transfer involves one variable shared by two activities interacting with participant services having potentially different contexts. The BPEL specification provides four types of interaction activities: <receive>, <reply>, <invoke>, and <onMessage> contained in <pick>. For an output variable, its source interaction activity may be <receive>, <onMessage> or <invoke>. For an input variable, its target interaction may be <reply> or <invoke>. By examining each variable in the composition, all implicit data transfers in the BPEL composition can be identified.

Algorithm 1. Identifying Explicit and Implicit Data Transfers

```
Input: BPEL process proc.
Output: The set of explicit data transfers $EDT = \{edt\}$,
        the set of implicit data transfers $IDT = \{idt\}$.
1.  set $EDT = \emptyset$, $IDT = \emptyset$;
2.  for each <assign> activity asn in proc
3.    var_s ← getSourceVariable(asn), var_t ← getTargetVariable(asn)
4.    act_s ← getSourceInteractionActivity(proc, asn),
5.    act_t ← getTargetInteractionActivity(proc, asn)
6.    edt ← getDataTransfer(var_s, var_t, act_s, act_t)
7.    $EDT \leftarrow EDT \cup \{edt\}$
8.  for each variable var in proc
9.    $L_{var}$ ← getInteractionActivitySeries(proc, var)
10.     for each source activity act_s1 in $L_{var}$
11.       act_s2 ← getNextSourceActivity($L_{var}$, act_s1),
12.       $T_{var}$ ← getTargetActivitySeries($L_{var}$, act_s1, act_s2)
13.       for each target activity act_t in $T_{var}$
14.         idt ← getDataTransfer(var, act_s1, act_t)
15.         $IDT \leftarrow IDT \cup \{idt\}$
16. return EDT, IDT;
```

Table 4. Data Transfers in the Composition Process of Example 3

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*, *marketCap*, *S3*, *openingMarketCap*, *CS*)

Algorithm 1 is developed to identify explicit and implicit data transfers. Using Algorithm 1, three implicit and two explicit data transfers are identified in the composition process of Example 3, as shown in Table 4. Instead of explicitly using the <assign> activity, the output of *S1* is directly transferred and consumed as the input of *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. In Figure 3, the composition process involves <receive>, <reply> and <invoke>; it does not involve <onMessage>.

3.2.2. Detecting 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. When 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 earlier in Figure 7. 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 ontology. 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 through data transfer *dt3* from *S1* to *S3*. There is no conflict for *tickerSymbol* because it has no modifier.

Each context conflict can be represented using the following form:

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

where *dt* is the data transfer in which the context conflict occurs. $[(m_i, mvs_i, mvt_i)]_{i=\{1,\dots,n\}}$ depicts the array of n modifiers with different values in *ctxt_s* and *ctxt_t*. Algorithm 2 is developed to automate the procedure of conflict determination. As shown in Table 5, three context conflicts in the naive BPEL composition are determined.

Algorithm 2. Detecting Context Conflicts

```
Input: BPEL process proc, the set of data transfers $DT = \{dt\}$,
          the set of annotated WSDL description $WS = \{ws\}$, Ontology onto;
Output: The set of context conflicts $CC = \{cc\}$;
1. set $CC = \varnothing$
2. for each data transfer dt in DT
3.    ws_s ← getSourceService(dt, proc, WS), ws_t ← getTargetService(dt, proc, WS)
4.    data_s ← getSourceDataElement(ws_s, dt),  data_t ← getTargetDataElement(ws_t, dt)
5.    c ← getConcept(ws_s, data_s)
6.    ctxt_s ← getContext(ws_s, data_s), ctxt_t ← getContext(ws_t, data_t)
7.    for each modifier m of c in onto
8.       mvs ← getModifierValue(c, m, ctxt_s), mvt ← getModifierValue(c, m, ctxt_t)
9.          if $mvs \neq mvt$
10.          then cc ← getContextConflict(C, m, ctxt_s, ctxt_t, mvs, mvt)
11.               $CC \leftarrow CC \cup \{cc\}$
12. return CC;
```

Table 5. Context Conflicts in the Composition Process of Example 3

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*, *marketCap*, *ctxt3*, *ctxt0*, [(scaleFactor, "1000", "1"); (currency, "USD", "GBP")])

3.2.3. Incorporating Conversions. Once a context conflict is determined within a data transfer, it is needed to assemble an appropriate conversion to reconcile the conflict. The appropriate conversion is either a predefined atomic conversion or a composite one assembled using several atomic conversions. For reconciliation, the identified conversion is incorporated into the data transfer to convert the data in the source context to the target context.

When 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 as *var*, and to insert an <assign> activity into the data transfer for copying *var* to *var_t*. As shown in Table 5, 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 as variable *tickerQuoteDate*. Since *tickerQuoteDate* has two data elements *dateofQuote* and *tickerSymbol*, the <assign> activity inserted into *dt3* has two <copy> activities 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 made explicit.

When a context conflict involves only one modifier, it can be reconciled using a predefined atomic conversion. 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 in Section 3.1.3, cvt_{currency} is implemented by the external currency converter service *S4* rather than using XPath function. Thus, an <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.

