An Approach to Composing Web Services with Context Heterogeneity

Xitong Li[1,2,†], Stuart Madnick[2], Hongwei Zhu[3], and Yushun Fan[1]
[1]*Department of Automation, Tsinghua University, Beijing 100084, P.R. China*
lxt04@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn
[2]*MIT Sloan School of Management, Cambridge, MA 02142, USA*
{xitongli, smadnick}@mit.edu
[3]*College of Business and Public Administration, Old Dominion University, Norfolk, VA 23529, USA*
hzhu@odu.edu

Abstract

The potential benefits of Web services composition heavily rely on semantic interoperability, i.e., the ability to exchange data meaningfully amongst Web services. Context heterogeneity, which refers to different implicit assumptions about interpreting the exchanged data, hampers the automatic composition of Web services. However, existing initiatives of Semantic Web Services (SWSs) often ignore context heterogeneity. In this paper, we introduce an approach to address this issue. The contexts of the involved Web services are defined in a lightweight ontology and their WSDL descriptions are annotated by an extension of a W3C standard, i.e., Semantic Annotation for WSDL and XML Schema (SAWSDL). The composition of Web services is described using BPEL specification. Given a BPEL file that ignores context heterogeneity, the approach automatically detects all context differences among the involved services, and reconciles them by producing a mediated BPEL file that incorporates necessary conversions using XPath functions and/or Web services.

1. Introduction

Web services composition addresses the situation in which a business need cannot be accomplished by a single Web service, whereas a composite Web service consisting of a combination of multiple independent services could satisfy the need. To fully realize the potential benefits of service composition, we have to deal with various issues of semantic interoperability [1], i.e., the ability to exchange data meaningfully amongst Web services. Unfortunately, the basic protocol standards for Web services (e.g., WSDL, BPEL) widely ignore data semantics. Existing initiatives, such as OWL-S [2], WSMF/WSMO [3, 4] and METEOR-S [5, 6], have developed languages and frameworks to explicitly add semantics into the descriptions of Web services. The produced services are often referred to as Semantic Web Services (SWSs) [7, 8].

However, these efforts also have not adequately addressed the context of the exchanged data between interoperating Web services. By context, we refer to the collection of implicit assumptions about how the exchanged data should be interpreted [9, 10]. 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 the contexts of the two services to compute the distance the car can go with a full tank of gas. As a result, context heterogeneity can hamper the correct interoperability and composition of Web services. The problem of context heterogeneity grows in complexity when composing multiple Web services developed by independent providers that may be distributed throughout the world and have disparate implicit assumptions of data interpretation. By searching the public service registry [11], we investigated a number of real-world Web services and composition scenarios, and found that context heterogeneity widely exists among the services, such as different scale factors of numbers, currencies of prices, different representations of time and positions. Thus, effective methods for reconciling context heterogeneity are demanded to facilitate the automatic composition of semantic Web services.

In this paper, we introduce an approach to automatic detection and reconciliation of context heterogeneity amongst Web services. It is based on an extension to the Context Interchange (COIN) strategy for semantic interoperability amongst multiple data sources [9, 10]. The approach requires that composition developers use a common ontology model and declaratively describe the contexts of each component service as well as the

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

composite service. In addition, the conversions for reconciling context differences, if detected, should also be provided declaratively. With these descriptions in place, the approach can detect context differences of all involved Web services and produce a mediated BPEL file that incorporates the necessary conversions to reconcile the context differences.

The rest of this paper is structured as follows. Section 2 presents the motivating example. Section 3 presents the mechanisms for representing contexts of Web services and necessary conversions for reconciliation. Section 4 describes the mediation procedure for automatic detection and reconciliation of context differences in Web services composition. Section 5 presents the validation. Section 6 reviews related work. Section 7 gives a brief conclusion.

2. Motivating example

Let us consider a scenario that an U.K. developer wants to develop a Web service, namely *QuoteofOpeningWS* (denoted as *CS*), to obtain the opening price of a company's stock on the first trading day. *CS* is intended for U.K. analysts to monitor the U.S. stock market. To provide the service and meet the business need, the developer tries to implement it by composing existing Web services. After searching a public service registry [11], he finds two component Web services that could be combined to develop *CS*, i.e., *StockIPOWS* and *HistoricalStockQuoteWS*, denoted as *S1* and *S2* respectively. *S1* provides the functionality for querying the IPO[1] 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 *getOpenPrice* of *S2* returns the open price[2] of a company's stock on a given date. The signatures of these Web services are summarized in Table 1. For simplicity, we do not include the details of WSDL descriptions and assume the low-level messages of these Web services have compatible data types (e.g., string, double, etc.).

