Improving Data Quality for Web Services Composition

Xitong Li
Stuart Madnick
Hongwei Zhu
Yushun Fan

Working Paper CISL# 2009-14

September 2009

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

Improving Data Quality for Web Services Composition

Xitong Li[1,2,†] Stuart Madnick[2] Hongwei Zhu[3] Yushun Fan[1]

[1]Tsinghua University, Beijing 100084, China
lxt04@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn
[2]MIT Sloan School of Management, Cambridge, MA 02142, USA
smadnick@mit.edu
[3]Old Dominion University, Norfolk, VA 23529, USA
hzhu@odu.edu

ABSTRACT

Independently developed Web services often have different assumptions about the interpretation of the exchanged data, such as inconsistent data representation, unit, precision, and scaling. In practice, data misinterpretation results in many data quality problems and further hampers the execution of service composition. In this paper, we present a context-based mediation approach to handle inconsistent data interpretations and improve data quality for Web services composition. The assumptions about data interpretation of the involved services are made explicit and represented as *contexts*. A common ontology is defined to describe the contexts. Necessary conversions between elements of the contexts are implemented using XPath functions and external (e.g., third-party) services to reconcile inconsistent contexts. The WSDL descriptions of Web services are annotated with appropriate contexts using the W3C standard SAWSDL. Given a naïve composition ignoring contexts, the reasoning engine can automatically detect context conflicts within the naïve composition and reconcile these conflicts by producing a mediated composition that incorporates appropriate conversions. A proof-of-concept prototype, Context Mediation Tool (*CMT*), has been developed to validate and demonstrate the approach.

1. INTRODUCTION

Web services have become a promising technology to develop and integrate distributed, Web-based applications [1]. Service composition addresses the situation in which a business need cannot be accomplished by a single component service, whereas a composite service consisting of a combination of multiple independent services working together could satisfy the need. While interfaces of single services are described by the Web Service Description Language (WSDL) [2], process logics of service compositions are usually specified in the Web Service Business Process Execution Language (WS-BPEL) [3]. In practice, the successful execution of service compositions can be hampered by many data quality problems which result from the misinterpretation of heterogeneous data semantics among disparate Web services, such as inconsistent data representation, unit, precision and scaling.

Since basic protocol standards of Web services (e.g., WSDL, WS-BPEL) widely ignore data semantics, existing initiatives, e.g., OWL-S [4], WSMF/WSMO [5, 6] and METEOR-S [7, 8], have developed languages and frameworks to explicitly add semantics into Web services descriptions. Despite these existing efforts, data quality problems in Web services composition are still open and need to be addressed. Certain data quality problems are somewhat subtle, because when they are not reconciled, service composition can still execute but the results may be wrong or susceptible to misinterpretation. For example, a gallon in the U.S. (so-called U.S. gallon) is approximately 3785 ml while the "same" gallon in the U.K. (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. For a given data element (e.g., *volume* and *price*) we often need additional information (e.g., *U.S. gallon* as unit of measure for *volume* and *USD* as currency of *price*) to interpret the meaning of the data element. Such additional information is often implicitly assumed by Web services. Independently developed Web services often have different assumptions about data interpretation. The complexity of addressing data misinterpretation and improving data quality grows when composing multiple services developed by independent providers that are distributed throughout the world.

In this paper, we present an approach to automatic determination and reconciliation of heterogeneous data interpretations in Web services composition with the purpose of improving data quality. Our approach is inspired by the Context Interchange (COIN) strategy for semantic interoperability among multiple data sources [9, 10] and the preliminary work of applying the strategy to service composition [11]. The approach requires composition developers to define a common ontology using an ontology expression language (e.g., RDFS, OWL-Lite) so that the exchanged data in a service composition can be understood at a generic conceptual level. The common ontology captures only the generic concepts among the services involved in the composition. Their various specifications, which are actually used by different

† The work is done while the author studies as a visiting Ph.D. student at MIT Sloan School of Management.

International Workshop on Quality in Databases (*QDB*), August 24, 2009, Lyon, France.

services, are represented using their context descriptions. Conversions between different contexts need to be provided so that these differences, once detected, can be reconciled. Then, the WSDL descriptions of the involved services need to be annotated to establish the correspondence between the data elements in the WSDL descriptions and the concepts in the common ontology. We use the W3C standard, the Semantic Annotation for WSDL and XML Schema (SAWSDL) [12] for the annotation. Further, the service composition is specified in the BPEL specification. This BPEL composition need not concern with context differences and is called the naïve BPEL. With the above descriptions in place, the approach presented in this paper first translates the annotated WSDL and the naïve BPEL to a formal description language, LOTOS NT [13]. LOTOS NT and its supporting tool CADP [14] provide the capability of formally analyzing both the static and dynamic aspects of service compositions. Based on LOTOS NT, the approach can automatically detect which and where context conflicts occur for the exchanged data in the composition, and then reconcile the detected conflicts by incorporating necessary conversions into a mediated composition. Finally, the mediated composition, now without any context conflicts, is translated back as the deployable code in BPEL, called the mediated BPEL.

The rest of the paper is organized as follows. Section 2 presents a motivating example that will be used to demonstrate our approach throughout the paper. Section 3 describes the mechanism for composition developers to define the common ontology and represent contexts. Conversions for reconciling these differences are also described. Section 4 introduces the method to annotate WSDL descriptions with semantics using SAWSDL. Section 5 presents the approach to automatically reconciling context conflicts in Web services composition. Section 6 introduces our mediation tool as a proof-of-concept of the approach. Section 7 reviews the related work. Finally, Section 8 concludes the paper and highlights the future work.

2. MOTIVATING EXAMPLE

Let us consider a scenario that an U.K. developer wants to develop a Web service, *QuoteOpeningPriceWS* (denoted as *CS* for short), to obtain the opening price of an U.S. company's stock on its first trading day. *CS* is intended for U.K. analysts to monitor the U.S. stock market. The developer decides to implement the service by composing existing services. After searching a public service registry[1], the developer discovers two existing Web services that could be combined as components to develop *CS*, i.e., *StockIPOWS* and *HistoricalStockQuoteWS*, denoted as *S1* and *S2* respectively for short. *S1* provides the functionality for querying the IPO[2] information of a company traded in the U.S. The operation *getDateofIPO* of *S1* returns the IPO date when it is queried by using the company's ticker symbol. *S2* provides historical stock quotes for companies traded in the U.S. The operation *getDailyOpenPrice* of *S2* returns the daily open price[3] of a company's stock on a given date.

[1] Web Service Search Engine @ seekda.com - http://seekda.com/.

[2] Initial Public Offering (IPO) is when a company issues common stock to the public for the first time.

[3] In this paper the "opening price" specifically refers to the open price of a company's stock on its first trading day, which can be valuable information for investors analyzing the performance of the stock.

