

A Context-Based Mediation Approach to Compose Semantic Web Services

MICHAEL MARISSA, CHIRINE GHEDIRA, and DJAMAL BENSLIMANE

Université Claude Bernard Lyon 1

ZAKARIA MAAMAR

Zayed University

and

FLORIAN ROSENBERG and SCHAHRAM DUSTDAR

Technical University of Vienna

Web services composition is a keystone in the development of interoperable systems. However, despite the widespread adoption of Web services, several obstacles still hinder their smooth automatic semantic reconciliation when being composed. Consistent understanding of data exchanged between composed Web services is hampered by various implicit modeling assumptions and representations. Our contribution in this article revolves around context and how it enriches data exchange between Web services. In particular, a context-based mediation approach to solve semantic heterogeneities between composed Web services is presented.

Categories and Subject Descriptors: D.2.12 [**Software Engineering**]: Interoperability

General Terms: Algorithms, Design, Languages, Standardization

Additional Key Words and Phrases: Web services, composition, mediation, semantics, context

ACM Reference Format:

Mrisa, M., Ghedira, C., Benslimane, D., Maamar, Z., Rosenberg, F., and Dustdar, S. 2007. A context-based mediation approach to compose semantic Web services. *ACM Trans. Intern. Tech.* 8, 1, Article 4 (November 2007), 23 pages. DOI = 10.1145/1294148.1294152 <http://10.1145/1294148.1294152>

Author's addresses: M. Mrisa, C. Ghedira, D. Benslimane, Université Claude Bernard Lyon 1, UFR Informatique, Bâtiment Nautibus 8, boulevard Niels Bohr, 69622 Villeurbanne cedex, France; email: michael.mrisa@fundp.ac.be; {chirine.ghedira, djamal.benslimane}@iris.cnrs.fr; Z. Maamar, Zayed University, Academic City, P.O. Box 19282; Dubai, United Arab Emirates; email: zakaria.maamar@zu.ac.ae; F. Rosenberg, S. Dustdar, Distributed Systems Group (DSG), Information Systems Institute, Technical University of Vienna, Argentinierstrasse 8/184-1, A-1040 Vienna, Austria; email: {florian, dustdar}@infosys.tuwien.ac.at.

Permission to make digital or hard copies part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org. © 2007 ACM 1533-5399/2007/11-ART4 \$5.00 DOI 10.1145/1294148.1294152 <http://doi.acm.org/10.1145/1294148.1294152>

ACM Transactions on Internet Technology, Vol. 8, No. 1, Article 4, Publication date: November 2007.

1. INTRODUCTION

Developed around a platform-independent protocol stack that heavily relies on standards like SOAP [Box et al. 2000], WSDL [Christensen et al. 2001], and UDDI [UDDI Working Group 2001], Web services¹ are now widely adopted as a means to interconnect applications over the Internet. Web services reach their full potential when composed into business processes that provide users with new functionalities. Composition means orchestrating several Web services according to a business process which is specified with a composition language. Nowadays, WS-BPEL [Andrews et al. 2003] is the de facto standard for Web services composition. Coming from the domain of workflow management, WS-BPEL provides several constructs that permit handling complex interactions between business partners, such as concurrent invocation, fault recovery, and conditional switches.

Despite a wide adoption, WS-BPEL still remains focused on the syntax level with no emphasis on the semantic aspects of composition. Moreover, the Web service protocol stack was not initially developed to meet the requirements of a successful semantic exchange. Semantic exchange makes Web services *understand* the content of messages they send and receive. For this purpose, recent work from the semantic Web community aims at explicitly describing the semantics of Web services. Several languages and approaches² aim at propelling Web services to the level of *semantic Web services*. Such approaches use ontologies (shared descriptions of domain knowledge [Gruber 2000]) as agreements on a common vocabulary.

However, the aforementioned initiatives do not exploit the potential that *context* offers when it comes to describing the different aspects of data. The use of context representation has been widely explored in the domain of multidatabase systems [Kashyap and Sheth 1996; Sciore et al. 1994] in order to clarify the semantic and schematic aspects of data, but remains barely looked into in the field of Web services. The term “context” relates to the collection of implicit assumptions that are required to perform a correct interpretation of data. We advocate that accurate data interpretation not only depends on a single semantic reference, but also on several properties and characteristics that form the *context of interpretation* of data. Context needs to be explicitly described so that data can be clearly understood according to this context. Particularly, in the case of Web services composition, overcoming the challenges of automatic semantic interpretation and data flow handling requires explicit context description and management.

In this article we aim at *investigating the automated semantic reconciliation of semantic Web services*. The rationale of this reconciliation is backed by

¹A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards (source: W3C).

²For example, WSMO [Arroyo and Stollberg 2004], SESMA [Peer 2005], DIANE [Klein et al. 2005], OWL-S [Martin et al. 2004], and WSDL-S [Miller et al. 2004].

[Maamar et al. 2006], who argue that a contextual semantic composition of Web services is subject to satisfying two conditions. The first condition is that Web services must agree on the meaning of the exchanged data, and the second is that semantic-data conflicts must be automatically resolved using the information that context provides.

Our main contribution revolves around a proposal for *context-based mediation for reconciling semantic Web services engaged in composition* as an extension of our previous work [Mrissa et al. 2006a, 2006b]. This proposal consists of: (1) a context-based annotation of WSDL messages with data semantics; (2) a model built on the notion of semantic objects to support explicit description of context; and (3) a semantic-mediation architecture for WS-BPEL processes, which handles information heterogeneities using semantic objects and Web services' contexts. Our experiments concern travel planning; nevertheless, our proposal is generic and can be applied to other domains such as online e-government, banking, and medical services. In fact, all the application domains that use semantic Web services can benefit from our proposed mediation architecture, as long as the WSDL documents of composed Web services are properly annotated with the necessary semantic information.

The rest of this article is organized as follows. Section 2 reviews literature on semantics and mediation of Web services. Section 3 presents a motivating example and summarizes the different discrepancies that can hamper the correct interpretation of data exchanged between Web services. Section 4 summarizes our context model and proposes a solution to manage context information. Section 5 details the functioning of the different elements of our mediation approach. Section 6 describes the prototype developed as a proof-of-concept of our approach. Section 7 concludes the article and gives directions for future work.

2. RELATED WORK

This section presents different initiatives related to semantic description and mediation aspects of Web services that helped shape our proposal.

2.1 Semantic Description of Web Services

At the crossing of the semantic Web and Web services domains, research in the field of semantic Web services is very active. Most approaches describe the semantics of Web services, either with novel semantic description languages [Martin et al. 2004; Peer 2005; Arroyo and Stollberg 2004; Klein et al. 2005] or with extensions to syntactic standards [Miller et al. 2004; SAWSDL Working Group 2006]. These approaches bind to domain ontologies in order to explicitly describe the intended meaning of data, including data that Web services exchange.

OWL-S [Martin et al. 2004] is a subset of the OWL [Schreiber and Dean 2004] ontology language. It is a general ontology for building semantic Web services, and was designed to be coupled with syntactic description formats like WSDL. Specifically, OWL-S consists of three elements: the *service profile*, *process model*, and *service grounding*, that describewhat the service does, how

the service works, and how to access the service, respectively. OWL-S advocates to separate the grounding and abstract views when describing the data that Web services exchange. Abstract view binds the data to an OWL conceptual description. Grounding view describes the low-level representation of data, which generally follows XML Schema [W3C 2004]. This separation allows different physical representations of the same concept, and strengthens as well the role of ontologies in the abstract representation of data semantics.