It appears that *CS* can be composed by feeding the output of the operation *getDateofIPO* of *S1* as the input to the operation *getOpenPrice* of *S2*. This composition can be specified in BPEL which has been proposed by OASIS as the industry standard. The composition is illustratively presented in Figure 1.

[1] Initial Public Offering (IPO) is when a company issues common stock to the public for the first time.
[2] Note that in this paper the opening price specifically refers to the open price of a company's stock on the first trading day. Opening price is the important information for the stock investors to analyze the performance of a company's stock.

Table 1. Signatures of involved Web services

	Operation	Input	Output
CS	*getOpeningPrice*	tickerSymbol	openingPrice
S1	*getDateofIPO*	tickerSymbol	dateofIPO, tickerSymbol
S2	*getOpenPrice*	dateofQuote tickerSymbol	openPrice

However, the composition does not consider the contexts of these services: *CS* and *S1* have the U.K. context, where the date format is "dd-mm-yyyy" and price currency is "GBP", while *S2* has the U.S. context, where the date format is "mm/dd/yyyy" and the price currency is "USD". Without reconciling these context differences, the so-composed *CS* (expressed in BPEL) cannot even execute, let alone provide meaningful data to the U.K. analysts. In this paper we call the BPEL file created without considering context differences the naive BPEL. The solution approach described below take the naive BPEL as an input and generates a new, mediated BPEL file, which reconciles these context differences.

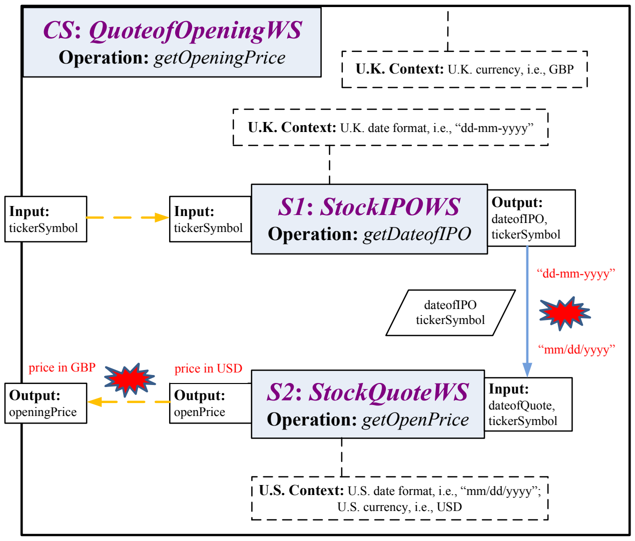
CS: QuoteofOpeningWS
Operation: getOpeningPrice
U.K. Context: U.K. currency, i.e., GBP
U.K. Context: U.K. date format, i.e., "dd-mm-yyyy"
Input: tickerSymbol
Input: tickerSymbol
S1: StockIPOWS
Operation: getDateofIPO
Output: dateofIPO, tickerSymbol
"dd-mm-yyyy"
dateofIPO tickerSymbol
"mm/dd/yyyy"
price in GBP
price in USD
Output: openingPrice
Output: openPrice
S2: StockQuoteWS
Operation: getOpenPrice
Input: dateofQuote, tickerSymbol
U.S. Context: U.S. date format, i.e., "mm/dd/yyyy"; U.S. currency, i.e., USD

Figure 1. Composition of Web services with context heterogeneity

3. Representation of contexts and conversions

To reconcile context heterogeneity in Web services composition, the contexts need to be explicitly represented by adopting an ontology and specifying the mapping relations of data and messages to the concepts in the ontology. Also, the conversions for reconciling context differences need to be specified.

3.1. Common ontology with context

Different from most other ontology-based approaches [7, 8, 12], the ontology in our approach accommodates multiple variations (i.e., contexts) for interpreting a same high-level concept. A construct *modifier* is introduced to represent contexts. A graphical representation of the ontology for the example is given in Figure 2.