The signatures of the involved services are summarized in Table 1. For simplicity, we do not show the verbose WSDL and BPEL code, and assume the low-level messages of these services have compatible data types (e.g., string, double). As shown in Figure 1, it appears that *CS* can be composed by feeding the output of the operation *getDateofIPO* of *S1* as the input to the operation *getDailyOpenPrice* of *S2*.

Table 1. Signatures of involved Web services

Service	**Operation**	**Input**	**Output**
CS	getOpeningPrice	tickerSymbol	openingPrice
S1	getDateofIPO	tickerSymbol	dateofIPO, tickerSymbol
S2	getDailyOpenPrice	dateofQuote, tickerSymbol	dailyOpenPrice

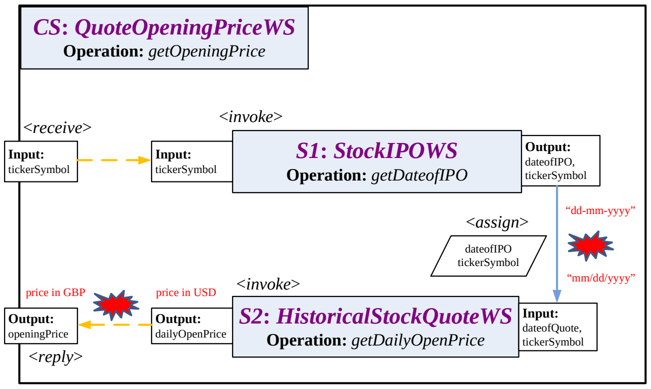


Figure 1. Composition scenario of the motivating example with data misinterpretation problems.

However, even if the inputs and outputs of these services are semantically compatible at a generic level, the simply composed *CS* (called the naïve composition) cannot correctly execute without considering the different assumptions of data interpretation: *CS* and *S1* use the date format "dd-mm-yyyy" and quote stock prices in the currency of British pounds (i.e., "GBP"), while *S2* uses the date format "mm/dd/yyyy" and quotes stock prices in the currency of U.S. dollars (i.e., "USD"). Unfortunately, in practice such assumptions are usually not explicitly represented in service descriptions and severely undermine data quality of Web services composition.

3. REPRESENTATION OF ONTOLOGY AND CONTEXT

3.1 Context-enriched Ontology Model

Ontology has been widely used as a formal representation of concepts and the relationships between these concepts. Our approach allows composition developers to define a common ontology to capture a set of generic concepts among the involved services and provides a mechanism to accommodate multiple specializations for interpreting the generic concepts.

Figure 2 shows the graphical representation of the ontology model for the motivating example. Concepts are depicted by round rectangles and *basic* is the special concept from which all other concepts inherit. The common ontology has three kinds of relationships: *inheritance*, *attribute* and *modifier*. Inheritance and attributes are two classic relationships originated from object-oriented modeling. For instance, concept stockPrice inherits from concept moneyValue, since stockPrice represents the money value of a company's stock. Attributes are the structural relationships between two concepts and depicted by solid arrows in Figure 2. For example, attribute quotedOn indicates that each object of stockPrice is quoted on a certain date.

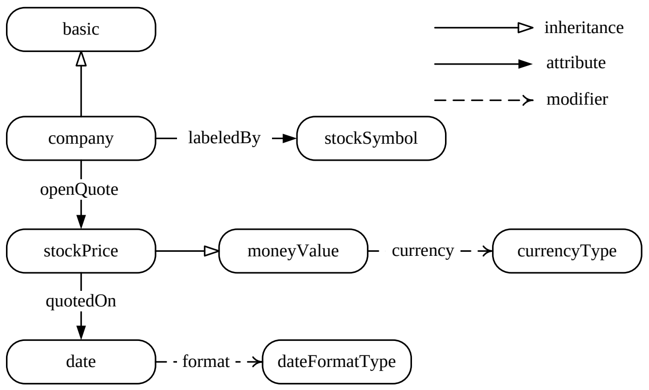


Figure 2. Common ontology.

In practice, it is usually straightforward for composition developers to find and define generic concepts among multiple independent services. For example, *S1* has a data type of *dateofIPO* and *S2* has data type of *dateofQuote* (see Table 1). Both data types correspond to the same concept date of the ontology. However, data instances of *dateofIPO* should be interpreted by date format "dd-mm-yyyy", while that of *dateofQuote* should be interpreted by "mm/dd/yyyy". Also, different currencies should be considered when interpreting data instances of the data types of *openingPrice* and *dailyOpenPrice*. That means data instances of a concept have to be interpreted according to certain assumptions of the corresponding services. To accommodate different data interpretations of a generic concept, a construct *modifier* is introduced to modify (i.e., interpret) the generic concept, so that the concept can have multiple specializations when it is associated with different services. In other words, modifiers are a special kind of attributes used to capture the additional information (which we call *context*) that affects data interpretation. Each modifier indicates a dimension of the specializations of a generic concept[4]. Also, a modifier can be inherited by a sub-concept from its super concept. Modifiers are depicted by dashed arrows in Figure 2. For example, concept date has a modifier format which indicates a dimension of the specific interpretations of its data instances. Data instances of concept date should be interpreted according to the values of dataFormatType which may take different values, e.g., "dd-mm-yyyy" and "mm/dd/yyyy". Similarly, the data instances of concept moneyValue may be interpreted in different currencies, as moneyValue has a modifier currency pointing to concept currencyType that may take different values, e.g., "GBP" and "USD". And concept stockPrice inherits the modifier currency from its super-concept moneyValue, so data instances of concept stockPrice may also be interpreted in "GBP" or "USD". In our approach, such different values associated with modifiers for data interpretation are grouped into multiple collections to define different contexts. The *context* refers to a collection of specializations of all modifiers in the common ontology. In our example, the U.K. context specifies date format to be "dd-mm-yyyy" and currency to be "GBP", while the U.S. context specifies the two modifiers to have values of "mm/dd/yyyy" and "USD", respectively. Thus, *CS* and *S1* are in the U.K. context, while *S2* is in the U.S. context.

3.2 Conversions for Context Differences

Context differences, once detected, need to be reconciled using conversions so as to convert the exchanged data from the source value *vs* to the target value *vt*. A conversion is defined for each modifier between two different modifier values. The general form of conversions is given below:

```
cvt(C, m, ctxt_s, ctxt_t, mvs, mvt, vs, vt)
```

Herein, *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 target context *ctxt_t*, respectively. *vs*, *vt* are the actual data values of *C* interpreted in *ctxt_s* and *ctxt_t*, respectively. According to the two modifiers of the common ontology, we need to define two conversions: cvt_{format}, and cvt_{currency}. For Web services composition, there are two methods to implement the conversions: XPath functions, and external Web services.