WSMF [Fensel and Bussler 2002], from the DERI laboratory, supports the development and description of semantic Web services with a conceptual model. This model recommends maximal decoupling between Web services, and was designed with the idea to enable mediation as a service. WSMO [Arroyo and Stollberg 2004] is a formal language and ontology based on the WSMF conceptual model that describes multiple aspects of semantic Web services.

Medjahed and Bouguettaya [2005] propose a foundational architecture for semantic Web services. Their work is based on the concept of community, which gathers services from the same domain of interest and publishes the functionalities offered by Web services as generic operations. The authors provide a general template referred to as community ontology for describing semantic Web services and communities. Their work follows a realistic community-centric point-of-view, and adopts a peer-to-peer solution to manage communities, which addresses the problems of centralized approaches.

DIANE elements (DE) and DIANE service description (DSD) are object-oriented languages built on a critical analysis of the requirements of semantic Web service description, and on the difficulties of OWL-S and WSMO to fulfill these requirements [Klein et al. 2005]. DE and DSD use configurable sets and fuzzy logic to support semantic discovery of Web services. DE is a general ontology language with specific features to enhance semantic Web service description. DSD describes services with the constructs provided by DE. Moreover, DSD revolves around the notions of service request and offer descriptions.

SESMA [Peer 2005] is another description format for semantic Web services that was designed to provide a language with a compact syntax. SESMA supports nondeterministic service description, and is compatible with standards such as WSDL and WS-BPEL. Its main advantage is being a lightweight language for semantic description of Web services. It remains close to WSDL and WS-BPEL descriptions, and its semantics is not built on top of OWL.

With WSDL-S, [Miller et al. 2004] annotate WSDL with several extensions related to operations and messages. These extensions refer to concepts of domain models in order to specify not only the semantics of messages, but also the preconditions and effects of operations. WSDL-S is also described as a lightweight approach for semantic annotation of Web services.

SAWSDL is a W3C working draft that defines a set of extension attributes to WSDL 2.0 (with WSDL 1.1 support) in order to describe the semantics of WSDL elements [SAWSDL Working Group 2006]. The objective of SAWSDL is to define how semantic annotation of WSDL is accomplished, but it is not intended to specify which language has to be used for the semantic description. It just provides the mechanisms to bind ontology concepts to WSDL annotations.

To wrap up this section, it should be noted that none of the existing approaches uses context to describe the semantics of data. The approach presented in this article is one step towards enriching descriptions of Web services with context information. Therefore, it should be possible to combine or extend the aforementioned works with the context-based approach we present in this article.

2.2 Mediation Between Web Services

Mediation between Web services has received a lot of attention from the research community. Many mediation approaches rely on the concept of mediators for solving data heterogeneities between participants in a composition. Mediators were first introduced in the domain of databases [Wiederhold 1992], and later adapted to the domain of Web services.

[Mocan et al. 2004] propose WSMX, an implementation of the Web service modeling ontology project (WSMO). WSMX is a mediation architecture for the integration of Web services. The mediator component is a key part of their architecture and is designed as a service. It mediates between concepts of the different ontologies to which business partners bind. The mediation solution that Mocan et al. propose relies on a semiautomated graphical interface that allows users to define conversion rules between ontologies. End-users are helped by suggestions from the system, based on the structure of concepts and already established relations between concepts.

Cabral and Domingue [2005] provide a broker-based mediation approach to compose semantic Web services. Their approach follows the WSMF conceptual framework. The mediator component is a key part of their architecture and mediates the concepts between ontologies to which business partners refer. Williams et al. [2005] use agents to perform semantic mediation between input and output parameters of Web services. They encapsulate the composition into an agent that controls the operation progress.

Spencer and Liu [2004] present a rule-based approach to semantically match Web services' outputs and inputs. A description-logic reasoning system analyzes OWL-S descriptions and generates multiple data transformation rules. This approach focuses on the conversion between different representations of matching OWL-S classes.

While mediation and semantic description of the Web services in a composition are very active research fields, to the best of our knowledge, none of these works actually considers context to solve semantic heterogeneities of data in Web services composition. In the following, we present a context-based mediation approach while arguing the benefits of context for Web services.

3. LIMITATIONS OF SEMANTIC APPROACHES

In this section, we demonstrate using the classical travel example the need for additional metainformation to accurately interpret the data exchanged between Web services. We stress as well the limitations of current semantic approaches relative to the concern of semantic interpretation of data.

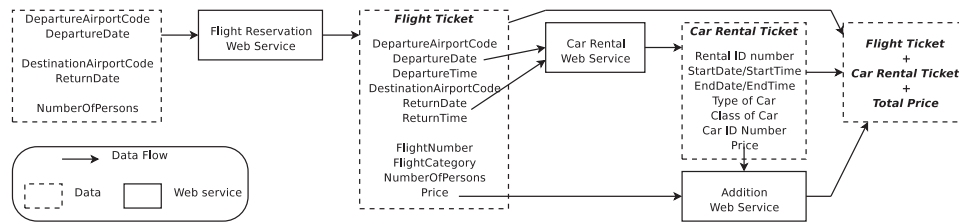


Fig. 1. Flight-booking and car rental Web services.

3.1 Motivating Example

We consider a user who plans a trip to Japan, and wishes to rent a car during her stay. The WSDL files of flight-booking and car rental companies are composed with the help of a graphical composition editor³ in order to constitute the workflow presented in Figure 1. A basic addition Web service is used to deliver the sum of flight-booking and car rental prices. This Web service is deemed appropriate because of the inherent limitations of XPath, which only supports integer arithmetic and is not intended to perform arbitrary computations [Andrews et al. 2003]. In addition, we assume that the data flows between Web services verify low-level data compatibility. By low-level compatibility, we mean that data objects are described with the same data types (generally with the XML schema type system). This verification is easily performed by the composition editor program.

In Figure 1 the flight-booking Web service computes prices in euros and with a scale factor⁴ of 1 because the WSDL file was provided by the European branch of the company. However, the car rental Web service uses local currency, namely Japanese yens, and a scale factor of 1000, so there is a need to multiply the value exchanged by 1000 to obtain the actual price. Prices need to be converted into the same currency and scale factor before they can be added and presented to the user.

Moreover, date and time representations differ. The flight-booking Web service uses a European notation (dd.mm.yyyy and 12:00 AM/PM), whereas the car rental Web service uses a Japanese notation (yy.mm.dd and 24:00). This example, albeit simple, shows the extent to which low-level compatibility is insufficient to meet the requirements of semantic exchange. Indeed, the interpretation of data will be inaccurate because data binds to different contexts and should be interpreted differently.

The WS-BPEL community currently solves semantic heterogeneities by manually describing the conversion of data between different semantic representations. However, this solution is tedious and not scalable. In addition, it needs to be performed up front (i.e., at design time) by domain experts, and relies on XSLT stylesheets and XPath expressions to convert data. It requires experts to have technical knowledge about both XPath and XSLT, and about

³Examples include Oracle BPEL Designer, IBM BPWS4J Editor, Vergil VCAB Composer, or Active Endpoints ActiveWebflow Designer.

⁴A number used as a multiplier in scaling (WordNet at <http://wordnet.princeton.edu/>).

the semantics of the domain concerned. Last but not least, it cannot deal with on-the-fly associations between Web services.