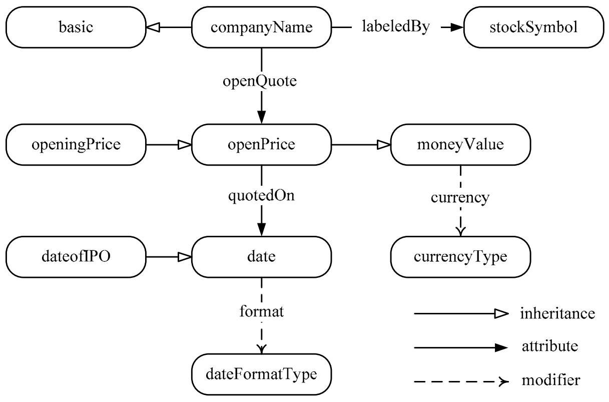

Figure 2. Common ontology

In Figure 2, concepts are depicted by round rectangles and *basic* is the special concept from which all other concepts inherit. In the common ontology, three kinds of relationships need to be explicitly depicted: *inheritance*, *attribute* and *modifier*. Inheritance and attribute are two classic relationships originated from object-oriented modeling. For example, the concept openingPrice inherits from the concept openPrice, since openingPrice represents the specific open price of a company's stock on the first trading day. Attributes are the structural relationships between two concepts and depicted by solid arrows in Figure 2. For instance, the attribute quotedOn indicates that each object of the concept openPrice is quoted on a specific date. To explicitly represent context semantics, we adopt the notion of modifier which originates from our previous work [9, 10, 13] to enrich the domain ontology. Modifiers are a special kind of attributes used to capture the additional metadata that affects the data interpretation of the exchanged messages. Each modifier corresponds to a context variation of the exchanged data. Once enriched by these modifiers, the contexts of the concepts in the ontology are made explicit. Modifiers are depicted by dashed arrows in Figure 2. For example, the concept moneyValue has a modifier currency which indicates a potential variation of the data interpretation of its primitive objects. The values of the data objects of moneyValue may vary according to the different values of currencyType. In our example, it may take two different values, i.e., "GBP" in the U.K. context and "USD" in the U.S. context. That means that the data values of moneyValue may be interpreted as British pounds or U.S. dollars. Similarly, the values of the concept date may be interpreted by "dd-mm-yyyy" or "mm/dd/yyyy", since it has a modifier format which relates to the concept dateFormatType that may take the two different values.

3.2. Semantic annotation using an extension of SAWSDL

The WSDL descriptions of the involved Web services are annotated using the W3C standard, i.e., Semantic Annotation for WSDL and XML Schema (SAWSDL) [14]. SAWSDL itself provides no explicit semantics, but enables developers to annotate the purely syntactic WSDL descriptions with pointers to concepts defined in ontologies [15]. SAWSDL provides an extension attribute, namely *modelReference*, for specifying WSDL components (e.g., data types, input and output messages) to point to corresponding semantic concepts in the common ontology. However, context semantics are ignored in the current SAWSDL specification. To this end, we propose to extend SAWSDL with another extension attribute, namely *contextReference*, which is intended to indicate the context of a WSDL file. Specifically, *contextReference* is used to annotate *<wsdl:descriptions>*.

Figure 3 (in the next page) presents a snippet of the annotated WSDL file of *S1* (i.e., *StockIPOWS*) in which the annotations are bold and emphasized. The extension attribute *contextReference* indicates that the data consumed and provided by *S1* should be interpreted using the U.K. context (indicated by the value "c_uk"). The member elements of input and output messages of *S1* are annotated with *modelReference* to point to concepts defined in the common ontology shown in Figure 2. The WSDL files of the component service *S2* (i.e., *HistoricalStockQuoteWS*) and composite service *CS* (i.e., *QuoteofOpeningWS*) would be annotated in a similar way.

3.3. Conversion for reconciliation

When context differences are detected, appropriate conversions between the different contexts need to be performed to convert the exchanged data from the source context to the target context. We propose two methods for the conversions: one uses XPath functions, and the other uses external Web services.