For certain simple cases, conversions can be specified using XPath functions. We adopt XPath functions, because the BPEL specification [3] and most BPEL engines (e.g., ActiveBPEL[5]) support XPath 1.0. For example, the conversion cvt_{format} for converting date formats from "dd-mm-yyyy" to "mm/dd/yyyy" can be implemented using the following XPath function and encapsulated as a custom function *cvtFormatUKtoUS* for further reuse, i.e., *Vt* = *cvtFormatUKtoUS* (*Vs*), as shown in Figure 3.

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

Figure 3. Conversion cvt_{format} for converting date formats from "dd-mm-yyyy" to "mm/dd/yyyy".

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

Figure 4. WSDL template for the external Web service.

In case XPath functions cannot address certain complex conversions, conversions have to be performed by external Web services. For example, it is needed to invoke an external currency exchange service (e.g., DOTSCurrencyExchange[6]) as the conversion cvt_{currency}. The external service translates the money value *vs* in

[4] There might be multiple dimensions involved – e.g., prices might differ in both currency and scale factor.

[5] http://www.activevos.com/community-open-source.php

[6] See http://ws2.serviceobjects.net/ce/CurrencyExchange.asmx?WSDL

USD to the money value *vt* in GBP. Figure 4 gives the WSDL template that developers can use to discover an appropriate service for the conversion. Note that the WSDL template is specified in WSDL 1.1, so developers who use WSDL 2.0 may produce a slightly different WSDL template.

4. SEMANTIC ANNOTATION

WSDL describes Web services at a syntactic level. To achieve the vision of SWSs, semantic annotation in a WSDL file is widely used to establish correspondence between the data elements in the WSDL and the concepts in a semantic model [7, 8]. Annotations can be done using the W3C standard, Semantic Annotation for WSDL and XML Schema (SAWSDL) [12]. SAWSDL itself doesn't provide any explicit semantics or enforce any language for expressing semantics, but it enables developers to annotate the syntactic WSDL descriptions with pointers to semantic concepts (identified via URIs) [15, 16]. In our work, we adopt SAWSDL to annotate WSDL descriptions.

SAWSDL provides an extension attribute *modelReference* for specifying the correspondence between WSDL components (e.g., data/element types, input and output messages) and semantic concepts in the ontology. We propose to use *modelReference* in two ways for context annotation: (1) Global context annotation: we allow the <wsdl:definitions>[7] element of the WSDL specification to have the attribute *modelReference* and use it to indicate that all data elements in the WSDL file subscribe to the context identified by 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 attribute *modelReference* to have an additional URI value to indicate the context of the data element. Note that multiple values are allowed by the SAWSDL standard which states that "the value of the *modelReference* attribute is a set of zero or more URIs, separated by whitespaces, that identify concepts in a semantic model" [12]. Thus, both global and local context annotations comply with the SAWSDL standard.

Global context annotation allows the developer to succinctly declare the context for all elements in a WSDL file, while the local context annotation provides a mechanism for certain elements to have their contexts different from the globally declared context. This "overriding" capability can be useful in case a small number of elements in a WSDL have contexts different from the context of the other elements.

```
<wsdl:definitions targetNamespace="http://stockQuote.coin.mit"
   xmlns:stkOntology="http://stockQuote.coin.mit/ontologies/stockOntology#"
   xmlns:sawsdl="http://www.w3.org/ns/sawsdl" …
  sawsdl:modelReference="stkOntology#ctxtUK">
<wsdl:types>
<schema elementFormDefault="qualified"
   targetNamespace="http://stockQuote.coin.mit"
   xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="getDateofIPO">
    <complexType>
      <sequence>
        <element name="tickerSymbol" type="xsd:string"
     sawsdl:modelReference="stkOntology#stockSymbol"/>
      </sequence>
    </complexType>
  </element>
  <element name="getDateofIPOResponse">
    <complexType>
      <sequence>
        <element name="getDateofIPOReturn" type="impl:IPOBean"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="IPOBean">
    <sequence>
      <element name="dateofIPO" nillable="true" type="xsd:string"
sawsdl:modelReference="stkOntology#date stkOntology#ctxtUK"/>
      <element name="tickerSymbol" nillable="true" type="xsd:string"
    sawsdl:modelReference="stkOntology#stockSymbol"/>
    </sequence>
  </complexType>
</schema>
</wsdl:type>
<wsdl:message name="getDateofIPORequest">
  <wsdl:part element="impl:getDateofIPO" name="parameters"/>
</wsdl:message>
<wsdl:message name="getDateofIPOResponse">
  <wsdl:part element="impl:getDateofIPOResponse" name="parameters"/>
</wsdl:message>
<wsdl:portType name="StockIPO">
  <wsdl:operation name="getDateofIPO">
   <wsdl:input message="impl:getDateofIPORequest"
                 name="getDateofIPORequest"/>
   <wsdl:output message="impl:getDateofIPOResponse"
                name="getDateofIPOResponse"/>
  </wsdl:operation>
</wsdl:portType>
</wsdl:definitions>
```

List 1. Annotated WSDL description of *S1* using global and local context annotations.

List 1 presents the annotated WSDL description of *S1* (i.e., *StockIPOWS*) in which the annotations are highlighted in bold. Each leaf data type element of *S1* uses *modelReference* to point to its corresponding concept. For example, the elements of *tickerSymbol* and *dateofIPO* point to concepts stockSymbol and date in the ontology (see Figure 2), respectively. As discussed in Section 3.1, *S1* uses the U.K. context: date format "dd-mm-yyyy" and currency "GBP". The *modelReference* attribute of element <wsdl:definitions> has the value "stkOntology#ctxtUK", which is the URI of the U.K. context defined in the common ontology. This annotation indicates that all messages/data of *S1* should be interpreted in the U.K. context. Also, each leaf data type element in List 1 is annotated using *modelReference*. Each *modelReference* attribute of the leaf data type element has one value or two values separated by a whitespace. When there is only one value, it is the URI of the concept to which the data element corresponds. When there are two values, the former value is the URI of the concept and the latter value is the URI of the context in which the element is interpreted. For illustration purposes, both global and local context annotations are used in List 1. In fact, the local annotation in List 1 is unnecessary because it does not override the global context for different contexts.

5. COMPOSITION REPRESENTATION AND MEDIATION

Context conflicts, which can be a source of data quality problems, can occur when a message (i.e., data) as output of the upstream

[7] <wsdl:definitions> is a WSDL element of WSDL 1.1. For WSDL 2.0, the corresponding element is <wsdl:definition>.

service with one context is transferred and consumed as input of the downstream service with another context, even if the schema declaration of the message defined in both services are the same. In the following sections, we assume that service compositions are defined in BPEL and the descriptions of component and composite services are specified in WSDL 1.1.