In the following, we discuss the main types of heterogeneities related to implicit assumptions that can hamper a correct interpretation of domain ontology concepts.

3.2 Heterogeneous Implicit Assumptions

The role of domain ontologies is to provide users with an agreement on the interpretation of described concepts. Therefore, the different semantic properties related to domain concepts remain implicit in domain ontologies. With semantic properties, we refer to the different aspects and characteristics that permit to establish the context of the current interpretation of a concept. These assumptions on the interpretation of concepts remain implicit in the ontology because the initial purpose of ontologies is to describe domain knowledge, which includes the concepts and relations between them, together with relevant properties of concepts, but not their different possible contexts of interpretation.

From a provider's point-of-view, reaching semantic interoperability with ontologies is often tedious because Web services usually make different interpretations of the same concept. Hence, providers need to adapt their own implicit assumptions on the interpretation of concepts to the assumptions of ontologies. Solving data interpretation discrepancies remains at the charge of Web service providers. This task is still performed manually and at design time. Discrepancies between semantic properties are due to different implicit assumptions on data interpretation. These discrepancies are summarized as follows.

3.2.1 Value Heterogeneities. Domain ontologies often make implicit assumptions about the values of semantic properties. As shown in Section 3.1, prices are generally assumed to have a scale factor of 1. However, some organizations usually handle prices with a scale factor of 1000. In this case, we say that the values of the *scale factor* semantic property differ. Resolving such heterogeneities requires explicit description of the semantic property and its associated value.

3.2.2 Structural Heterogeneities. Domain ontologies implicitly follow a static structural representation of semantic properties that are relevant to the described application domain. However, it appears that even in the same domain, different semantic properties and structural organizations could be relevant to Web service providers.

For example, the concept of *price* could have different relevant semantic properties depending on the needs of travel agencies. Some agencies could use different *scale factors* and ignore the *currency* aspect because their partners all use the same currency, whereas some other agencies could make the opposite assumption. These structural heterogeneities between semantic properties need to be explicitly described, so that it becomes possible to determine whether one specific structure of interpretation of a concept is compatible with another.

3.2.3 Semantic Heterogeneities. Semantic heterogeneities relate to the semantics used for describing semantic properties. They are not visible as long as the context of interpretation remains implicit, but appear when the context is explicitly described. For example, an English provider could use the word “VATIncluded”, whereas a French one would use the word “TVAINclude” to describe the same semantic property, which specifies whether value-added taxes are included in a price (synonymy conflict). In order to solve such semantic conflicts, explicit vocabulary is required to describe semantic properties.

It appears from the preceding that different types of heterogeneities affect semantic properties. In the following section, we show how the notions of context and semantic object can be used in order to explicitly describe semantic properties. We then present a context-based approach to handle the aforementioned limitations.

4. A CONTEXT MODEL FOR WEB SERVICES

In order to overcome the limitations we report in Section 3, we proposed in Mrissa et al. [2006a] a context model that is built around the notion of semantic object and context. This model gives providers the means to explicitly describe the implicit assumptions they make on data. In this section, we present the arguments that back this model, and provide insights on the description of context information, before discussing its integration into the Web services protocol stack.

4.1 Context Model: Main Concepts

The context model revolves around the notion of *semantic object*, which extends a data object with additional metadata so that the context of interpretation of the data object is made explicit. Basically, a semantic object contains: (1) a *data part* that has a value v of type t described in a type system language, and (2) a *semantic part* that has a concept c of the application domain, and a context C represented as a tree of metaattributes referred to as modifiers. Modifiers make explicit the semantic properties of semantic objects. They are also semantic objects, so they have a value, a type, a domain concept they refer to, and possibly a context. Web service providers need to make these modifiers available for a correct interpretation of a semantic object. Semantic objects with different contexts may be converted into a common context, which allows to compare them and even to enable data exchange between Web services.

For the needs of our model, we defined two categories of modifiers: *static* and *dynamic*. It is mandatory to provide static modifiers in order to make clear the semantics of a semantic object. On the other hand, dynamic modifiers can be inferred from other modifiers belonging to the same semantic object. More details about the properties of modifiers and the possibilities of conversion raised by this model are given in Mrissa et al. [2006a].

```

concept = domain_ns:price ,
value = 5,
type = xsd:double ,
Context = [
  (ctxt_ns:currency , xsd:string , ‘‘euro’’ , [
    (ctxt_ns:country , xsd:string , ‘‘France’’ , null)
    (ctxt_ns:date , ns:date , 15.05.2005 , [
      ctxt_ns:dateformat , xsd:string , ‘‘dd.mm.yyyy’’ , null ]])
  ctxt_ns:scalefactor , int , 1 , null
  ctxt_ns:VATIncluded , xsd:bool , true , [
    ctxt_ns:VATRate , float , 19,6 , null ]
] //end of context

```

Listing 3. Sample semantic object.

In order to illustrate the proposed context, let us consider the example presented in Listing 3.

This example describes *price* as a semantic object along with its contextual characteristics. To handle a concrete price one needs to know the following details: currency, the date related to this price due to currency changes over time, and the value added taxes (VAT) rate applied to this price. We notice from this representation that the (*ctxt_ns:country*, *xsd : string*, “*France*”, *null*) modifier is static, that is, it cannot be inferred from the values of other modifiers. This static modifier can help infer the “euro” value of the *currency* modifier, being given a rule stating that euro is the currency of France. The *currency* modifier is then qualified as dynamic. In addition, we notice that the *country* modifier helps infer the value of *dateformat*, which is a dynamic modifier too. Accurate interpretation of the *price* semantic object is then possible with the attached context.

The multiple cases of heterogeneity presented in Section 3.2 can now be explicitly handled with context, and thus will not be viewed as implicit discrepancies anymore, but rather as well-known heterogeneities between instances of schemas. Indeed, context information could be directly added to the domain ontology. However, we explain in the next section why we deem appropriate to separate context from domain knowledge, and to store context information into dedicated context ontologies.

4.2 Context Representation and Integration

In this section, we detail our recommendations on how to store the information required to represent context. To this purpose, we first give an insight on the different strategies for ontology design. Then, we present the advantages of context ontologies, and explain how they are separated from domain ontologies, before presenting our WSDL annotation that achieves the integration of context into the Web services protocol stack.

4.2.1 Strategies for Ontology Design. The design of a domain ontology can be seen from different perspectives which are related to top-down, bottom-up, and middle-out approaches [Noy and Mc Guinness 2000; Noy and Hafner 1997]. A top-down approach consists in first agreeing on the most general concepts of the ontology in order to provide a shared representation of the world. This shared representation is then adjusted and specialized for the sake of meeting

the needs of the local views of the participants. It is a very reliable strategy in a limited environment in which the number of participants and the discrepancies between their views of the world are limited. However, it is not efficient in an open world like the Internet. Indeed, it is not guaranteed that an unknown and constantly changing number of participants could agree on a single view of the world. On the contrary, a bottom-up approach starts from the local and very specific conceptualizations of the participants, and follows a generalization process towards a consistent representation of the domain knowledge. Compared to the top-down approach, this one is more adapted to interactions in an open world, but appears to be less efficient in a closed environment.