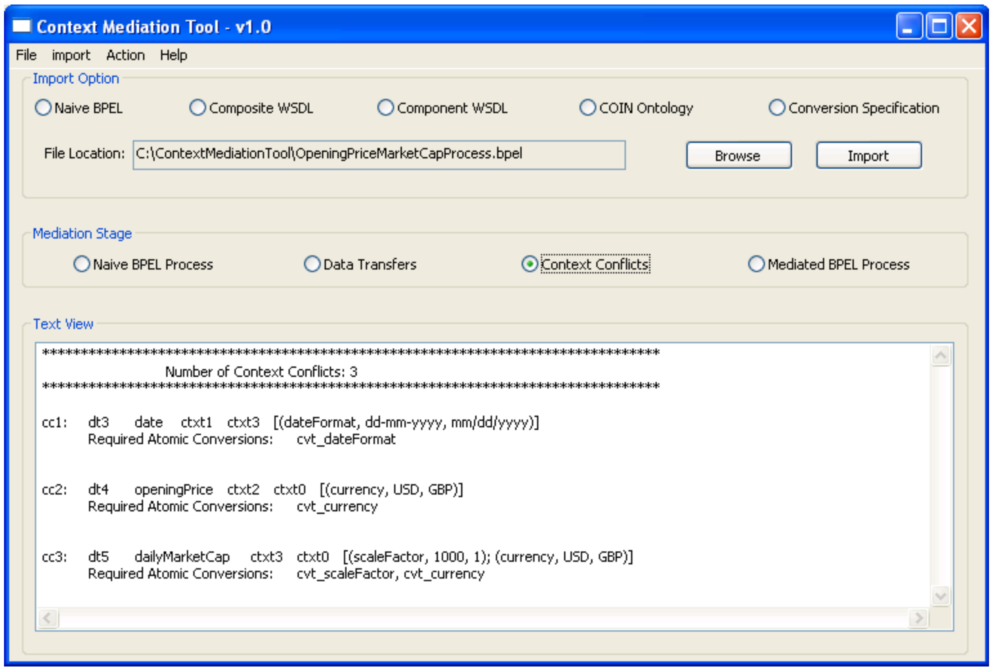
Algorithm 3. Incorporating Conversions

```
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), insertConversion(mediatedProc, cvt_i)
14. return mediatedProc;
```

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 this case, 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*. Algorithm 3 is developed to automate the procedure of assembling conversions and generating the mediated BPEL to reconcile the determined context conflicts.

4. PROTOTYPE IMPLEMENTATION

We implemented a proof-of-concept prototype, named Context Mediation Tool (CMT), as a JAVA application, to demonstrate the reconciliation approach. The lightweight ontology with structured contexts is defined using the COIN Model Application Editor[14] which is a Web-based tool for creating and editing COIN-style ontology and contexts in RDF/OWL. Atomic conversions between the contexts are defined in a specification file. The WSDL descriptions of the composite and component services (e.g., *CS* and *S1* ~ *S3* of Example 3) are annotated using our context annotation method. To facilitate the annotation task, we extended an open-source Eclipse plug-in for semantic annotation (i.e., Radiant[15]) and developed the context annotation tool *Radiant4Context*. We assume naive BPEL composition processes with possible data misinterpretation problems are defined using any typical BPEL tool.



File Location: C:\ContextMediationTool\OpeningPriceMarketCapProcess.bpel

Mediation Stage: [✓ Context Conflicts]

```
*********************************************************************************
                    Number of Context Conflicts: 3
*********************************************************************************

cc1:   dt3    date    ctxt1   ctxt3   [(dateFormat, dd-mm-yyyy, mm/dd/yyyy)]
       Required Atomic Conversions:    cvt_dateFormat


cc2:   dt4    openingPrice   ctxt2   ctxt0   [(currency, USD, GBP)]
       Required Atomic Conversions:    cvt_currency


cc3:   dt5    dailyMarketCap    ctxt3   ctxt0   [(scaleFactor, 1000, 1); (currency, USD, GBP)]
       Required Atomic Conversions:    cvt_scaleFactor, cvt_currency
```

Fig. 8. Snapshot of CMT at Stage Context Conflicts.

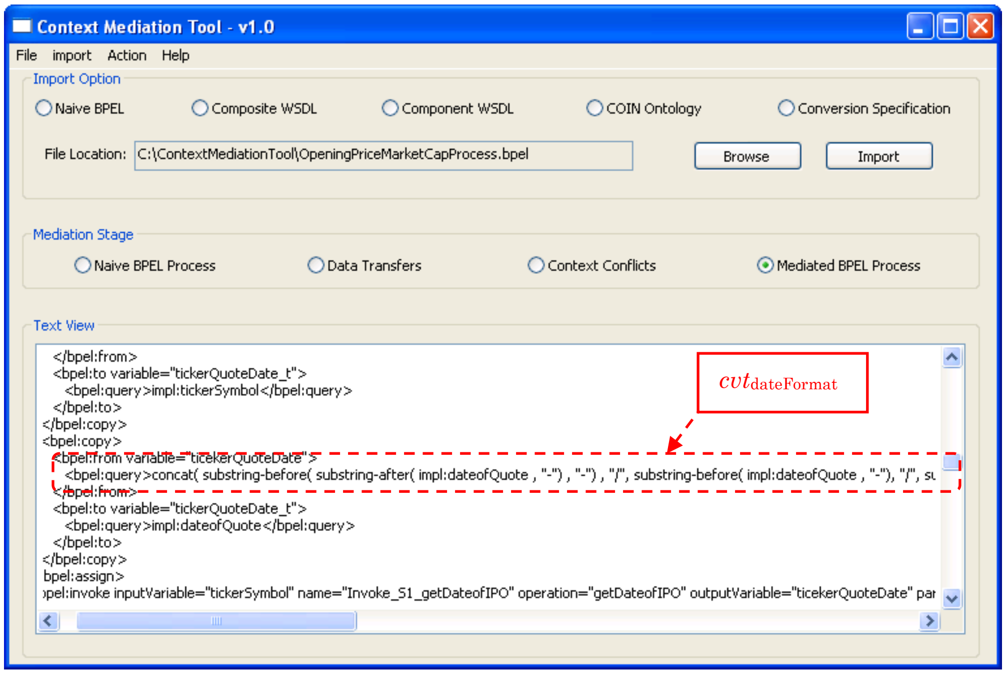
[14] http://interchange.mit.edu/appEditor/TextInterface.aspx?location=MIT