In this section, we first briefly introduce a formal description language, LOTOS NT [13], which is compatible with the main concepts of BPEL specification and provides the capability of formally analyzing both static and dynamic aspects of Web services composition. Then, we describe the correspondence and translation between WSDL/BPEL and LOTOS NT. After that, we introduce the procedures that automatically detect context conflicts in the composition and incorporate necessary conversions into the mediated composition to reconcile the detected conflicts.

5.1 LOTOS NT in a Nutshell

LOTOS, as a kind of Process Algebras (PAs), can describe and analyze both static and dynamic aspects of distributed software systems. It consists of two sub-languages: the static part is dedicated to the description of data structures, which is based on algebraic specification language ACT ONE [17] for abstract data types; the dynamic part is based on the process algebra approach for concurrency, like CCS [18] and CSP [19]. Recently, LOTOS has attracted considerable attention for describing, composing and reasoning with Web services at an abstract level [20-23]. LOTOS NT removes a few undesirable characteristics of LOTOS and follows the main concepts of E-LOTOS[8] (for Extended-LOTOS) which has become an ISO specification language and is supported by the TRAIAN compiler [13].

We chose LOTOS NT as the abstract formalism of our approach because: (1) LOTOS NT provides rich expressiveness for defining complex data structures and data handling so that we can describe and analyze the messages (e.g., data types) defined in WSDL/BPEL, which usually have more than one part/element. Other PAs roughly describe the messages as tokens, so different parts/elements of the message cannot be distinguished [20]; (2) LOTOS NT is a strongly typed and fully imperative language which is necessary for description safety and analysis of data types. For example, complex types can be defined using type constructors with existing types, and variables, once initialized, can store the data and be accessed by read and write operations; (3) LOTOS NT allows the modular description of systems and description reusability which is compatible with WSDL/BPEL specifications.

5.2 Translating WSDL/BPEL to LOTOS NT

We identify the correspondence between WSDL/BPEL and LOTOS NT. Specifically, the static aspect of service composition (e.g., data types, variables, etc.) is mapped to type declarations of LOTOS NT, and the dynamic aspect (i.e., process descriptions) is mapped to process declarations of LOTOS NT.

5.2.1 Translation of Static Aspect

WSDL descriptions of the composite and component Web services (e.g., data types, messages and operations) as well as descriptions of variables and partner links in BPEL are considered as the static aspect of a service composition, which is translated to type declarations of LOTOS NT. Type declarations are used to define new types, regardless whether they consist of several subtypes. The type declarations in LOTOS NT follow the general approach of constructed types in functional languages, which use the special operation *constructor* [13].

Take the WSDL description of data types as an example. The annotated WSDL description of *S1* (see List 1), defines a complex type *IPOBean*. The definition of *IPOBean* is translated to the type declaration of LOTOS NT, as shown in List 2. To preserve the WSDL complex type "*IPOBean*", we define a LOTOS NT type with the name "*IPOBeanComplexType*". The suffix "*ComplexType*" added at the end is used to keep track of the fact that this type is originally a complex type in WSDL. A constructor, also named as "*IPOBeanComplexType*", is declared with two parameters, *tickerSymbol* and *dateofIPO*. This constructor is used to initialize type *IPOBeanComplexType*. In List 2, "*string*"is a predefined type of LOTOS NT which indicates that data types of *tickerSymbol* and *dateofIPO* are originally "*xsd:string*" in the WSDL description. The annotations using SAWSDL are captured as *comments* in LOTOS NT, i.e., the text from "*(**" to "**)*". Using these comments, the reasoning engine is able to know the corresponding concept and context of a data type and check possible context conflicts, as presented later.

```
(* @ tickerSymbol:mdlRef="stkOntology#stockSymbol"
 * @ dateofIPO:mdlRef="stkOntology#date stkOntology#ctxtUK"
 *)
type IPOBeanComplexType is
   IPOBeanComplexType (tickerSymbol:string, dateofIPO:string)
end type
```

List 2. Snippet of type declaration in LOTOS NT.

Other static components of WSDL/BPEL (e.g., simple/complex types, messages, port types, operations and partner link types in WSDL, and partner links and variables in BPEL) can also be translated to type declarations of LOTOS NT in a similar way.

5.2.2 Translation of Dynamic Aspect

LOTOS NT provides a means for describing concurrent (parallel) evolution of software systems and their communication actions. The actions must be communicated by the rendezvous on communication points called *gates*. LOTOS NT allows us to name an action using a process definition [13]. The process description (i.e., workflow logic) and activities in BPEL can be compatibly translated to the action/process declaration of LOTOS NT. For example, several interaction activities in BPEL (e.g., <*receive*>, <*reply*>, <*invoke*>, and <*onMessage*> of <*pick*>) can be translated to the communication action in LOTOS NT.

We identify several mappings between the constructs of BPEL process and actions/processes of LOTOS NT, as presented in Table 2. Consider the first two mappings where *v* is a variable and *c* is a gate through which the variable *v* is exchanged. With the identifier of a gate, the attributes of *partner link*, *port type* and *operation* in the BPEL specification can be derived. Action "*c*?*v*" and "*c*!*v*" denote a message (i.e., data represented by a variable) *v* is received and emitted over the gate *c*, respectively. The LOTOS NT operator ";" denotes the sequential order of two

[8] http://www.inrialpes.fr/vasy/pub/elotos/

activities/actions in the workflow logic. In the BPEL specification, variables, which can be shared by activities within a scope or the whole process, are defined to store the exchanged data. The activity <*assign*> and its element <*copy*> are used to explicitly specify data transfers. In LOTOS NT, explicit data transfers are represented using the assignment operator ":=". A comprehensive set of mappings between BPEL and LOTOS can be found in [20, 24]. According to the mappings given in Table 2, the naïve BPEL composition of the motivating example, illustrated in Figure 1, can be translated to a LOTOS NT process, known as naïve LOTOS NT process. The naïve LOTOS NT process is given in List 3.

Table 2. Mappings between BPEL and LOTOS NT

BPEL	LOTOS NT
<receive variable="v" …/>	(*c*?*v*)
<reply variable="v" …/>	(*c*!*v*)
<invoke inputVariable="v1" outputVariable="v2"…/>	(*c*!*v1* ; *c*?*v2*)
<assign …> <copy> <from variable="v1"> <to variable="v2"/> </copy> <copy> <from variable="v3"> <to variable="v4"/> </copy> </assign>	(*v2* := *v1*; *v4* := *v3*)
<sequence …> <…activity1…/> <…activity2…/> </sequence>	(*action1* ; *action2*)
<flow …> <…activity1…/> <…activity2…/> </flow>	(*action1* [] *action2*)
<pick …> <onMessage variable="v1"…> <…activity1…> </onMessage> <onMessage variable="v2"…> <…activity2…> </onMessage> </pick>	(*c1*?*v1* ; *action1* [] *c2*?*v2* ; *action2*)

5.3 Detecting Context Conflicts

Context conflicts occur within data transfers where a message in a process is provided by one action (*source action*) and then consumed (i.e., received) by another action (*target action*). Specifically, context conflicts occur in case data interpretations of source and target actions of a data transfer are different. Thus, each data transfer in a process needs to be checked in order to detect all possible context conflicts.