The middle-out approach strengthens the middle concepts of the ontology into identifiable groups, and follows both specialization and generalization steps to build the domain knowledge representation. Generally, it is recommended to combine top-down, bottom-up, and middle-out approaches in order to model the domain knowledge of different participants in an ontology [Noy and Mc Guinness 2000]. In our case, we identify two of these three approaches and associate them with context and domain ontologies in order to describe providers' semantics.

4.2.2 Context versus Domain Ontologies. Our proposal to make the distinction between context and domain knowledge comes from the fact that the heterogeneities presented in Section 3 do not concern the domain knowledge itself, but rather relate to providers' local and yet implicit assumptions on the interpretation of domain concepts. These assumptions, referred to as context, are related to cultural, geographical, and temporal situations of Web services, such as, when, where, and how they are designed, deployed, and executed.

In most cases, when several participants intend to agree on a shared domain ontology, they already have different contexts. Especially in an open world like the Internet, we have seen that it is very difficult to reach a common agreement on a shared representation of domain knowledge when adopting a top-down approach only. This is mainly due to the different, already-existing contexts of the participants. As a consequence, we set several objectives to facilitate both ontology design and the reconciliation of Web services, given as follows.

- Focus the role of domain ontologies on describing knowledge that can be agreed on with a top-down approach;
- adopt a bottom-up approach to reconcile the contexts of Web services, as they already exist before adhering to a domain ontology; and
- give providers the responsibility of describing the structural and organizational aspects of their local contexts, and of providing the connections to other providers' contexts when they adhere to a domain ontology.

In order to meet these objectives, we define the notion of *context ontologies*, which are intended to make context explicit for each concept of a domain ontology. The difficulties raised when agreeing on a shared representation of a domain knowledge remain, but they are now identified and isolated due to explicit and separate description of context heterogeneities. Thus, context heterogeneities can be handled because the contexts of the participants

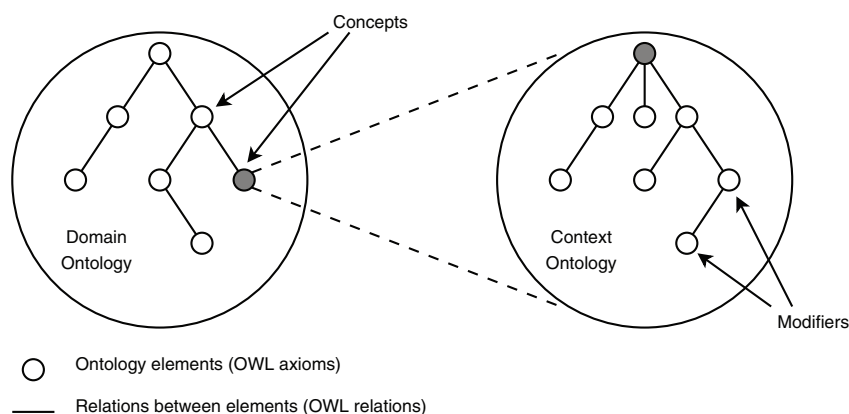


Fig. 2. Context versus domain ontologies.

are made explicit in context ontologies. Rich description languages like OWL allow specifying complex relations between ontology concepts. The separation between context and domain ontologies is illustrated in Figure 2 in order to clarify the terminology used for context description.

As context ontologies provide shared vocabularies to specify structural and semantic representations of context, there is a need to extensionally specify the values that modifiers take. We recall that modifiers are either static or dynamic. Consequently, we develop two different solutions to instantiate modifiers. We insert a description of static modifiers into WSDL in a way that remains compliant with the Web services protocol stack [Mrissa et al. 2006a]. Descriptions of static modifiers provide the means to calculate dynamic modifiers at runtime, using appropriate inference rules.

The use of context ontologies and WSDL annotation helps providers make explicit the context of data. It provides a scalable solution to integrate context into the Web services protocol stack. Moreover, it enables semantic mediation of data during the execution of a composition. In the following, we give an overview of our solution, detailed in previous work [Mrissa et al. 2006b], for annotating descriptions of composed Web services.

4.3 Extending WSDL with Context

Using the model of Section 4.1 requires enriching the description of Web services with context, by annotating WSDL message parts so that they can be now considered as semantic objects. In WSDL descriptions, `<message>` elements describe data exchanged for an operation. Each message consists of one or more `<part>` elements. We also refer to `<part>` elements as “parameters” in the rest of this article. Each parameter has a `<name>` and a `<type>` attribute, and allows additional attributes. Our annotation takes advantage of the extension proposed in the WSDL specification [Christensen et al. 2001], so that annotated WSDL documents operate seamlessly with both classical and annotation-aware clients. To keep the article self-contained, we overview a simplified structure of the annotated WSDL metamodel in Figure 3.