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

CMT is used to create a mediation project and consume all the above documents. The reasoning engine implemented within CMT can automatically perform the reconciliation algorithms described in Section 3.2. Take Example 3 for instance. CMT first performs Algorithm 1 to identify the three implicit and two explicit data transfers in the naive BPEL composition process. Then, CMT continues to use Algorithm 2 to determine the three context conflicts. Finally, CMT uses Algorithm 3 to select three atomic conversions $cvt_{\text{dateFormat}}$, $cvt_{\text{scaleFactor}}$ and cvt_{currency} from predefined conversion library [16] and incorporates them into corresponding data transfers to reconcile the conflicts.

CMT has three working areas for the mediation tasks, as shown in Figure 8. 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, 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 provide the intermediate and final results that the approach produces while addressing context differences among services involved in the composition. Eventually, CMT produces the mediated BPEL composition process. Note that CMT can perform all the mediation steps in an automatic and consecutive way.

Figure 8 shows the snapshot of CMT at the stage Context Conflicts where the three context conflicts in the composition process of Example 3 and corresponding atomic conversions required for the reconciliation are identified. At the stage Mediated BPEL Process, CMT produces the mediated BPEL composition process with incorporated conversions. Figure 9 shows the snapshot of CMT in which the XPath function for the conversion $cvt_{\text{dateFormat}}$ is embedded in the mediated BPEL composition process.



File Location: C:\ContextMediationTool\OpeningPriceMarketCapProcess.bpel

Mediation Stage: Naive BPEL Process Data Transfers Context Conflicts [✓ Mediated BPEL Process]

```
  </bpel:from>
  <bpel:to variable="tickerQuoteDate_t">
    <bpel:query>impl:tickerSymbol</bpel:query>
  </bpel:to>
</bpel:copy>
<bpel:copy>
  <bpel:from variable="ticekerQuoteDate">
    <bpel:query>concat( substring-before( substring-after( impl:dateofQuote , "-") , "-") , "/", substring-before( impl:dateofQuote , "-"), "/", su
  </bpel:from>
  <bpel:to variable="tickerQuoteDate_t">
    <bpel:query>impl:dateofQuote</bpel:query>
  </bpel:to>
</bpel:copy>
bpel:assign>
›pel:invoke inputVariable="tickerSymbol" name="Invoke_S1_getDateofIPO" operation="getDateofIPO" outputVariable="ticekerQuoteDate" par
```

Fig. 9. Snapshot of CMT at Stage Mediated BPEL Process.

[16] We recommend that libraries of such atom conversions be established that can be reused for future compositions.

5. VALIDATION AND EVALUATION

5.1 Validation

We validated the solution approach by applying it to several composition processes that involve various interpretation conflicts. Here we show the results of applying the approach to Example 3 (see Section 2.1.3) and Example 2 (see Section 2.1.2). For Example 3, Figure 10 shows the snapshot of the naive BPEL composition process defined using ActiveVOS BPEL Designer. Note that we have used a schematic notation in Figure 3 to illustrate the naïve BPEL composition process. Since the interpretation conflicts exist at the data instance level, ActiveVOS BPEL Designer cannot detect the conflicts of data interpretation and fails to alert any error. But severe errors and failures will occur when one attempts to executes the naive BPEL composition.