A process may contain two types of data transfers: *explicit* and *implicit*. Explicit data transfers are easily identified, as they are indicated using the assignment operator ":=" in LOTOS NT. For example, the data transfers of *tickerSymbol* and *dateofIPO* between the invocations of *S1* and *S2* are explicit, because the LOTOS NT process, as shown in List 3, contains an assignment statement. Implicit data transfers occur when the data is transferred through a shared variable but its reception and emission gates are different. Thus, implicit data transfers can be identified through checking the shared variables. As shown in List 3, variable *getDateofIPO*, in which *tickerSymbol* is defined as its element, is initialized after the data is received as input of *CS* over gate *QuoteOpeningPricePL*. Then, *getDateofIPO* is directly used as the input variable to invoke *S1*, so the data of *getDateofIPO* is consumed by *S1*. In this case, the data transfer of *tickerSymbol* carried in *getDateofIPO* from the input of *CS* to the input of *S1* is implicit. Similarly, the data transfer of *openingPrice* from the output of *S2* to the output of *CS*, through variable *getDailyOpenPriceResponse*, is also implicit. In other cases implicit data transfers may also occur between two invocations of component services.

Once a data transfer is identified, its reception and emission gates are checked to identify corresponding source and target partners (i.e., services) using the information of *partner link*, *port type* and *operation* derived from the gates. Then, we compare the contexts of the source and target services to check whether the contexts are consistent. Take the explicit data transfer of *dateofIPO* as an example. Its source variable is *getDateofIPOResponse*, whose data type is defined in the LOTOS NT module *StockIPO* corresponding to the WSDL description of *S1*. Its target variable is *getDailyOpenPrice*, whose data type is defined in the LOTOS module *HistoricalStockQuote* corresponding to the WSDL description of *S2*. Accordingly, its source and target services are *S1*, *S2*, respectively. According to the semantic annotation presented in Section 4 and Section 5.2.1, both *dateofIPO* and *dateofQuote* are annotated to point to concept date in the ontology (see Figure 2). By reasoning with the ontology enriched with contexts, we know that concept date has a modifier format with two different values: "dd-mm-yyyy" in the U.K. context and "mm/dd/yyyy" in the U.S. context. A context conflict is thus detected within the data transfer of *dateofIPO*. After checking all the data transfers in the LOTOS NT process shown in List 3, we detect another context conflict in the implicit data transfer of *dailyOpenPrice* (through variable *getDailyOpenPriceResponese*).

5.4 Incorporating Conversions into LOTOS NT Process

Once a context conflict is detected within a data transfer, necessary conversion needs to be identified from the predefined conversions (see Section 3.2) and incorporated into the data transfer for converting data between different contexts.

In case that the detected context conflict occurs in an implicit data transfer, the data transfer needs to be made explicit. As shown in Section 5.3, a context conflict occurs within the implicit data transfer of *dailyOpenPrice* because of different currencies used to interpret the stock prices: "USD" for *S2* and "GBP" for *CS*. To make it explicit, a new variable, *getOpeningPriceResponse*, is automatically declared using the same data type of variable *getDailyOpenPriceResponse*. Then, an assignment statement is inserted in the data transfer to initialize variable *getOpeningPriceResponse* with the value of variable *getDailyOpenPriceResponse*. Also, the output variable of the emission gate from *CS* is changed as *getOpeningPriceResponse*.

```
 (* Definition of a module with three module importation
 * directives which corresponds to the WSDL descriptions
 * of the composite and two component services. *)
module QuoteOpeningPriceProcess (QuoteOpeningPrice,
  StockIPO, HistoricalStockQuote) is

(* Each partner link corresponds to a communication gate with
 * which necessary information, such as partner link, port type,
 * operation and partner service, can be derived. *)
gate StockIPOPLGate is
  (StockIPOPLTPartnerLinkType) end gate
gate HistoricalStockQuotePLGate is
  (HistoricalStockQuotePLTPartnerLinkType) end gate
gate QuoteOpeningPricePLGate is
  (QuoteOpeningPricePLTPartnerLinkType)  end gate

(* Process declaration *)
process QuoteOpeningPriceProcess
  [StockIPOPL:StockIPOPLGate,
   HistoricalStockQuotePL:HistoricalStockQuotePLGate,
   QuoteOpeningpricePL:QuoteOpeningPricePLGate] () is


(* Declaration of variables that are used in the process *)
var getDateofIPO:getDateofIPOType,
    getDateofIPOResponse:getDateofIPOResponseType,
    getDailyOpenPrice:getDailyOpenPriceType,
   getDailyOpenPriceResponse:getDailyOpenPriceResponseType
in

(* Corresponding to <receive> of CS *)
QuoteOpeningPricePL(?quoteOpeningPriceRole
(getOpeningPriceInput(getOpeningPriceRequestMessage
(getDateofIPO)))) ;                                  Implicit data transfer
(* Corresponding to <invoke> of S1 *)
StockIPOPL(!stockIPOServiceRole(getDateofIPOInput
(getDateofIPORequestMessage(getDateofIPOType
(getDateofIPO)))) ;
StockIPOPL(?stockIPOServiceRole(getDateofIPOOutput
(getDateofIPOResponseMessage(getDateofIPOResponse)))) ;

(* Corresponding to <assign> *)                           Explicit data transfer
getDailyOpenPrice := getDailyOpenPriceType
(getDateofIPOResponse.getDateofIPOReturn.tickerSymbol,
 getDateofIPOResponse.getDateofIPOReturn.dateofIPO) ;

(* Corresponding to <invoke> of S2 *)
HistoricalStockQuotePL(!historicalStockQuoteServiceRole
(getDailyOpenPriceInput(getDailyOpenPriceRequestMessage
(getDailyOpenPrice)))) ;
HistoricalStockQuotePL (?historicalStockQuoteServiceRole
(getDailyOpenPriceOutput(getDailyOpenPriceResponseMessage
(getDailyOpenPriceResponse)))) ;

(* Corresponding to <reply> of CS *)                        Implicit data transfer
QuoteOpeningPricePL(!quoteOpeningPriceRole
(getOpeningPriceOutput(getOpeningPriceResponseMessage
(getDailyOpenPriceResponse))))

end var
end process
end module
```