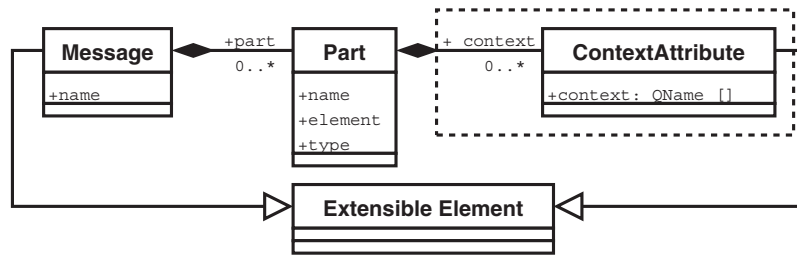


Fig. 3. Partial representation of the extended WSDL metamodel.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>...
  <wsdl:message name="CarRentalTicket">
    <wsdl:part name="inputPrice" type="xsd:double"
      ctxt:context="dom1:Price ctxt1:France
      ctxt1:VATIncluded ctxt1:ScaleFactorOne"/>
  </wsdl:message>...
</wsdl:definitions>
  
```

Listing 1. Car rental annotation snippet.

Specifically, `<part>` elements are annotated with a context attribute that describes the names and values of static modifiers using a list of qualified names. The first qualified name of the list specifies the domain ontology concept of the value (c). Additional elements refer to instances of static modifiers described in the context ontology. Listing 1 illustrates the proposed extension with our car rental Web service of Section 3.1. The annotation provides the values of the static modifiers for this Web service.

Relying on this annotation, a value v and its data type t described in the WSDL document are enriched with the concept c and the necessary modifiers to define the context C , thus forming a semantic object $\langle c, v, t, C \rangle$. In order to complete the context C , rules help infer the values of dynamic modifiers at runtime. Using rules offers several advantages: Rules are easily modifiable, making this solution adaptable to changes in the underlying semantics. In addition, the often-changing values of modifiers cannot be statically stored, so using rules simplifies the annotation to WSDL. Furthermore, rules separate application logic from the rest of the system, so updating them does not require rewriting application code. In the following, we detail our context mediation architecture that integrates mediators into composition as Web services, and show its internal functioning, which relies on rule-based mechanisms. Our mediation architecture aims at reconciling Web services at the semantic level.

5. A CONTEXT-BASED MEDIATION ARCHITECTURE

In this section, we present a context-based mediation architecture that takes advantage of the features offered with the aforementioned context model. First, we discuss the advantages of service-based integration of mediators into the composition. Second, we give an overview of the proposed architecture with a WS-BPEL business process based on the example of Section 3.1, before detailing

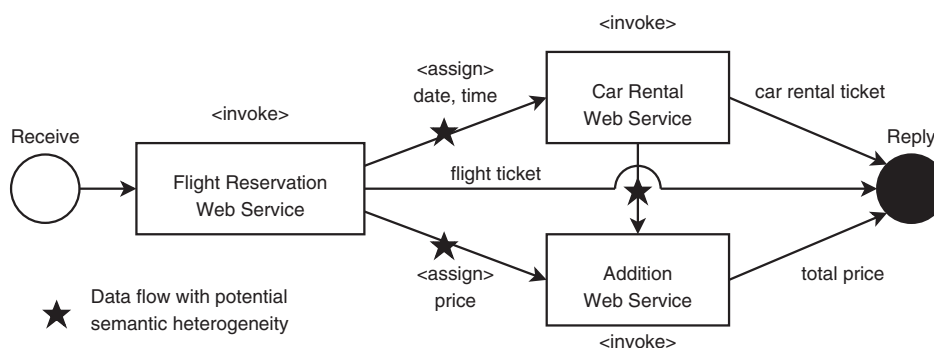


Fig. 4. View of the original business process.

the generation of contextualized business processes that include mediator Web services.

5.1 Advantages of Service-Based Mediation

The solution proposed in this article follows a service-based approach to implement mediators, referred to in the following as *mediator Web services*. Service-based mediation presents several advantages. First, the standardized access to Web services through their WSDL descriptions allows better independence from composition languages and engines. Second, managing the mediation concern in a service-oriented way is more scalable because it does not require extending any language nor modifying existing composition architectures, but rather reusing already-deployed software components. Third, the loosely coupled aspect of a service-based architecture allows keeping the mediation concern independent from the original functionalities of Web services.

However, the main issue is the need to adapt the input and output data of mediator Web services to the data representation expected by the Web services with which they communicate. We answer this limitation in the following and propose a solution that can be applied as a predeployment step during the deployment of a composition.

5.2 Overview of the Mediation Process

For consistency purposes, we use the example of Section 3.1 in order to illustrate the mediation process. The WS-BPEL process depicted in Figure 4 implements the composition logic of the workflow shown in Figure 1. It has been previously demonstrated that several heterogeneities hamper the correct execution of this workflow. We assume that the WSDL files used in this article are correctly annotated with context information [Mrissa et al. 2006b], and that the corresponding domain and context ontologies are available. Our mediation approach is a three-step process, as shown in Figure 5.

—*Contextualization Step.* A contextualization algorithm (Section 5.3) analyzes the WS-BPEL process (i.e., as shown in Figure 4) to locate explicit and

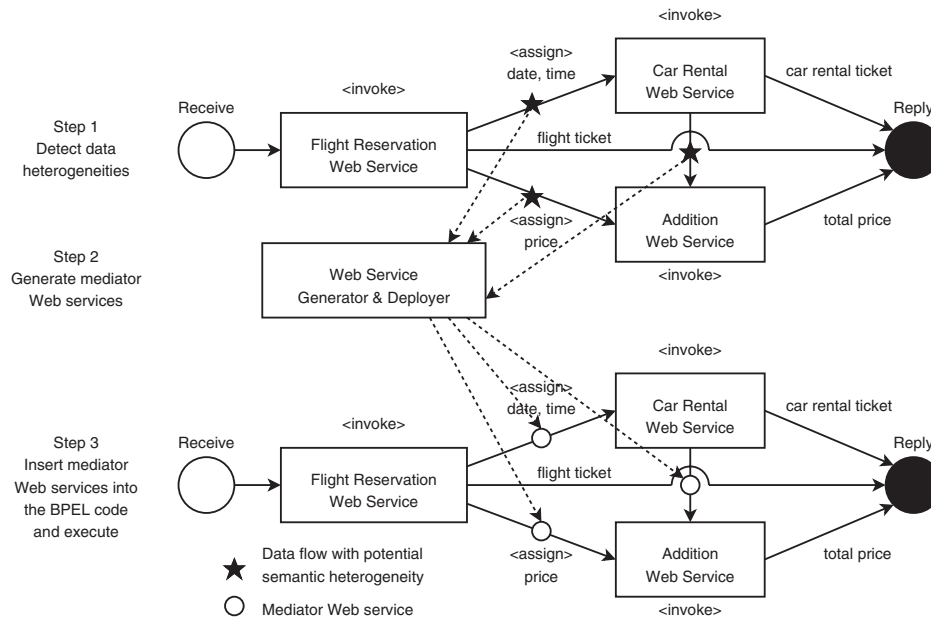


Fig. 5. Generation of the mediator Web service.

implicit data flows. Relevant data flows for possible context heterogeneities are symbolized with stars in the figure. In our running example, the data flows of interest are: (a) where the price variables are sent to the addition Web service, and (b) where the date and time variables are sent to the car rental Web service.

—*Automatic Mediator Web Services Generation Step.* Here, a mediator Web service is automatically generated and deployed for each data flow where the contextualization algorithm detected possible context heterogeneities. The generation of mediator Web services is handled by our Web service code generator (WS-CG), to be described in Section 6.1. Each generated mediator Web service implements an operation, named *mediateX2Y*, where *X* and *Y* are replaced with the names of the input and output messages of operations specified in the WSDL files related to the mediator Web service. Inputs to the mediation operation are the values that have to be transformed into the representation specified in context annotations. Outputs of the operation are the transformed values that have the expected meaning for the target operation in the composition. Details on the runtime generation and deployment of mediator Web services are given in Section 6.1.

—*Updating Original Composition Process Step.* Invocations to the generated mediator Web services from step 2 are inserted into the original WS-BPEL code according to Algorithm 1, presented next. The new WS-BPEL code uses the endpoints of dynamically generated mediator Web services combined with required `<invoke>` elements. A general template for mediator Web service invocation is depicted in Listing 2. After replacing all implicit and explicit data

```

<sequence>
  <assign>
    <copy>
      <from variable="source_ncname" part="ncname" />
      <to variable="mediation_input_ncname" part="ncname" />
    </copy>
  </assign>
  <!-- Call to the mediator Web service here-->
  <invoke name="mediation" partnerLink="mediator"
    portType="cm:ContextMediator" operation="mediate"
    inputVariable="mediation_input_ncname"
    outputVariable="mediation_output_ncname" />
  <assign>
    <copy>
      <from variable="mediation_output_ncname" part="ncname" />
      <to variable="destination_ncname" part="ncname" />
    </copy>
  </assign>
</sequence>

```

Listing 2. Mediator invocation in WS-BPEL.

flows, the contextualized WS-BPEL process is ready to be executed by any usual WS-BPEL execution engine. During the execution of the contextualized WS-BPEL process, mediator Web services are invoked to handle the semantic heterogeneities of data. The different steps performed by mediator Web services are summarized in Section 6.2.

5.3 Dynamic Generation of Contextualized Processes

WS-BPEL does not always make the data flows in a composition explicit, since it is meant to be “programming at large.” Data flows in WS-BPEL are encapsulated in `<variable>` elements. We distinguish between implicitly shared data between partners or explicitly shared data copied using `<assign>` elements. Our approach consists in locating implicit and explicit data flows and replacing them with invocations to mediator Web services.

First, let us consider explicit data flows described with `<assign>` elements. Such elements contain one or more `<copy>` elements, themselves containing a `<from>` element and a `<to>` element that, respectively, describe where data comes from and where it goes. Mediation is concerned with elements that are assigned from one variable to another variable only. We assume that the semantics of data entered manually by the composition designer (expressions or literal values) matches the semantic requirements of the business process. To integrate mediator Web services into WS-BPEL, we replace selected `<assign>` statements with an invocation to a generated mediator Web service, with the sequence depicted in Listing 2.

A `<sequence>` element allows consecutive execution of the contained elements by first building the input message for the mediator invocation by using an `<assign>` activity. Secondly, the mediator is invoked using an `<invoke>` element followed by an `<assign>` activity to extract the mediated data from the output message to the destination variable. By replacing the original `<assign>` element with the WS-BPEL code presented in Listing 2, the mediator Web service is inserted into the WS-BPEL composition to intercept and adapt the data flow. The generation of mediator Web services from the WSDL descriptions of the source and target Web services is described in detail in Section 6.1.

Algorithm 1. BPEL Contextualization Algorithm