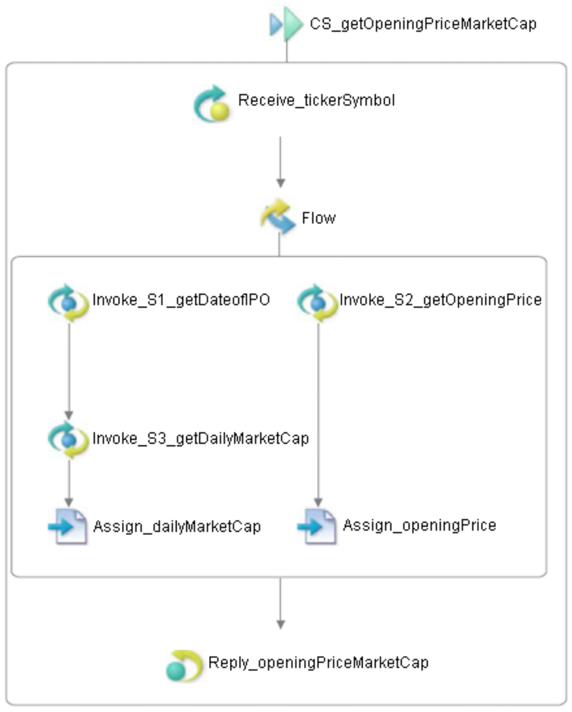

CS_getOpeningPriceMarketCap
Receive_tickerSymbol
Flow
Invoke_S1_getDateofIPO
Invoke_S2_getOpeningPrice
Invoke_S3_getDailyMarketCap
Assign_dailyMarketCap
Assign_openingPrice
Reply_openingPriceMarketCap

Fig. 10. Naive BPEL composition process with context conflicts.

The prototype CMT can automatically produce the mediated BPEL composition consecutively. After the mediated BPEL composition is produced, we import it into ActiveVOS BPEL Designer for validation purpose. Figure 11 shows the snapshot of the mediated BPEL process with the incorporated conversions. As we can see, CMT inserts a <assign> activity into the composition process between the invocations of *S1* and *S3* in order to reconcile the conflict of date format (i.e., *cc1* in Table 5). In fact, CMT embeds the XPath conversion function $cvt_{\text{dateFormat}}$ in the <copy> element of the <assign> activity and uses it to convert the date format from "dd-mm-yyyy" to "mm/dd/yyyy". To reconcile the conflict of currency (i.e., *cc2* in Table 5), CMT inserts the invocation of the external currency converter service *S4*. By invoking *S4*, the output *openingPrice* in "USD" from *S2* is converted to the equivalent price in "GBP" as the output of *CS*. Finally, CMT inserts a <assign> activity and a <invoke> activity consecutively in the composition process to reconcile the conflicts of scale factor and currency (i.e., *cc3* in Table 5). The XPath conversion function $cvt_{\text{scaleFactor}}$ is embedded by CMT in the <copy> element of the <assign> activity and used to reconcile the conflict of scale factor. *S4* is used to reconcile the conflict of currency (see *cc2* and *cc3* in Figure 11).

In order to validate the correctness of the mediated BPEL composition process, we provide a number of testing data values for the input of *CS* and the output of the services (i.e., *S1* ~ *S3* and *S4*). We utilize the simulation feature of ActiveVOS BPEL Designer to simulate the execution of the mediated BPEL process. The execution results indicated that: a) the mediated BPEL process properly completed without any deadlocks or errors; b) all the context conflicts were successfully reconciled – different date formats, scale factors and currencies were correctly converted between the involved services; and c) *CS* produced the expected output: *openingPrice* and *openingMarketCap*.

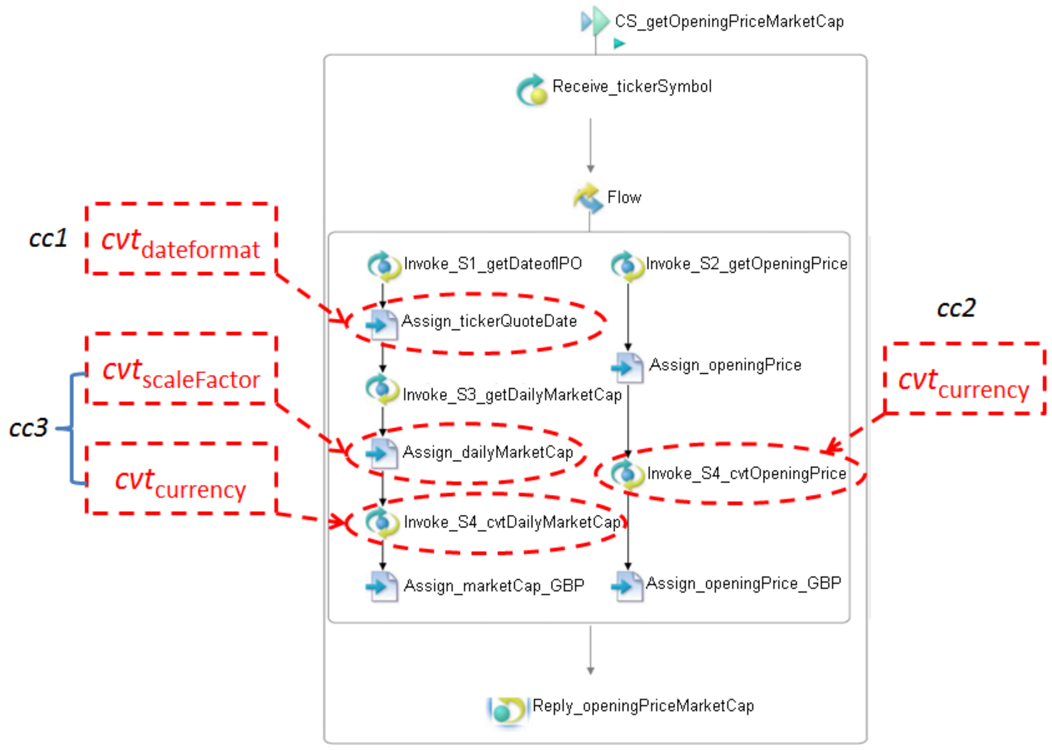


Fig. 11. Mediated BPEL composition process with incorporated conversions.