For certain simple conversions, the conversion code can be specified in a lightweight way, e.g., XPath funcions. For example, Figure 4 presents the conversion function defined in XPath to convert the date instance *Vs* of the object *_O* of concept date in the context *Ctxt* where the value of modifier format is *Mvs* (i.e, "dd-mm-yyyy") to the date instance *Vt* of object _O in the target context where the value of modifier format is *Mvt* (i.e., "mm/dd/yyyy"). The XPath function can be encapsulated as a custom function for further reuse, namely *Vt* = convertDateFormatFmUK2US(*Vs*).

```
<wsdl:definitions targetNamespace="http://stockQuote.coin.mit"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" …
sawsdl:contextReference="c_uk" >
<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="http://interchange.mit.edu/c
oin/ontologies/stockQuote#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="http://interchange.mit.edu/c
oin/ontologies/stockQuote#dateofIPO" />
   <element name="tickerSymbol" nillable="true" type="xsd:string"
sawsdl:modelReference="http://interchange.mit.edu/c
oin/ontologies/stockQuote#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>
```

Figure 3. Snippet of an annotated WSDL file of *S1*

As a lightweight method, XPath functions could not address certain complex conversions. In such cases, conversions can be performed by external Web services. In the motivating example, the stock price in USD should be converted to the price in GBP. The currency conversion is thus required and can be performed by a external Web service, namely, DOTSCurrencyExchange (available at [16]), which is discovered in the public service registry [11].

```
CvtUK2US(date, _O, format, Ctxt, Mvs, Vs, Mvt, Vt) :-
    Vt = Concat(
    substring-before( substring-after(Vs,"-") ,"-") ,"/",
    substring-before(Vs,"-"),"/",
    substring-after(substring-after(Vs,"-") ,"-") )
```

Figure 4. Conversion rule for converting date formats from "dd-mm-yyyy" to "mm/dd/yyyy"

4. Reconciliation approach

4.1. Translation from BPEL to process algebra

To assist with analysis and reasoning, we first translate the naive BPEL to LOTOS which is a kind of Process Algebras (PAs). LOTOS is an ISO specification language [17] can describe both the static and dynamic aspects of BPEL processes. We choose LOTOS, rather than other PAs, as the abstract formalism because of its rich expressiveness – it allows for the definition of complex data structures and data handling [18, 19]. Using LOTOS, we can depict and analyze the messages defined in BPEL (and involved WSDL) which usually have more than one part/element. In case of other PAs, the messages are depicted as tokens and its parts/elements cannot be distinguished.

Table 2. Mapping examples of BPEL and simplified LOTOS

BPEL	Simplified LOTOS
<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>*	(*v1* >> *v2*; *v3* >> *v4*)
<sequence …> *<…action1…/>* *<…action2…/>* *</sequence>*	(…*action1*…); (…*action2*…)

We identify the mapping between basic constructs of BPEL and the simplified LOTOS. A few mapping examples are given in Table 2, which are used for the motivating example. Consider the first two mappings, where *v* is a variable, while *c* is a channel derived from the attributes of *partner link*, *port type* and *operation* in the BPEL specification. Thus, "*c*?*v*" and "*c*!*v*" denote a message *v* is received and emitted through the channel *c*, respectively. The LOTOS operator ";" denotes the sequential order of two 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, explicit data transfers are represented using the operator “>>”. Details of other mappings and process algebra operators can be found in [18, 20].

By using the above mappings, we translate the naive BPEL to a process of the simplified LOTOS, as shown in Figure 5. To simplify illustration, we use label (i.e., CS, S1, S2) to represent corresponding channels of the actions and XPath expression (i.e., “/”) to identify certain part/element nested in the messages of WSDL.

```
variable getOpeningPrice
variable getDateofIPOResponse
variable getOpenPrice
variable getOpeningPriceResponse
(CS?getOpeningPrice)
;(S1!getOpeningPrice;        // implicit transfer
   S1?getDateofIPOResponse)
;(getDateofIPOResponse/
    impl:getDateofIPOReturn/
    impl:tickerSymbol >>    // explicit transfer
    getOpenPrice/impl:tickerSymbol;
   getDateofIPOResponse/
    impl:getDateofIPOReturn/
    impl:dateofIPO >>       // explicit transfer
    getOpenPrice/impl:dateofStock)
;(S2!getOpenPrice;
   S2?getOpeningPriceResponse)
;(CS!getOpeningPriceResponse)  // implicit transfer
```

Figure 5. Process in simplified LOTOS

4.2. Detection of context conflicts

Based on our observation, context conflicts within a BPEL process occur whenever data transfers exist - the data provided by a Web service is consumed by another one through either the activity assign or a shared variable storing the data. In the former case the data transfer is explicit, while in the latter case the data transfer is implicit. We need to identify both types of data transfers within the BPEL expressed in LOTOS.