```

1: for all assign ∈ findElements("assign")
2:   in ← getFromElement(assign)
3:   out ← getToElement(assign)
4:   if in ∈ "variable" ∧ out ∈ "variable"
5:     newAssign ← createMediationSeq(in, out)
6:     replace(assign, newAssign)
7:   end if
8: end for
9: for all seq ∈ findElements("sequence")
10:  for all (a, b) ∈ (getInvokeChildren(seq)2)
11:    if getOutputVar(a) = getInputVar(b) ∧ isBefore(a, b)
12:      mediationCode ← createMediationSeq(getOutputVar(a), getInputVar(b))
13:      insertBefore(b, mediationCode)
14:    end if
15:  end for
16: end for
17: for all flow ∈ findElements("flow")
18:  for all (a, b) ∈ (getInvokeChildren(flow)2)
19:    if getOutputVar(a) = getInputVar(b)
20:      if a.hasChild("source") ∧ b.hasChild("target")
21:        src ← a.getChild("source")
22:        target ← b.getChild("target")
23:        if getLinkName(src) = getLinkName(target)
24:          mediationCode ← createMediationSeq(getOutputVar(a), getInputVar(b))
25:          mediationCode.getChild("sequence").append(b)
26:          replace(b, mediationCode)
27:        end if
28:      end if
29:    end if
30:  end for
31: end if

```

To handle implicit data flows, we need to locate shared variables, namely, variables that are first used as output of an `<invoke>` element, and then directly used as input of another consecutively executed `<invoke>` element. In WS-BPEL this situation happens in the following cases: (1) A `<sequence>` element contains several `<invoke>` child elements, and (2) a `<flow>` element contains several `<invoke>` child elements that are bound together through a `<link>` element. Algorithm 1 shows the detection in WS-BPEL of the explicit and implicit data flows described previously, and the modifications performed to insert the mediation code described in Listing 2.

The first part of the algorithm (lines 1 to 8) detects explicit data flows described with `<assign>` elements. The *findElements* function is used to locate all the assign elements. Then, `<from>` and `<to>` child elements are extracted from each `<assign>` element (lines 2 and 3) and if both are variables (line 4), they are used by the *createMediationSeq* function to create the mediation code (line 5) that replaces the former `<assign>` element (line 6).

Lines 9 to 16 show the detection of consecutive `<invoke>` elements in a sequence. Function *getInvokeChildren(sequence)* gets the `<invoke>` child elements that belong to the same *sequence*. The *getInputVar(invoke)* and *getOutputVar(invoke)* functions extract the information contained in the *inputVariable* and *outputVariable* attributes of the selected `<invoke>` element. Function *isBefore(a, b)* verifies that element *a* is executed before element *b* in the WS-BPEL code. So, if two `<invoke>` elements of a sequence (line 9 and 10) have matching output and input variables and are in the right execution order (line 11), the mediation code (line 12) is inserted just before the second `<invoke>` operation with the *insertBefore* function (line 13), so that mediation is only performed if necessary. In effect, it should be noted that other elements such as `<switch>` may change the execution of the workflow.

Lines 17 to 31 show the detection of related `<invoke>` elements in a flow. The algorithm identifies `<invoke>` elements that have identical *inputVariable* and *outputVariable* attributes, which possibly characterizes an implicit data flow (lines 17 to 19). These elements have to be related to each other by containing a `<source>` and `<target>` children that have the same *linkName* attribute (lines 20 to 23). In this case, the generated mediation code (line 24) includes the second `<invoke>` element (line 25) that is added with the *append* function. So, it replaces the original `<invoke>` element with a sequence including both the mediation code and the original invocation.

The aforesaid algorithm is essential to generate the contextualized WS-BPEL, which weaves the mediation concern into the original business process by including calls to mediator Web services generated on-the-fly.

6. IMPLEMENTATION WORK

A prototype has been developed as a proof-of-concept of the feasibility of this architecture under the Java™ environment. It includes several components. A graphical user interface enables providers to annotate WSDL files with context. A model-driven Web service generator deploys mediator Web services in Axis2 runtime. A mediator Web service has been implemented which reads-in context annotation from WSDL files and converts data from a source context to a target context. The contextualization algorithm for WS-BPEL processes has also been implemented.

6.1 Model-Driven Web Service Generation

During the contextualization step of the original WS-BPEL process, one or several mediator Web services need to be generated. Therefore, we implemented a flexible and lightweight model-driven code generator based on a platform-independent object model (IOM). Based on the IOM, we implemented a platform-specific model (PSM) for the Java platform.

The main elements of the code generator are shown in Figure 6. A model description specified in XML acts as an input to the code generator. The input file is parsed by an `XMLImporter` component, which builds the IOM corresponding to the XML description of the model. The IOM is then transformed into PSM-oriented Java. The `JavaExporter` component operates directly on the PSM

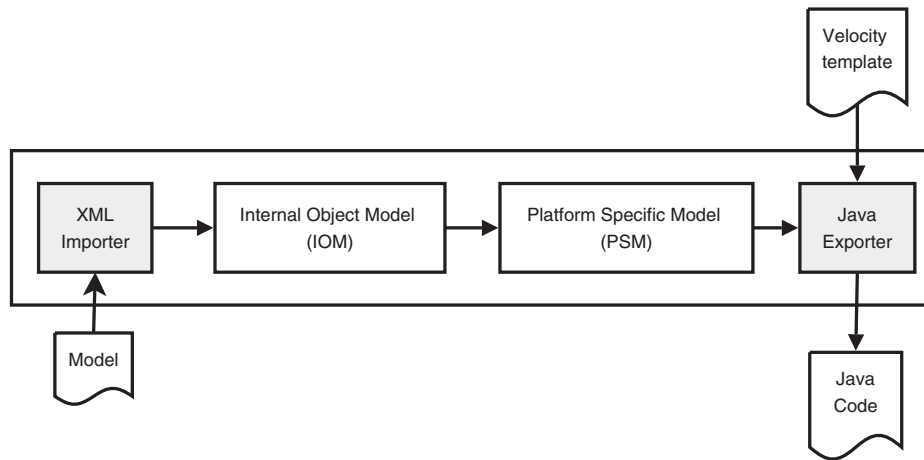


Fig. 6. Model-driven Web service generator.

and iterates through every class and interface to generate the Java code. The actual code generation is supported by the Apache Velocity template engine [Apache Software Foundation 2006b]. Based on this model-driven code generator, we implemented a special `Axis2CodeGenerator` component for generating a Web service for the new Axis2 [Apache Software Foundation 2006a] runtime, which executes the following steps as next described.

- (1) *Dynamic Code Generation and Compilation.* The aforementioned code generator is used to dynamically generate and compile a Java class which will be deployed as Web service. All libraries for compiling the code are dynamically added to the classpath.
- (2) *Deployment Descriptor Generation.* Axis2 requires a special deployment descriptor called `services.xml` which specifies the main class implementing the service logic. Additionally, each operation of the class that should be exposed as a Web service operation has to be specified. Axis2 implements purely document-style Web services by leveraging a top-down (or “WSDL-first”) approach. Due to the fact that we generate the implementation of the Web service directly, we use a bottom-up approach, which is typically used for RPC-style Web services. Therefore, we use a special message handler, called `RPCMessageReceiver`, which is specified in the deployment descriptor. It is responsible for converting the document-style nature of a Web service as required by Axis2 to the RPC structure that we use internally (by exposing a Java class as a Web service).