For Example 2 (see Figure 2), three context conflicts are determined using CMT: the date format difference, the currency difference, and the VAT difference – *HotwireDeals* provides the room charge not including value-added taxes, while *ConfHotelDeals* is expected to provide the hotel expense including the taxes. Similar to Example 3, the date format difference and the currency difference can be resolved by $cvt_{dateFormat}$ and $cvt_{currency}$, respectively. Differently, the VAT difference needs to be resolved by using a new conversion cvt_{VAT} which is implemented as an external service *TaxesCalculator*. *TaxesCalculator*'s operation *getVATAdded* consumes a money value without value-added taxes and returns the money value with value-added taxes. In a similar way, CMT produces the mediated composition for *ConfHotelDeals* with all determined context conflicts reconciled. Figure 12 illustrates the mediated composition process with all necessary conversions (indicated in bolded red boxes) inserted.

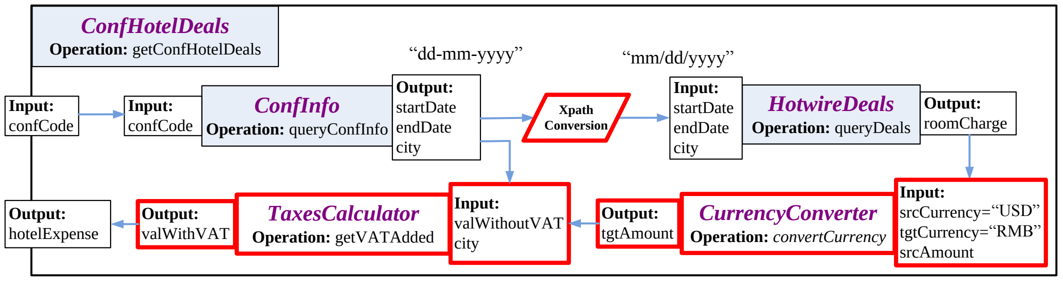


Fig. 12. Mediated composition with conceptual VAT difference reconciled.

5.2 Evaluation

The reconciliation approach is evaluated both qualitatively and quantitatively. The evaluation results are presented in the following two subsections.

5.2.1. Qualitative Evaluation. The qualitative evaluation of the reconciliation approach is conducted by checking whether it can handle more general and complicated composition situations. Specifically, we try to answer the following two questions: (1) What types of data misinterpretation problems can the approach address? and (2) What types of Web services composition can the approach support? The method of qualitative evaluation used in this paper is similar to the method of key feature comparison, which is a credible method for evaluating software engineering-based approaches (VIDE 2009) and recently used by (Abeywickrama and Ramakrishnan 2012) as well[17].

For the first question, we find that the use of modifiers in a lightweight ontology is a quite versatile modeling technique. It allows for the representation of each type of interpretation conflicts discussed in Section 2.2. For example, to address the difference of date format or currency (a kind of unit of measure) at the representational level, we use the modifier of date format or currency and corresponding conversions (i.e., $cvt_{\mathrm{dateFormat}}$, cvt_{currency}) and demonstrate the feasibility through Example 3. In Example 2 we use the modifier of value-added taxes and the conversion cvt_{VAT} to deal with the difference of value-added taxes, a kind of conceptual-level data misinterpretation problems. Other conceptual-level problems like those of "Corporate Householding" (Madnick et al. 2003) and temporal-level problems (Zhu and Madnick 2009) can also be modeled using appropriate modifiers and addressed in a similar way. With the ontology/context modeling and semantic annotation in place, all the possible data misinterpretation problems in Table 2 that may occur in Web services composition can be addressed by the approach.

Since BPEL becomes the OASIS standard for defining Web services composition in practice, the approach presented in this paper focuses on addressing BPEL-based composition processes. BPEL specification provides four types of interaction activities (i.e., <receive>, <reply>, <invoke> and <onMessage> within <pick>) to define interaction patterns between the composition process and participant services. Also, BPEL provides several basic workflow constructs (e.g., *sequence*, *parallel*, *choice* and *iteration*) to define the composition processes. In our work all these interaction activities and workflow constructs have been taken into consideration when we developed Algorithm 1. In other words, Algorithm 1 can be used to automatically inspect any composition process defined using BPEL and identify data transfers within the process. For example, we demonstrate the capability of the approach to address Example 3 which involves three of the four types of interaction activities (i.e.,

[17] Note that the key feature comparison of our work with the prior approaches is presented in Section 6.

<receive>, <reply>, <invoke>) and the *sequential* and *parallel* workflow constructs. <onMessage> is similar to <receive>, as both handle the message arrival. Thus, Algorithm 1 analyzes <onMessage> in a similar way as it does for <receive>. Since control-flow conditions of choice and iteration are irrelevant to the identification of data transfers, Algorithm 1 will examine each workflow branch defined by the construct of choice or iteration in a similar way as it does for the sequence or parallel workflows. After the data transfers in the composition process are identified using Algorithm 1, Algorithm 2 and Algorithm 3 are used to determine and reconcile possible data interpretation conflicts. Therefore, the approach can support any Web services composition defined using the BPEL and WSDL standards.