4.2.1. Identifying data transfers. Explicit data transfers can be identified easily, as they are indicated using the operator “>>” in the simplified LOTOS. For example, the data transfers of *tickerSymbol* and *dataofIPO* between the invocations of the two component Web services (i.e., *S1* and *S2*) are explicit, as shown in Figure 5. Implicit data transfers can be inferred according to shared variables. In the motivating example, *tickerSymbol* is initially received as the input of the composite service (i.e., *CS*) and stored in variable *getOpeningPrice* – *tickerSymbol* is an element defined in the nested data type of *getOpeningPrice*. As shown in Figure 5, *getOpeningPrice* is directly used as the input variable of the invocation of *S1* and then received by *S1*. Thus, the data transfer of *tickerSymbol* from the input of *CS* to the input of *S1*, through variable *getOpeningPrice*, is implicit. In general, a data transfer is considered to be implicit, once the data is transferred through a shared variable but its reception and emission channels (expressed in LOTOS) are different. Implicit data transfers can be identified through inferring the processes of the simplified LOTOS. Similarly, the data transfer of *openingPrice* from the output of *S2* to the output of *CS*, through variable *getOpeningPriceResponse*, is also implicit.

4.2.2. Detecting context conflicts in data transfers. In the BPEL process, each data transfer relates a source Web service and a target Web service, both of which can be identified by analyzing the reception and the emission channels of the data transfer. Since context conflicts occur in data transfers, we need to check whether the contexts of the source and the target services are different. Take the explicit data transfer of *dateofIPO* as an example. Its source variable is *getDateofIPOResponse* whose data structure is defined in the WSDL of *S1*, while the target variable is *getOpenPrice* defined in the WSDL of *S2*. Correspondingly, its source and target services are *S1*, *S2*, respectively. With the semantic annotation presented in Section 3.2, *dateofIPO*, which is in the annotated WSDL file of *S1*, points to concept dateofIPO (see the common ontology in Figure 2). Also, *dateofStock*, which is in the annotated WSDL file of *S2*, points to concept date which is the super concept of dateofIPO. By reasoning in the common ontology, we know that date has a modifier (i.e., format) and dateofIPO inherit this modifier. The modifier format points to concept dateFormatType which has 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 data transfers of the BPEL process of the motivating example, we detect another context conflict within the implicit data transfer of *openingPrice*.

4.3. Generation of mediated BPEL

The mediated BPEL is translated from the mediated LOTOS which is generated by inserting a necessary conversion into each data transfer with context conflicts. The conversion is specified in Section 3.3 to reconcile these conflicts, i.e., to convert the exchanged data in source context to target context.

4.3.1. Making implicit data transfers with context conflicts explicit. To insert conversion functions, each implicit data transfer with context conflicts needs to be made explicit. In the motivating example, the implicit data transfer of *openingPrice* contains a context conflict in terms of different currencies of the stock price. The currency of *S2* is “USD”, while that of *CS* is “GBP”. To make the data transfer explicit, a new variable, namely *getOpenPriceResponse*, is declared as the output variable of the invocation of *S2* in the composition. The data type of *getOpenPriceResponse* is defined as the same with variable *getOpeningPriceResponse*. Then, an action of assigning *getOpenPriceResponse* to *getOpeningPriceResponse* is inserted to the process.

```
variable getOpeningPrice
variable getDateofIPOResponse
variable getOpenPrice
variable getOpeningPriceResponse
variable getOpenPriceResponse
(CS?getOpeningPrice)
;(S1!getOpeningPrice;S1?getDateofIPOResponse)
;(getDateofIPOResponse/
   impl:getDateofIPOReturn/
   impl:tickerSymbol >>
   getOpenPrice/impl:tickerSymbol;
  getDateofIPOResponse/
convertDateFormatFmUK2US(impl:getDateofIP
OReturn/impl:dateofIPO) >>
   getOpenPrice/impl:dateofStock)
;(S2!getOpenPrice;S2?getOpenPriceResponse)
;(ES!getOpenPriceResponse;
   ES?getOpeningPriceResponse)
;(CS!getOpeningPriceResponse)
```