List 3. LOTOS NT process translated from the naïve BPEL composition of the motivating example.

```
module QuoteOpeningPriceProcess (QuoteOpeningPrice,
  StockIPO, HistoricalStockQuote) is

gate StockIPOPLGate is (StockIPOPLTPartnerLinkType) end gate
gate HistoricalStockQuotePLGate is
(HistoricalStockQuotePLTPartnerLinkType) end gate
gate QuoteOpeningPricePLGate is
  (QuoteOpeningPricePLTPartnerLinkType)  end gate

(* A new gate is declared to communicate with
 * the external Web service for currency conversion. *)
gate CurrencyConvertorPLGate is
  (CurrencyConvertorPLTPartnerLinkType)  end gate

(* The module of currency convertor is imported. *)
process QuoteOpeningPriceProcess
  [StockIPOPL:StockIPOPLGate,
   HistoricalStockQuotePL:HistoricalStockQuotePLGate,
   CurrencyConvertorPL: CurrencyConvertorPLGate,
   QuoteOpeningpricePL:QuoteOpeningPricePLGate] () is

(* A new variable is declared in the process. *)
var getDateofIPO:getDateofIPOType,
   getDateofIPOResponse:getDateofIPOResponseType,
   getDailyOpenPrice:getDailyOpenPriceType,
   getDailyOpenPriceResponse:getDailyOpenPriceResponseType,
   getOpeningPriceResponse: getDailyOpenPriceResponseType
in

QuoteOpeningPricePL(?quoteOpeningPriceRole
(getOpeningPriceInput (getOpeningPriceRequestMessage
(getDateofIPO)))) ;

StockIPOPL(!stockIPOServiceRole (getDateofIPOInput
(getDateofIPORequestMessage (getDateofIPOType
(getDateofIPO.tickerSymbol))))) ;
StockIPOPL(?stockIPOServiceRole(getDateofIPOOutput
(getDateofIPOResponseMessage(getDateofIPOResponse)))) ;

(* The XPath function, which is encapsulated in
 * a custom function with the name cvtFormatUKtoUS,
 * is incorporated. *)                                  XPath function for conversion
getDailyOpenPrice := getDailyOpenPriceType
(getDateofIPOResponse.getDateofIPOReturn.tickerSymbol,
 cvtFormatUKtoUS (getDateofIPOResponse.
getDateofIPOReturn.dateofIPO) ) ;

HistoricalStockQuotePL(!historicalStockQuoteServiceRole
(getDailyOpenPriceInput (getDailyOpenPriceRequestMessage
(getDailyOpenPrice)))) ;
HistoricalStockQuotePL (?historicalStockQuoteServiceRole
(getDailyOpenPriceOutput(getDailyOpenPriceResponseMessage
(getDailyOpenPriceResponse)))) ;

(* Invocation statement of ES *)                        External service for conversion
CurrencyConvertorPL (!currencyConvertorServiceRole
(getPriceInGBPInput (getPriceInGBPRequestMessage
(getDailyOpenPriceResponse)))) ;
CurrencyConvertorPL (?currencyConvertorServiceRole
(getPriceInGBPOutput (getPriceInGBPResponseMessage
(getOpeningPriceResponse)))) ;

(* The output variable is changed. *)
QuoteOpeningPricePL(!quoteOpeningPriceRole
(getOpeningPriceOutput(getOpeningPriceResponseMessage
(getOpeningPriceResponse))))

end var
end process end module
```

List 4. Mediated LOTOS NT process with incorporated conversions.

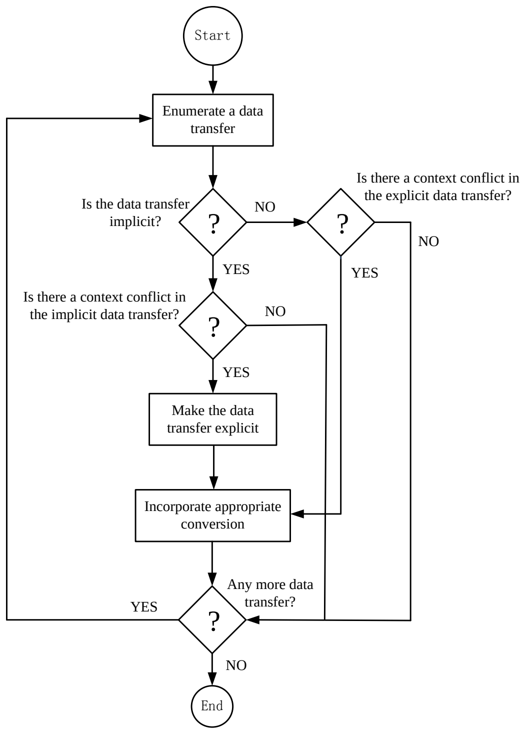

Start
Enumerate a data transfer
Is the data transfer implicit?
?
NO
Is there a context conflict in the explicit data transfer?
?
NO
YES
YES
Is there a context conflict in the implicit data transfer?
?
NO
YES
Make the data transfer explicit
Incorporate appropriate conversion
Any more data transfer?
YES
?
NO
End

Figure 5. Algorithm for reconciling context conflicts.

Now that all data transfers with context conflicts are explicit, necessary conversions need to be identified and inserted into these data transfers. Two context conflicts exist in the naïve composition of the motivating example: different data formats for interpreting concept date with modifier format, and different currencies for interpreting concept stockPrice with modifier currency. As presented in Section 3.2, two conversions are defined with modifier format and modifier currency, respectively - in our work the predefined conversions are maintained in the conversion repository. The conversion for modifier format (i.e., cvt_{format}), which converts date format from "dd-mm/-yyyy" to "mm/dd/yyyy", is defined as a custom function *cvtFormatUKtoUS* using XPath functions (see Figure 3). This conversion is inserted to the data transfer of *dateofIPO*, that is, function *cvtFormatUKtoUS* consumes variable *dateofIPO* and produces the date interpreted by "mm/dd/yyyy". The conversion for modifier currency (i.e., cvt_{currency}), which converts prices in "USD" to "GBP", is defined using an external service, denoted by *ES* for short. *ES* has an operation *getPriceInGBP* which consumes a price in USD and produces the equivalent price in GBP[9]. According to this conversion, an action for invoking the operation *getPriceInGBP* of *ES* needs to be inserted into the data transfer of *dailyOpenPrice*. In fact, the assignment statement created above is substituted by the invocation statement of operation *getPriceInGBP* in which the input and output variables are *getDailyOpenPriceResponse* and *getOpeningPriceResponse*, respectively. The mediated LOTOS NT process of the composition is given in List 4. The generated conversion code in the mediated LOTOS NT process is highlighted in bold.

The algorithm for reconciling context conflicts is illustrated in Figure 5. The algorithm enumerates and examines each data transfer of the LOTOS NT process. If a context conflict is detected, the algorithm identifies an appropriate conversion and incorporates it into the process. Eventually, the mediated LOTOS NT process without any context conflict is produced.