5.2.2. Quantitative Evaluation. A quantitative evaluation of the proposed approach is carried out with the focus on assessing human efforts needed for reconciling data interpretation conflicts. Although a direct measurement of human efforts can be obtained through empirical experiments, it is often difficult to set up such appropriate experiments to reliably and objectively measure the evaluation metrics. Instead, we will consider the complexity of how mediation is accomplished in the brute-force approach compared with our approach.

Let us suppose an extreme case where there are *N* services (including the composite service) that have different data interpretations and interact with each other in the composition. In such a case, there are *N*(N-1)/2* service-to-service interactions in the composition. Thus, the brute-force approach (see the discussion in Section 2.3) has to examine each of the service-to-service interactions to ensure the interoperability between every two interacting services. Each service-to-service interaction involves an XML message probably with multiple data elements. Suppose on average there are *K* data elements in the XML message between any two services and *D* dimensions of data interpretation conflicts (e.g., currency and scale factor) associated with each data element, then in total the brute-force approach has to examine *K*D*N*(N-1)/2* possible places where data interpretation conflicts might occur. Wherever a data interpretation conflict is detected, the brute-force approach has to construct a conversion and insert it to the appropriate place in the composition. As the number of services *N* and the number of data elements in XML messages *K* increase, the amount of manual work of inspecting and rewriting BPEL increases quickly. Maintaining manually created BPEL over time is also labor-intensive and error-prone.

In contrast, our reconciliation approach requires manual creation of a lightweight ontology, annotation of each service, and provision of atomic conversions, each of which concerns only one data interpretation dimension. Although this may appear to be undesirable beforehand, it actually reduces the amount of pairwise manual inspection and conversion construction using annotation for individual services. More importantly, our approach can automatically examine the XML message between each service-to-service interaction, identify context conflicts, and build and insert appropriate conversions in the composition. Thus, the key advantage of our reconciliation approach lies in the automatic generation of mediated BPEL which otherwise would require significant amount of manual work as in the brute-force approach.

Let us use a specific example to demonstrate the advantage of our approach. Assume that the developer of Example 3 later wanted to serve diverse users that require any combination of 10 different currencies and 4 scale factors (i.e., 1, 1K, 1M, 1B). The component services, e.g., *S3*, may also change their currencies and scale factors. In such a case, both the output *dailyMarketCap* of *S3* and the output *openingMarketCap* of *CS* may use 40 (=10×4) different data interpretations. To convert the output *dailyMarketCap* of *S3* to the output *openingMarketCap* of *CS*, it would be most likely for the developers to manually specify 1560 (=39×40) custom conversions if they used the brute-force approach. An even worse case would arise if currencies and scale factors of *CS*, *S2* and *S3* changed over time independently. Comparatively, our approach only requires two parameterized conversions (i.e., $cvt_{\text{scaleFactor}}$ and cvt_{currency}). More importantly, as long as no additional dimension of data interpretation difference is introduced, there is no need to define new conversions even if the involved services were to be added (or removed) in the composition, or the workflow logic of the composition process were to be changed. In practice such situations frequently happen because the implementations of Web services and service composition often evolve in the fast-changing global business environment.

There are two points to note regarding the examples in this paper. First, for reasons of brevity and simplicity, the examples in the paper only include a few web services. There are large complex applications built using hundreds of web services, they would not be so easy for a human to examine the naive BPEL and resolve all the conflicts – and do that error-free. Second, the scalability issue not only exists at initial development of the composite application but over its entire life cycle. If a change is needed to the application or happens to the specifications of one or more of the web services, then the entire resolution process must be reviewed and appropriate changes made by the human. With our approach, most of this is automated, only the context specifications (and occasionally the ontology) have to be updated.

6. RELATED WORK AND COMPARISON

The basic Web services standards (e.g., WSDL, BPEL) generally ignore data semantics, rendering semantic composition and interoperability far from reality. A research area, referred to as Semantic Web Services (SWSs), has emerged to apply Semantic Web technologies to Web services (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.

Data misinterpretation among Web services can be considered as a semantic heterogeneity problem. However, the literature provides only a few approaches to handle the challenging problem in Web services composition. The initial work in (Spencer and Liu 2004) proposes to use data transformation rules to convert the data exchanged between services. This work requires a common ontology described in OWL (particularly in description logic) and the correspondences between the ontology and WSDL descriptions defined using OWL-S. Rather than using OWL-S, the approach in (Nagarajan et al. 2006; Nagarajan et al. 2007) proposes to perform semantic annotation by using WSDL-S which is the ancestor of SAWSDL and more consistent with existing industrial standards and practices. The approach focuses on addressing schematic differences of the exchanged messages by using schematic conversations (e.g., XSLT). The work in (Gagne et al. 2006; Sabbouh et al. 2008) proposes a set of mapping relations to establish direct correspondences between the messages of two WSDL-based services. Then, the common ontology can be constructed based on these correspondences and data-level differences are resolved by predefined conversions. Generally, those approaches require each participant services to be annotated and mapped to a common ontology serving as the global schema. However, it is more costly 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 or transformation rules required by those approaches are created manually to perform direct conversions between the exchanged messages. In contrast, the actual conversions in our approach can be automatically composed using a small number of atomic, parameterized conversions. Furthermore, those approaches only focus on dealing with a pair of participant services, rather than a composition consisting of multiple services.