Figure 6. Mediated LOTOS without context conflicts

4.3.2. Inserting conversions to explicit data transfers with context conflicts. As mentioned above, context conflicts can be identified by referring to context definitions described in the common ontology and all data transfers with context conflicts are made explicit. In the motivating example, two context conflicts are detected: one is the difference of interpreting the date formats (i.e., “dd-mm-yyyy” vs. “mm/dd/yyyy”), and the other is the difference of interpreting the price currencies (i.e., “USD” vs. “GBP”). The conversion of the two different formats of dates can be directly performed by the pre-defined XPath function (presented in Figure 4) which needs to be inserted in the appropriate place of the process in LOTOS. The conversion of two different price currencies has to invoke the external Web service, i.e., DOTSCurrencyExchange (denoted by *ES* for short), which is previously discovered in Section 3.3. The action of invocation of *ES* is inserted to substitute the action of assignment which is added in Section 4.3.1. Figure 6 presents the mediated process in the simplified LOTOS. The actual code of the mediated BPEL is then generated using the translation from LOTOS to BPEL.

5. Validation

To validate the feasibility of our approach, the motivating example presented in Section 2 has been successfully demonstrated in our Java-based development prototype. The two component Web services [3], i.e., *StockIPOWS* and *HistoricalStockQuoteWS*, were developed by Eclipse Web Tools Platform (WTP)[4] and hosed by the Apache Axis2 Web service engine. The WSDL description of the composite service, i.e. *QuoteofOpeningWS*, was also developed. These WSDL descriptions were annotated with semantic pointers using an Eclipse plug-in, i.e., Radiant [21]. The common ontology including context semantics, was defined using our COIN Model Application Editor [22] which is a lightweight tool for creating and editing ontologies in RDF/OWL. By deliberately ignoring context heterogeneity amongst the involved Web services, we assumed that the U.K. developer created a naive BPEL process by using ActiveVOS Designer[5]. Also, we have developed a mediation tool which is a Java-based GUI for the translation between abstract BPEL processes to LOTOS. By relating the variables defined in the LOTOS statement to semantic concepts, the mediation tool performed reasoning on the common ontology. Potential context conflicts within the naive BPEL process were fully detected. The mediation tool generated the mediated LOTOS by introducing the predefined conversion function and currency exchange service. Eventually, the mediation tool translated the mediated LOTOS to produce the mediated BPEL without any context conflict.

By searching the public service registry [11], we also investigated other real-world Web services and composition situations, and found that context heterogeneity widely exists among the existing Web services, such as different scale factors of numbers, currencies of prices, different representations of time and positions. Our approach can be used to reconcile these types of context heterogeneity to develop correct composite Web services.

6. Related Work

6.1. Semantic Web services

Applying Semantic Web technologies to Web services is an active research area, referred to as Semantic Web Services (SWSs) [7, 8, 12, 23]. In this area, OWL-S, WSMF/WSMO and METEOR-S are three major initiatives. OWL-S [2] provides a general specification

[3] These are simplified versions of actual Web services.
[4] http://www.eclipse.org/webtools/
[5] http://www.activevos.com/community-open-source.php

language for semantically describing Web services and binding the descriptions to ontologies in OWL/RDF. WSMF [3] is a framework consisting of four core elements for describing SWSs, and WSMO [4] provides an ontological specification for these core elements. While OWL and WSMF/WSMO propose rich ontologies and frameworks, METEOR-S [5, 6] focuses on semantic annotation of the industrial standards of Web services, such as WSDL. As a significant result developed by METEOR-S, SAWSDL [14, 15] is the W3C's first step towards standardizing semantic annotation for WSDL. Besides that, many promising approaches have been developed for the vision of SWSs, such as semantic discovery [24, 25], and composition [12, 26]. However, few of the above works addresses context heterogeneity among Web services.