- (3) *Code Packaging and Deployment.* The compiled code together with the deployment descriptor and the required libraries are packaged together into an `.aar` file (Axis2 archive). The new deployment model of Axis2 allows a very simple deployment step. The archive file created in the previous step is simply copied to the Axis2 deployment directory (specified via properties in our system). The deployment itself is then handled by the Axis2 runtime.

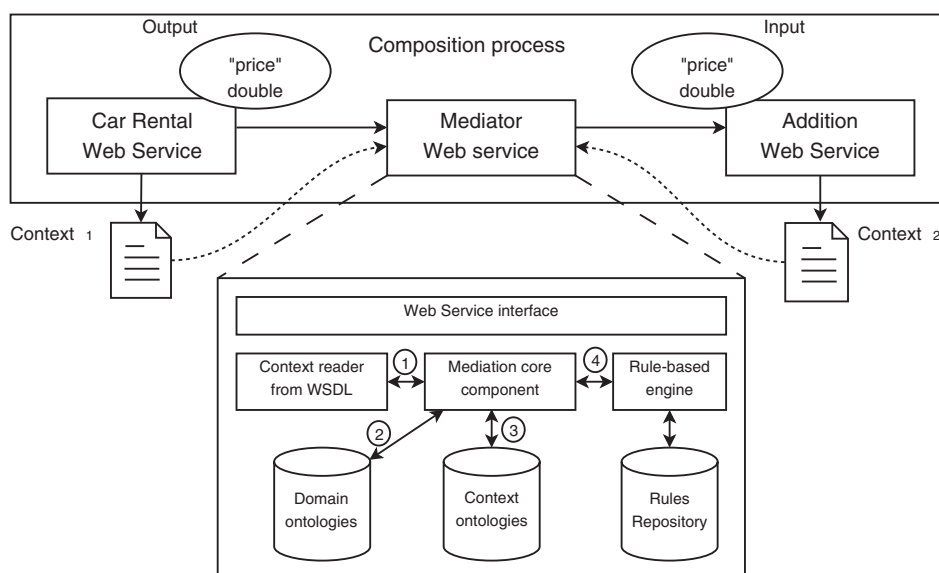


Fig. 7. Detailed view of the mediator Web service.

On the basis of the `Axis2CodeGenerator`, we developed a component named `MediationServiceBuilder` that currently builds the mediator Web service based on the WSDL files and the operation names it gets as an input. This input data defines the services and data types for which a mediator has to be generated. Based on this input data, it dynamically builds a model for the Web service, consisting of a Java class and one operation. The implementation of the mediator operation is presented in detail in the next section.

One important aspect about the lifecycle of the mediator Web service is the fact that it automatically gets undeployed if the corresponding WS-BPEL process is undeployed. At a technical level this is achieved by registering a deployment listener to the WS-BPEL engine to receive notifications about the deployment or undeployment of various processes. If the `MediationServiceBuilder` receives such a notification, it has to determine the corresponding mediator Web service and undeploy it from the Axis2 runtime.

6.2 Operation of Mediator Web Services

In this section, we detail the internal operation of mediator Web services. The latter are inserted into composition processes between original Web services that may have context heterogeneities. We discuss the mediation steps performed by mediator Web services with the example of this article. We consider the data flow from the car rental Web service to the addition Web service, passing through the mediator Web service. This one takes as input a *price* message part sent by our car rental Web service, and then performs the steps described in Figure 7 before sending the result of the computation, in a *price* message part, to the addition Web service.

In step one, the WSDL files of the car rental and addition Web services are fetched by the mediator Web service. Then, these files are parsed in order to extract the required elements. In particular, we are interested in the annotations of the message parts that the mediator receives and sends. In our running example, these are the output and input *price* message parts of the car rental and addition Web services, respectively. Extracted annotations refer to the domain concept and the semantic properties necessary to obtain a correct data interpretation. An example of such an annotation of the car rental Web service was depicted in Listing 1.

In step two, the mediator Web service identifies the exchanged concepts in domain ontologies. The annotation is a list of attributes, and the first annotated attribute always refers to the domain concept. Here, the annotated message part refers to the *price* concept of the domain ontology identified with the namespace *dom1* in the WSDL file. The mediator Web service checks that the concepts of both the car rental and addition Web services match, namely, that they verify a subsumption or equivalence relation. This is a simple approach to semantic matching, but additional capacities can be integrated into the mediator. For a good survey on semantic integration techniques, see Noy's work [2004].

In step three, an in-memory tree is built from the context ontology to represent the context related to the *price* concept. The annotated context attributes corresponding to the *price* concept are identified and their values added to the tree. Annotated context attributes refer to OWL individuals (instances), so they describe not only modifiers, but also the values they take in the context of this Web service. In Listing 1, the *ctxt1:France* attribute is an instance of the *country* concept in the context ontology. Moreover, the *ctxt1:ScaleFactorOne* attribute is an instance of the *scaleFactor* concept in the context ontology. We assume that providers of Web services correctly add this information to the context ontology before annotating WSDL files.

In step four, the mediator Web service communicates with a rule engine to perform several tasks. The first task consists in inferring the values of dynamic modifiers that are part of the context, using rules stored in the knowledge base. For example, being given (*country = France*) as a fact, the rule engine infers (*currency = euro*) by querying the knowledge base. Thus, the *currency* modifier is being given the value *euro*.

The second task consists in converting received data into the required context representation. From the previous steps, we obtained two in-memory context representation trees that have valued or not-valued elements. Next, the mediator compares each element of the context and queries the rule engine to see if the contained values are convertible. If a value is missing, the mediator queries the knowledge base for a rule that specifies a default value. If no such rule exists, the conversion is canceled and an error is thrown.

The knowledge base also contains conversion rules that allow dynamic conversion between context values. For instance, the *price* in our example is converted into the required currency by calling a remote component that provides an up-to-date currency conversion rate. Such conversion needs to be dynamic so that it can answer the requirements of the temporal perspective of context.

6.3 Test Case and Evaluation

A test experiment has been conducted on the basis of the example developed in this work. Our current composition example is hosted by an Apache Tomcat container, and our mediator Web service implementation uses Jena 2 API and Drools rule engine to access and manipulate OWL ontologies and perform data conversion.⁵ The prototype includes illustrative domain and context ontologies for describing the required concepts and contexts.⁶

The implementation performs at-runtime context mediation, enabling meaningful execution of composition. In the example of this article, not only do the *price* concepts match, but data is transformed at runtime to comply with the different scale factors, heterogeneous date formats (that allow getting up-to-date conversion rates between currencies), and VAT rates (that also are not always included in the price) described in the context ontology.