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 this problem by separating ontology and context definitions from the annotated WSDL descriptions. (3) 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. (4) Since in their work each context conflict needs to be reconciled using the a priori specification of an external service, they miss the opportunity to reuse predefined atomic conversions and the capability of conversion composition. 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.

In addition to the literature on Web services, it is worth noting some interesting works (Sun et al. 2006; Tan et al. 2009; Hamid et al. 2010) from the domain of process/workflow management. Sun et al. (2006) develop a data-flow specification for detecting data-flow anomalies within a process/workflow, including missing data, redundant data and potential data conflicts. With a different focus from our work, 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 in their work. We believe that Algorithm 1 can be adapted to construct data-flow specification, so that potential data-flow anomalies can be also addressed. Both Tan et al. (2009) and Hamid et al. (2010) focus on developing mediator services that could address the workflow inconsistencies between services involved in the composition. Our work complements those studies in that we focus on resolving data misinterpretation conflicts in the composition.

7. CONCLUSIONS

Differences of data interpretation widely exist among Web services and severely hamper their composition and interoperability. To this end, we adopt the context perspective to deal with the data misinterpretation problems. We describe the 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 relate the syntactic WSDL descriptions to the ontology. Given the naive BPEL composition ignoring semantic differences, the reconciliation approach can automatically determine context conflicts and produce the mediated BPEL that incorporates necessary conversions. The incorporated conversions can be predefined atomic conversions or composite conversions that are dynamically constructed using the atomic ones. The context-based reconciliation approach has desirable properties of adaptability, extensibility and scalability. In the long run, it can significantly alleviate the reconciliation efforts for Web services composition.

Our approach has two limitations. First, the lightweight ontology enriched with modifiers and contexts needs to be defined manually. Although the ontology has a small number of generic concepts compared to other heavyweight ontologies, efforts are required to define the ontology. Second, our approach requires the participant services be annotated with respect to the ontology. Although it is a nontrivial task, the semantic annotation allows for separation of declarative semantic descriptions from the programming code (e.g., JAVA and ASP.NET) and provides the prerequisite through which our approach can automatically detect and reconcile the data misinterpretation conflicts. To alleviate the cost of the annotation task, we have extended an open-source Eclipse plug-in (i.e., Radiant) and developed a context annotation tool. Thus, developers can easily use our context annotation tool to add context information.

Fortunately, there has been a growing trend (Savas et al. 2009) that authors of data services are encouraged to provide certain metadata definition and semantic annotation. Also, researchers have begun to develop various solutions (Uren et al. 2006; Mrissa et al. 2007; Di Lorenzo et al. 2009), albeit with limited scope, to produce context information for interpreting the data provided by Web services. Therefore, we expect over time such context information will become increasingly available in the published Web services so that our proposed approach can be used more easily and smoothly.

Future work is needed to address the limitations of our approach. Specifically, we plan to develop techniques to automate the construction of the lightweight ontology for Web services. Also, we intend to integrate existing annotation methods (Uren et al. 2006) with our approach to facilitate semantic annotation. Additionally, we plan to adapt several existing service discovery techniques and integrate them with our approach so that the necessary external mediation services could be more easily discovered and used by the tool CMT.

Despite the identified future work, our approach, even in its current form, can substantially reduce the effort and possible errors of manual Web services composition. We expect our approach and the prototype can be applied in the practice of SOC and the development of Web-based information systems.

REFERENCES

Abeywickrama, D., Ramakrishnan, S. Context-aware services engineering: models, transformation and verification. ACM Transactions on Internet Technology 2012; 11(3), No. 10.

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; 11.

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 2003; 1(3): 231-263.

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 2000; 13(2): 165-180.

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

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

Di Lorenzo, G., Hacid, H., Paik, H. and Benatallah, B. Data integration in mashups. ACM SIGMOD Record, 38 (1), 2009. 59-66.

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 2002; 1(2): 113-137.

Gagne, D, Sabbouh, M, Bennett, S, and Powers, S. Using Data Semantics to Enable Automatic Composition of Web Services, Services Computing, 2006. SCC '06. IEEE International Conference on, 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, System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on, Hawaii, 2009; pp. 1-11.

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

Halevy, A. Why Your Data Won't Mix. Queue 2005; 3(8): 50-58.

Motahari Nezhad, H., Xu, G., and Benatallah, B. Protocol-aware matching of web service interfaces for adapter development. Proceedings of the 19th international conference on World Wide Web (WWW 2010): pp. 731-740.

Kopecký, J, Vitvar, T, Bournez, C, and Farrell, J. SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE INTERNET COMPUTING 2007; 11(6): 60-67.

Krishnan, R, Peters, J, Padman, R, and Kaplan, D. On data reliability assessment in accounting information systems. Information Systems Research 2005; 16(3): 307.

Lausen, H, Polleres, A, and Roman, D. Web Service Modeling Ontology (WSMO). W3C Member Submission 2005; 3.

Li, X, Madnick, S, Zhu, H, and Fan, Y. An Approach to Composing Web Services with Context Heterogeneity, Prof. of the 7th Intl. Conf. on Web Services (ICWS 2009), Los Angeles, CA, USA, 2009a; pp. 695-702.