6.2. Semantic interoperability of Web services

The promising vision of service composition heavily relies on semantic interoperability. The challenges of semantic interoperability (e.g., semantic heterogeneity) have been discussed in [1]. Based on METEOR-S, semantic annotation and mapping are proposed [1, 27] to deal with the semantic heterogeneity to facilitate service interoperability. For transforming discrepant messages from the upstream Web service to the downstream one, a resolution-based inference mechanism for running data transformation rules is developed [28]. Based on WSMO, the mediation architecture, i.e., WSMX [29], is proposed which acts on the semantic descriptions. However, context semantics and heterogeneity are widely ignored by these efforts. To the best of our knowledge, only two projects exist in the literature towards the specific direction. One is the work in [30, 31] which proposes mapping relations for reconciling context heterogeneity. But the composition situations addressed in this work are different from ours. In practice, most composition processes expressed in BPEL specification suffer from context heterogeneity that needs to be reconciled. The other work in [32-34] develops an approach to reconciling context heterogeneity and generating the mediated BPEL. The authors propose an ad hoc extension of the WSDL specification to explicitly represent the context semantics in WSDL descriptions. However, there are several distinctions between our approach and their work. For example, the WSDL descriptions are directly annotated with the context definitions in their work - modifier values are enumerated in the WSDL elements. In case of a large number of modifier values, it is difficult for developers to enumerate and maintain so many modifier values in the WSDL elements. Differently, our approach specifies modifier values in the common ontology separate from WSDL descriptions and adopts an extension of SAWSDL to annotate WSDL elements.

7. Conclusion

Context heterogeneity widely exists among existing Web services and needs to be reconciled when composing them. In this paper, we presented an approach to detecting and reconciling context heterogeneity in BPEL composition. The approach can generate appropriate mediated BPEL file to reconcile context heterogeneity. One benefit of our approach lies in the separation of context definition and WSDL description. Correspondences between them are established using an extension of SAWSDL. Also, our approach provides a solid formalism (i.e., LOTOS) for analyzing data transfers and detecting context conflicts in BPEL processes.

In the future, we plan to enhance the ability of our approach to address more complex composition situations. We will also adapt our approach to address schematic heterogeneity in Web services composition.

8. Acknowledgement

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). Also, the authors thank Frank Manola for his valuable suggestions in improving the work.

9. References

[1] M. Nagarajan, K. Verma, and A. P. Sheth, et al., "Semantic Interoperability of Web Services - Challenges and Experiences", *Proc. of the 4th Int. Conf. on Web Services*, Chicago, USA, 2006, pp. 373-382.

[2] D. Martin, M. Burstein, and D. McDermott, et al., "Bringing Semantics to Web Services with OWL-S", *World Wide Web*, vol. 10, no. 3, 2007, pp. 243-277.

[3] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF", *Electronic Commerce Research and Applications*, vol. 1, no. 2, 2002, pp. 113-137.

[4] H. Lausen, A. Polleres, and D. Roman, "Web Service Modeling Ontology (WSMO)", *W3C Member Submission*, vol. 3, 2005.

[5] K. Sivashanmugam, K. Verma, and A. Sheth, et al., "Adding Semantics to Web Services Standards", *Proc. of the 1st Intl. Conf. on Web Services*, 2003, pp. 395–401.

[6] A. A. Patil, S. A. Oundhakar, and A. P. Sheth, et al., "Meteor-s web service annotation framework", *Proc. of the 13th Intl. Conf. on World Wide Web*, 2004, pp. 553-562.

[7] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services", *IEEE Intelligent Systems*, vol. 16, no. 2, 2001, pp. 46-53.

[8] B. Medjahed, A. Bouguettaya, and A. Elmagarmid, "Composing Web services on the Semantic Web", *The VLDB Journal*, vol. 12, no.4, 2003, pp. 333-351.

[9] C. H. Goh, S. Bressan, and S. Madnick, et al., "Context interchange: new features and formalisms for the intelligent integration of information", *ACM Transactions on Information Systems*, vol. 17, no. 3, 1999, pp. 270-293.

[10] S. Bressan, C. Goh, and N. Levina, et al., "Context Knowledge Representation and Reasoning in the Context Interchange System", *Applied Intelligence*, vol. 13, no. 2, 2000, pp. 165-180.

[11] "Web Service Search Engine @ seekda.com", http://seekda.com/.

[12] K. Sycara, M. Paolucci, and A. Ankolekar, et al., "Automated discovery, interaction and composition of Semantic Web services", *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 1, 2003, pp. 27-46.