6. CONTEXT MEDIATION TOOL

A proof-of-concept prototype, *Context Mediation Tool* (*CMT*), has been developed to validate and demonstrate our approach. The WSDL descriptions of composite and component services (i.e., *CS*, *S1* and *S2*) are annotated with semantic references using *Radiant4Context* which is an Eclipse plug-in we have developed as an extension to Radiant[10], an open source project for semantic annotation. The common ontology enriched with contexts is defined using our COIN Model Application Editor[11], which is a lightweight Web-based tool for creating and editing ontologies and contexts in RDF/OWL. Atomic conversions between contexts are defined in a specification file. We assume that the U.K. developer, without being aware of semantic heterogeneity among the services, created the naïve BPEL composition of the motivating example using ActiveVOS Designer[12]. After consuming all these documents, *CMT* first translates the naïve BPEL process and the involved WSDL descriptions to the naïve LOTOS NT Process. Then, the reasoning engine implemented within *CMT* uses the algorithm to automatically detect context conflicts within the naïve LOTOS NT Process and incorporate appropriate conversions into the mediated LOTOS NT Process. Finally, *CMT* translates the mediated LOTOS NT Process back to the mediated BPEL process, according to the mappings between BPEL and LOTOS NT given in Table 2.

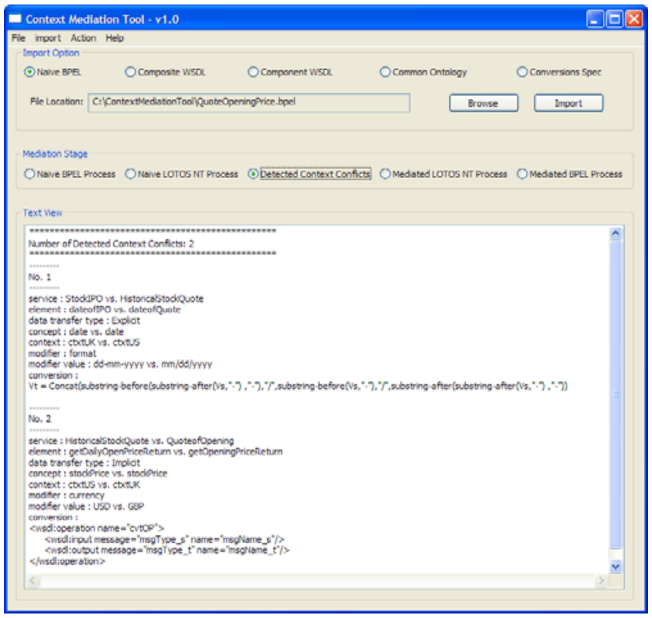
Figure 6 shows a snapshot of *CMT* which has three working sections. The first working section requires the user to import the involved documents, as described above, into a mediation project. To monitor the results of different steps of the mediation task, the second working section of *CMT*, *Mediation Stage*, allows the user to choose one of the five consecutive stages, including naïve BPEL process, naïve LOTOS NT process, detected context conflicts, mediated LOTOS NT process and mediated BPEL process. These five stages reveal the intermediate and final results that our approach produces while handling context differences in the service composition. Particularly, Figure 6 shows the stage *Detected Context Conflicts* where two detected context conflicts in the naïve composition of the motivating example are listed. Detailed information of each context conflict is displayed, such as corresponding services, messages/data elements, concepts, contexts, modifiers, modifier values, and conversions. The detailed information clearly explains what the context conflict is and where it occurs in the composition. It is worth noting that *CMT* can produce not only the mediated LOTOS NT process (e.g., List 4), but also the mediated BPEL process.

[9] General purpose conversions, such as that between any pair of currencies, can also be used, but discussion of this point is beyond the space constraints of this paper.

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

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

[12] http://www.activevos.com/activevos-enterprise-download-test.php



Import Option: [✓ Naive BPEL] Composite WSDL Component WSDL Common Ontology Conversions Spec

File Location: C:\ContextMediationTool\QuoteOpeningPrice.bpel

Mediation Stage: Naive BPEL Process Naive LOTOS NT Process [✓ Detected Context Conflicts] Mediated LOTOS NT Process Mediated BPEL Process

```
*************************************************
Number of Detected Context Conflicts: 2
*************************************************
---------
No. 1
---------
service : StockIPO vs. HistoricalStockQuote
element : dateofIPO vs. dateofQuote
data transfer type : Explicit
concept : date vs. date
context : ctxtUK vs. ctxtUS
modifier : format
modifier value : dd-mm-yyyy vs. mm/dd/yyyy
conversion :
Vt = Concat(substring-before(substring-after(Vs,"-") ,"-"),"/",substring-before(Vs,"-"),"/",substring-after(substring-after(Vs,"-") ,"-"))

---------
No. 2
---------
service : HistoricalStockQuote vs. QuoteofOpening
element : getDailyOpenPriceReturn vs. getOpeningPriceReturn
data transfer type : Implicit
concept : stockPrice vs. stockPrice
context : ctxtUS vs. ctxtUK
modifier : currency
modifier value : USD vs. GBP
conversion :
<wsdl:operation name="cvtOP">
    <wsdl:input message="msgType_s" name="msgName_s"/>
    <wsdl:output message="msgType_t" name="msgName_t"/>
</wsdl:operation>
```

Figure 6. Snapshot of *CMT* at stage Detected Context Conflicts.

7. RELATED WORK AND DISCUSSION

The challenges of heterogeneous data semantics among Web services have been discussed in [25]. In the literature, however, only a few approaches have been developed to handle the data quality problems of Web services composition that result from heterogeneous data semantics. Based on METEOR-S, semantic annotation and mapping are proposed to deal with schema structural heterogeneity of the exchanged messages in service composition [25, 26]. The approach in [27] use data transformation rules, which are specified in a Prolog notation, to convert the exchanged messages/data from the upstream service to the downstream one. The approach can only deal with a pair of interoperating services rather than a composition consisting of several services. The work in [28, 29] proposes a set of mapping relations which can semi-automatically compose Web services. However, the mapping relations between two services have to be specified by developers manually to generate the composition. Also, this work is restricted to simple composition scenarios in which only two services are integrated, since mapping relations are usually defined between two services. In practice, however, service composition may involve multiple services and have complicated workflow logic. To the best of our knowledge, the work in [30-32], which is also drawn on the original COIN strategy, is most related to this paper. However, our approach is different from that work in several aspects: (1) Their work only considers component services but ignores the composite service, while context differences among both composite and component services can be handled using our solution; (2) The WSDL description is directly annotated with the context definition - modifier values defining the contexts are enumerated as the extension of the WSDL elements. In case of a large number of modifier values, it is difficult to enumerate and maintain so many modifier values in the WSDL elements. Differently, we specify modifier values in the ontology definition separate from WSDL descriptions and propose the flexible, standard-compliant mechanism for annotating WSDL descriptions using SAWSDL; (3) Only external services are considered in their work to reconcile context differences, while both XPath functions and external services are proposed in our reconciliation solution. In certain cases, it is applicable and more efficient to use XPath functions as conversions, such as that for date formats in different styles and numbers in different scale factors; (4) Only context conflicts between the <*invoke*> activities in the BPEL composition are considered in their work, while context conflicts between all interaction activities (e.g., <*receive*>, <*reply*>, <*invoke*> and <*onMessage*>) can be handled using our solution; (5) Their work performs the detection of context conflicts in verbose BPEL and WSDL files, while we exploit and extend LOTOS NT so that service compositions in BPEL/WSDL are translated to LOTOS NT and analyzed based on the formalism.