Li, X, Madnick, S, Zhu, H, and Fan, YS. Reconciling semantic heterogeneity in Web services composition, Proceedings of the 30th International Conference on Information Systems (ICIS 2009), Phoenix, AZ, USA, 2009b.

Madnick, S, Firat, A, and Siegel, M. The Caméléon Web Wrapper Engine, Proceedings of the VLDB Workshop on Technologies for E-Services, Cairo, Egypt, 2000; pp. 269–283.

Madnick, S, Wang, R, and Xian, X. The design and implementation of a corporate householding knowledge processor to improve data quality. Journal of management information systems 2003; 20(3): 41-70.

Madnick, S, and Zhu, H. Improving data quality through effective use of data semantics. Data & Knowledge Engineering 2006; 59(2): 460-475.

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 2007; 10(3): 243-277.

McIlraith, SA, Son, TC, and Zeng, H. Semantic Web Services. IEEE Intelligent Systems 2001; 16(2): 46-53.

Motahari Nezhad, H,, Xu, G,, and Benatallah, B. Protocol-aware matching of web service interfaces for adapter development. Proceedings of the 19th international conference on World Wide Web (WWW 2010); pp. 731-740.

Mrissa, M, Ghedira, C, Benslimane, D, and Maamar, Z. Context and Semantic Composition of Web Services, Proc. of the 17th International Conference on Database and Expert Systems, 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, Proc. of the 25th International Conference on Conceptual Modeling, 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 2007; 8(1): 4.

Nagarajan, M, Verma, K, Sheth, AP, Miller, J, and Lathem, J. Semantic Interoperability of Web Services - Challenges and Experiences, Proc. of 4th International Conference on Web Services, Chicago, USA, 2006; pp. 373-382.

Nagarajan, M, Verma, K, Sheth, AP, and Miller, JA. Ontology driven data mediation in web services. International Journal of Web Services Research 2007; 4(4): 104-126.

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

Patil, AA, Oundhakar, SA, Sheth, AP, and Verma, K. Meteor-s web service annotation framework, Proceedings of the13th international conference on World Wide Web, 2004; pp. 553-562.

Sabbouh, M, Higginson, JL, Wan, C, and Bennett, SR. Using Mapping Relations to Semi Automatically Compose Web Services, Services - Part I, 2008. IEEE Congress on, 2008; pp. 211-218.

Savas, P, Evelyne, V, and Tony, H. A "Smart" Cyberinfrastructure for Research. Communications of the ACM 2009; 52(12): 33-37.

Seligman, LJ, Rosenthal, A, Lehner, PE, and Smith, A. Data Integration: Where Does the Time Go? IEEE Data Engineering Bulletin 2002; 25(3): 3-10.

Sheth, A, Ramakrishnan, C, Thomas, C. Semantics for the Semantic Web: The Implicit, the Formal and the Powerful. International Journal on Semantic Web & Information Systems 2005; 1(1): pp. 1-18.

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

Spencer, B, and Liu, S. Inferring Data Transformation Rules to Integrate Semantic Web Services, Proceedings of the 3rd International Semantic Web Conference Hiroshima, Japan: Springer Verlag, 2004; pp. 456-470.

Storey, V, Burton-Jones, A, Sugumaran, V, and Purao, S. CONQUER: A Methodology for Context-Aware Query Processing on the World Wide Web. Information Systems Research 2008; 19(1): 3-25.

Sun, SX, Zhao, JL, Nunamaker, JF, and Sheng, ORL. Formulating the data-flow perspective for business process management. Information Systems Research 2006; 17(4): 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 2003; 1(1): 27-46.

Tan, W, Fan, Y, Zhou, M. A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. IEEE Transactions on Automation Science and Engineering 6(1): 94-106 (2009)

Uren, V, Cimiano, P, Iria, J, Handschuh, S, Vargas-Vera, M, Motta, E, and Ciravegna, F. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. Web Semantics: Science, Services and Agents on the World Wide Web 2006; 4(1): 14-28.

van der Aalst, W, and Kumar, A. XML-based schema definition for support of interorganizational workflow. Information Systems Research 2003; 14(1): 23-46.

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

VIDE. 2009. VIsualize all moDel drivEn programming. Report WP11: Deliverable number D11.3 (Supported by the European Commission within Sixth Framework Programme), Polish-Japanese Institute of Information Technology. Jan. http://www.vide-ist.eu/download/VIDE_D11.3.pdf (Last accessed on 01/14/2012).

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 2008; 17(3): 537-572.

Zhu, H. Effective Information Integration and Reutilization: Solutions to Technological Deficiency and Legal Uncertainty, MIT Ph.D. Thesis, 2005.

Zhu, H, and Madnick, S. Reconciliation of Temporal Semantic Heterogeneity in Evolving Information Systems. Ingénierie des Systèmes d'Information (Networking and Information Systems) 2009; 14(6): 59-74.

Zhu, H, and Madnick, SE. Scalable Interoperability Through the Use of COIN Lightweight Ontology, The 2nd VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems (ODBIS 2007), Seoul, Korea: SPRINGER-VERLAG, 2007; 37-50.