[13] M. D. Siegel and S. E. Madnick, "A Metadata Approach to Resolving Semantic Conflicts", *Proc. of the 17th Intl. Conf. on Very Large Data Bases*, 1991, pp. 133-145.

[14] J. Farrell and H. Lausen, "Semantic Annotations for WSDL and XML Schema", *W3C Recommendation*, August 28, 2007.

[15] J. Kopecký, T. Vitvar, and C. Bournez, et al., "SAWSDL: Semantic Annotations for WSDL and XML Schema," IEEE Internet Computing, vol. 11, no . 6, 2007, pp. 60-67.

[16] "Currency conversion service: DOTSCurrencyExchange", http://ws2.serviceobjects.net/ce/CurrencyExchange.asmx?WSDL.

[17] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS", *Computer Networks and ISDN Systems*, vol. 14, no. 1, 1987, pp. 25-59.

[18] A. Ferrara, "Web services: a process algebra approach", *Proc. of the 2nd Intl. Conf. on Service Oriented Computing,* 2004, pp. 242-251.

[19] L. Bordeaux and G. Salaun, "Using Process Algebra for Web Services: Early Results and Perspectives", *Proc. of the 5th Intl. Workshop on Technologies for E-Services*, 2004, pp. 54–68.

[20] G. Salaun, L. Bordeaux, and M. Schaerf, "Describing and reasoning on Web Services using Process Algebra", *International Journal of Business Process Integration and Management*, vol. 1, no . 2, 2006, pp. 116-128.

[21] K. Gomadam, K. Verma, and D. Brewer, et al., "Radiant: A tool for semantic annotation of Web Services", *Proc. of the 4th Semantic Web Conference*, Demo paper, 2005.

[22] P. W. Lee, "Metadata Representation and Management for Context Mediation", *Composite Information Systems Laboratory Working Paper# 2003*, vol. 1, 2003.

[23] M. Burstein, C. Bussler, and T. Finin, et al., "A semantic Web services architecture", *IEEE Internet Computing*, vol. 9, no. 5, 2005, pp. 72-81.

[24] K. Sivashanmugam, K. Verma, and A. Sheth, "Discovery of Web services in a federated registry environment", *Proc. of 2nd Intl. Conf. on Web Services*, 2004, pp. 270-278.

[25] K. Sycara, M. Paolucci, and J. Soudry, et al., "Dynamic Discovery and Coordination of Agent-Based Semantic Web Services", *IEEE Internet computing*, vol. 8, no. 3, 2004, pp. 66-73.

[26] I. B. Arpinar, R. Zhang, and B. Aleman-Meza, et al., "Ontology-driven Web services composition platform", *Information Systems and E-Business Management*, vol. 3, no. 2, pp. 175-199, 2005.

[27] Z. Wu, A. Ranabahu, and K. Gomadam, et al., "Automatic Composition of Semantic Web Services using Process and Data Mediation", *Proc. of the 9th Intl. Conf. on Enterprise Information Systems*, Funchal, Portugal, 2007, pp. 453-461.

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

[29] E. Cimpian and A. Mocan, "WSMX Process Mediation Based on Choreographies", *Proc. of Workshop on Web Service Choreography and Orchestration for Business Process Management*, 2006, pp. 130-143.

[30] D. Gagne, M. Sabbouh, and S. Bennett, et al., "Using Data Semantics to Enable Automatic Composition of Web Services", *Proc. of the 3rd Intl. Conf. on Services Computing*, 2006, pp. 438-444.

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

[32] M. Mrissa, C. Ghedira, and D. Benslimane, et al., "A Context Model for Semantic Mediation in Web Services Composition", *Proc. of the 25th Intl. Conf. on Conceptual Modeling*, 2006, pp. 12-25.

[33] M. Mrissa, C. Ghedira, and Z. Maamar, et al., "Context and Semantic Composition of Web Services," *Proc. of the 17th Intl. Conf. on Database and Expert Systems*, 2006, pp. 266-275.

[34] M. Mrissa, C. Ghedira, and D. Benslimane, et al., "A context-based mediation approach to compose semantic Web services", *ACM Transactions On Internet Technology*, vol. 8, no. 1, 2007, pp. 4:1-23.