As future work, we envision further practical tests and performance evaluation. However, such experiments require additional domain and context ontologies that must be validated by domain experts, as well as additional sets of conversion rules and functions. Therefore, and for the purpose of this article, we limited our experiments to the test case developed previously as a proof-of-concept of the feasibility of our architecture. Ongoing work also concerns integration of the contextualization algorithm with WS-BPEL implementations such as ActiveBPELTM or Apache OdeTM.

7. CONCLUSION

In this article, we presented a context-based approach for semantic Web services composition. The approach revolves around the following aspects: annotating WSDL descriptions so that Web services are now described with contextual details, deploying a context-based mediation architecture so that implicit assumptions on data flow are made explicit, and automatically generating and invoking Web service mediators so that the data heterogeneities between Web services are handled during the composition.

Future work aims at looking into the following issues. First, unexpected changes in some Web services' nonfunctional properties could lead to substituting some Web services with others offering the same functionality. The challenge is to make this substitution automatic, dynamic, and transparent. Second, Web service discovery and selection steps could ease semantic mediation during the composition step. The challenge is to ensure that semantic mediation is taken into account during discovery and selection stages.

REFERENCES

- ANDREWS, T., CURBERA, F., DHOLAKIA, H., GOLAND, Y., KLEIN, J., LEYMAN, F., LIU, K., ROLLER, D., SMITH, D., THATTE, S., TRICKOVIC, I., AND WEERAWARANA, S. 2003. Business process execution language for web services (BPEL4WS), version 1.1.
- APACHE SOFTWARE FOUNDATION. 2006a. Axis2. <http://ws.apache.org/axis2/> (last accessed: May 29, 2006).

⁵<http://tomcat.apache.org/>, <http://jena.sourceforge.net/> and <http://www.drools.org/>.

⁶<http://www710.univ-lyon1.fr/~mmrissa/>

- APACHE SOFTWARE FOUNDATION. 2006b. Velocity—Java-Based template engine. <http://jakarta.apache.org/velocity/> (last accessed: May 29, 2006).
- ARROYO, S. AND STOLLBERG, M. 2004. WSMO primer. WSMO deliverable D3.1, DERI working draft. Tech. Rep., WSMO. <http://www.wsmo.org/2004/d3/d3.1/>.
- BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSON, N., NIELSEN, H. F., THATTE, S., AND WINER, D. 2000. Simple object access protocol (SOAP) 1.1. Tech. Rep., W3C.
- CABRAL, L. AND DOMINGUE, J. 2005. Mediation of semantic web services in IRS-III. In *Proceedings of the Joint 1st International Workshop on Mediation in Semantic Web Services (MEDIATE) and 3rd International Conference on Service Oriented Computing (ICSOC)*, Amsterdam, The Netherlands.
- CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. 2001. Web services description language (WSDL) 1.1, W3C note. Tech. Rep., W3C. March.
- FENSEL, D. AND BUSSLER, C. 2002. The web service modeling framework WSMF. Tech. Rep., Vrije Universiteit Amsterdam, The Netherlands.
- GRUBER, T. 2000. What is an ontology? <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>.
- KASHYAP, V. AND SHETH, A. P. 1996. Semantic and schematic similarities between database objects: A context-based approach. *VLDB J* 5, 4, 276–304.
- KLEIN, M., KÖNIG-RIES, B., AND MÜSSIG, M. 2005. What is needed for semantic service descriptions - A proposal for suitable language constructs. *Int. J. Web Grid Serv* 1, 3-4, 328–364.
- MAAMAR, Z., BENSLIMANE, D., AND NARENDRA, N. C. 2006. What can context do for web services? *Commun. ACM* 49, 12, 98–103.
- MARTIN, D. L., PAOLUCCI, M., MCLRAITH, S. A., BURSTEIN, M. H., McDERMOTT, D. V., MCGUINNESS, D. L., PARSIA, B., PAYNE, T. R., SABOU, M., SOLANKI, M., SRINIVASAN, N., AND SYCARA, K. P. 2004. Bringing semantics to web services: The OWL-S approach. In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, J. Cardoso and A. P. Sheth, eds. Lecture Notes in Computer Science, vol. 3387. Springer, 26–42.
- MEDJAHED, B. AND BOUGUETTAYA, A. 2005. A dynamic foundational architecture for semantic web services. *Distrib. Parallel Databases* 17, 2, 179–206.
- MILLER, J., VERMA, K., RAJASEKARAN, P., SHETH, A., AGGARWAL, R., AND SIVASHANMUGAM, K. 2004. WSDL-S: Adding semantics to WSDL (white paper). Tech. Rep., Large Scale Distributed Information Systems. <http://ldis.cs.uga.edu/library/download/wsd-s.pdf>.
- MOCAN, A., CIMPIAN, E., ZAREMBA, M., AND BUSSLER, C. 2004. Mediation in web service modeling execution environment (WSMX). In *Information Integration on the Web (IIWeb)*, Toronto, Canada.
- MRISSA, M., GHEDIRA, C., BENSLIMANE, D., AND MAAMAR, Z. 2006a. A context model for semantic mediation in web services composition. In *Proceedings of the 25th International Conference on Conceptual Modeling*, D. W. Embley et al. eds. Lecture Notes in Computer Science, vol. 4215. Springer, 12–25.
- MRISSA, M., GHEDIRA, C., BENSLIMANE, D., AND MAAMAR, Z. 2006b. Towards context-based mediation for semantic web services composition. In *Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, San Francisco, CA.
- NOY, N. F. 2004. Semantic integration: A survey of ontology-based approaches. *SIGMOD Rec.* 33, 4, 65–70.
- NOY, N. F. AND HAFNER, C. D. 1997. The state of the art in ontology design: A survey and comparative review. *AI Mag.* 18, 53–74.
- NOY, N. F. AND MCGUINNESS, D. 2000. Ontology development 101: A guide to creating your first ontology. Tech. Rep. KSL-01-05, Stanford University, California.
- PEER, J. 2005. Semantic service markup with SESMA. In *Web Service Semantics Workshop (WSS) at the 14th International World Wide Web Conference (WWW)*.
- SAWSDL WORKING GROUP. 2006. Semantic annotations for WSDL, W3C working draft. Tech. Rep., W3C. September.
- SCHREIBER, G. AND DEAN, M. 2004. OWL web ontology language reference. <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
- SCIORE, E., SIEGEL, M., AND ROSENTHAL, A. 1994. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Trans. Database Syst.* 19, 2, 254–290.

- SPENCER, B. AND LIU, S. 2004. Inferring data transformation rules to integrate semantic web services. In *Proceedings of the International Semantic Web Conference*, S. A. McIlraith et al. eds. Lecture Notes in Computer Science, vol. 3298. Springer, 456–470.
- UDDI WORKING GROUP. 2001. Universal description, discovery, and integration of business for the web. <http://www.uddi.org>.
- W3C. 2004. XML schema part 2: Datatypes second edition. Tech. Rep., W3C. October. <http://www.w3.org/TR/xmlschema-2/>.
- WIEDERHOLD, G. 1992. Mediators in the architecture of future information systems. *IEEE Comput.* 25, 3, 38–49.
- WILLIAMS, A. B., PADMANABHAN, A., AND BLAKE, M. B. 2005. Experimentation with local consensus ontologies with implications for automated service composition. *IEEE Trans. Knowl. Data Eng.* 17, 7, 969–981.

Received June 2006; revised January 2007; accepted April 2007