8. CONCLUSION

With the increasing opportunity to access and integrate data from diverse Web services, data quality issues of service composition have been drawing more attention in recent years. Many data quality problems are actually data misinterpretation problems which result from heterogeneous data semantics among Web services. In this paper, we introduce a mechanism for composition developers to flexibly define a small set of generic concepts among the services involved in a composition as a common ontology and use contexts to describe the specializations of the generic concepts. We have developed a flexible, standard-compliant method to annotate the WSDL descriptions of Web services with contexts (i.e., using SAWSDL). Given a naïve BPEL composition unaware of context differences, our approach can automatically produce a mediated BPEL composition that incorporates necessary conversions to reconcile context conflicts. The approach alleviates the reconciliation task for addressing heterogeneous data semantics among Web services and resolves many data quality problems for their composition.

In the future, we plan to enhance the ability of our approach to handle more complex composition scenarios, such as complex data types of WSDL descriptions, complex definitions of contexts, complex workflow logics of the BPEL composition, and complex structural differences among the exchanged messages.

9. ACKNOWLEDGMENTS

The authors thank Hubert Garavel and Damien Thivolle, from INRIA Rhone-Alpes / VASY, France, for their valuable discussion about the translation of LOTOS NT. Also, the authors thank Frank Manola, Independent Consultant, Wilmington, MA, USA, for his valuable comments. This work was partially supported by the MIT Sloan China Management Education Project and grants from the National Natural Science Foundation of China (No. 60674080 and No. 60704027) and the National High-Tech R&D (863) Plan of China (No. 2007AA04Z150).

10. REFERENCES

[1] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and managing Web services: issues, solutions, and directions. *The VLDB Journal*, 17(3): 537-572, 2008.

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

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

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

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

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

[7] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. In *Proc. of the 1st Intl. Conf. on Web Services (ICWS'03)*, Las Vegas, USA, 395–401, 2003.

[8] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. Meteor-s web service annotation framework. In *Proc. of the 13th Intl. Conf. on World Wide Web (WWW'04)*, New York City, NY, USA, 553-562, 2004.

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

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

[11] X. Li, S. Madnick, H. Zhu, and Y. Fan. An Innovative Approach to Composing Web Services with Context Heterogeneity. In *Proc. of the 7th Intl. Conf. on Web Services (ICWS'09)*, Los Angeles, CA, USA, 695-702, 2009.

[12] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. *W3C Recommendation*, August, 2007.

[13] M. Sighireanu. LOTOS NT User Manual (Release 2.6). *Technical Report*, Projet VASY, Inria Rhne-Alpes, Montbonnot Saint-Martin, France, February 2008. Available at http://www.inrialpes.fr/vasy.

[14] H. Garavel, R. Mateescu, F. Lang, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In *Proc. of the 19th Intl. Conf. on Computer Aided Verification (CAV'07)*, Berlin, Germany, 158-163, 2007.

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

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

[17] J. de Meer, R. Roth, and S. Vuong. Introduction to algebraic specifications based on the language ACT ONE. *Computer Networks and ISDN Systems*, 23(5): 363-392, 1992.

[18] R. Milner. *Communication and concurrency*: Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1989.

[19] C. A. R. Hoare. *Communicating sequential processes*: Prentice-Hall International, 1985.

[20] A. Ferrara. Web services: a process algebra approach. In *Proc. of the 2nd Intl. Conf. on Service-Oriented Computing (ICSOC'04))*, New York City, NY, USA, 242-251, 2004.

[21] G. Salaün, A. Ferrara, and A. Chirichiello. Negotiation among Web Services Using LOTOS/CADP. In *Proc. of the 1st European Conf. on Web Services (ECOWS'04)*, Erfurt, Germany, 198-212, 2004.

[22] A. Chirichiello and G. Salaün. Encoding abstract descriptions into executable web services: Towards a formal development. In *Proc. of the 3rd IEEE/WIC/ACM Intl. Conf. on Web Intelligence (WI'05)*, 457-463, 2005.

[23] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. In *Prof. of the 6th Intl. Conf. on Service-Oriented Computing (ICSOC'08)*, Sydney, Australia, 84-99, 2008.

[24] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on Web Services using Process Algebra. *Intl. Journal of Business Process Integration and Management*, 1(2): 116-128, 2006.

[25] M. Nagarajan, K. Verma, A. P. Sheth, J. Miller, and J. Lathem. Semantic Interoperability of Web Services - Challenges and Experiences. In *Proc. of the 4th Intl. Conf. on Web Services (ICWS'04)*, Chicago, USA, 373-382, 2006.

[26] Z. Wu, A. Ranabahu, K. Gomadam, A. P. Sheth, and J. A. Miller. Automatic Composition of Semantic Web Services using Process and Data Mediation. In *Proc. of the 9th Intl. Conf. on Enterprise Information Systems (ICEIS'07)*, Funchal, Portugal, 453-461, 2007.

[27] B. Spencer and S. Liu. Inferring Data Transformation Rules to Integrate Semantic Web Services. In *Proc. of the 3rd Intl. Semantic Web Conference*, Japan, 456-470, 2004.

[28] D. Gagne, M. Sabbouh, S. Bennett, and S. Powers. Using Data Semantics to Enable Automatic Composition of Web Services. In *Proc. of the 4th IEEE Intl. Conf. on Services Computing (ICSOC'06)*, 438-444, 2006.

[29] M. Sabbouh, J. L. Higginson, C. Wan, and S. R. Bennett. Using Mapping Relations to Semi Automatically Compose Web Services. In *Proc. of IEEE Congress on Services - Part I (SERVICES'08)*, 211-218, 2008.

[30] M. Mrissa, C. Ghedira, D. Benslimane, and Z. Maamar. A Context Model for Semantic Mediation in Web Services Composition. In *Proc. of the 25th Intl. Conf. on Conceptual Modeling (ER'06)*, Tucson, Arizona, USA, 12-25, 2006.

[31] M. Mrissa, C. Ghedira, D. Benslimane, and Z. Maamar. Context and Semantic Composition of Web Services. In *Proc. of the 17th Intl. Conf. on Database and Expert Systems (DEXA'06)*, Krakow, Poland, 266-275, 2006.

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