Ontology Based Cross-Domain Enterprises Integration and Interoperability

Yue Ni\textsuperscript{1,2}, Yushun Fan\textsuperscript{1}
\textsuperscript{1} Department of Automation, Tsinghua University, Beijing 100084, China
\textsuperscript{2} Guilin Air Force Academy, Guilin 541003, China
E-mail: niy04@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn

Abstract

Web services are gaining momentum as key elements in cross-domain enterprises integration because more and more functions within intra- and inter-enterprises have been encapsulated as Web services. However, these services are typically provided and invoked by different service providers in different enterprises distributed in multi-domains, so it is inevitable to face the problem of semantic inconsistency during enterprise integrating and cooperating. In this paper, ontology is brought out to solve this problem by building reference ontologies corresponding to each domain, and then collaboration ontologies are constructed semi-automatically, whereafter OWL-S files generated from collaboration ontologies are mapped to BPEL and WSDL files respectively. By this way, the semantic information will be kept in processes and Web services, so that there is a common understanding among cross-domain cooperating enterprises. The OCDEII system architecture is proposed to support such integration and interoperability and further a prototype system is implemented in Project ImportNET.

1. Introduction

Nowadays, the management of processes and information between cross-domain collaborating companies usually lacks sufficient knowledge exchange and sharing. Therefore the following challenges need to be addressed: the connection of proprietary systems; the sharing of knowledge crossing company borders, cultures and engineering domains; and the common understanding of data and knowledge among cooperating companies in order to assure a smooth and flexible collaboration process. Among above, the most critical issues are data integration and interoperability which need the “understanding” of processes in different enterprises with different cultures and domains.

In order to transform the provided “data” into “information” or even “knowledge”, it is necessary to use the same semantics and all the partners have a common understanding. This becomes more difficult in case of cross-cultural and cross-domain collaborations where the participants might have different semantic interpretations. Without an overarching structure to manage changes and provide timely communication during the product development cycle, engineers in different domains may find themselves working on the wrong versions, and using out of date information, without “single source of truth” to guide and coordinate their actions. Consequently a precondition for sharing common knowledge is to create a common understanding within the collaboration.

Therefore, the integration and interoperability of the enterprises in different domains have been brought out to gain the goal of collaboration. Ontology defines a common vocabulary for stakeholders who need to share information in one domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. Ontology represents the structured depictions or models of known (and accepted) facts about some topics or some subject areas and provides the logical statements that describe what the terms are, how they are related to each other, as well as the rules for combining terms and relations to define extensions to the vocabulary. So ontologies are a suitable “single source of truth” to guide and coordinate actions of the engineers in different domains [1].

Some researchers proposed semi-automatically annotated method based on the process ontology and the domain ontology to target the semantic interoperability issue in process models discovery [2]. Taking [2] for reference, we bring out a collaboration architecture using ontology based workflow to string all the functions of cross-domain enterprises which already have been encapsulated as web services [3].

Suppose there are two domains: mechanics and electronics, and two enterprises \textit{M} and \textit{E}, which belong to these two domains respectively. \textit{Mike} is a mechanical engineer in enterprise \textit{M} and \textit{Eric} is an electronic engineer in enterprise \textit{E}. They need to cooperate to design a kind of hub which is a common connection point for devices in a network. When Mike changed the position of a hole in the main circuit board, Eric should know the change in order to make some changes in the design of corresponding circuits. The problem is that they are in different domains and may not understand each other.

The example is quite simple but illustrates the key problem that we want to solve in this paper. Our method is, first of all, construct OWL-S based collaboration ontologies according to the reference ontologies from different domains, then transform them to BPEL and WSDL documents in order to achieve a semantic enabled
process, and thus Mike and Eric can understand each other and the design process can proceed automatically.

Our work is an effort to reach the process collaboration and knowledge sharing between cross-domain and cross-culture enterprises, and also roughly achieve the transformation from OWL-S to BPEL and WSDL. The processes abstracted from ontologies have semantic attributes, which make it possible that the discovery, deployment and management of web services could be realized automatically. Some related work about ontology based integration and interoperability did not bridge ontologies to processes directly, so the semantic information within ontologies cannot be brought to processes and Web services efficiently. We will elaborate it in Section 7.

This paper is organized as follows. In Section 2 the architecture for enterprise integration and interoperability is proposed. The mapping from OWL-S to BPEL and the mapping from OWL-S to WSDL are presented in Section 3 and 4. The BPEL based workflow engine is depicted in Section 5. In Section 6, a prototype that implements our architecture is described. Finally the related work and conclusions are given in Section 7 and 8.

2. OCDEII System Architecture

We present an architecture to delineate the Ontology based Cross-Domain Enterprises Integration and Interoperability (OCDEII), as seen in Figure 2.

As groundwork, the reference ontologies are developed from the specific domain. In our case it is the mechanical ontology and electronical ontology built by Applied Logic Laboratory in Hungary and Salzburg Research in Austria, and both of them are partners in project ImportNET [3]. It is supposed that the reference ontology includes two main segments: one describes the design process (process ontology), and the other describes the universe of the domain (artifact ontology).

The reference ontology may be developed by various methodologies and different kinds of top ontologies may be used. Presently, the mechanical reference ontology is based on the Extended Description and Situation (D&S) module of the DOLCE top ontology library [4]. In order to keep the generality, the ontologies of different styles are transformed to a standard form which also uses the DOLCE ontology library, but only the top DOLCE Lite module.

Reference ontology is in the format of *.owl. It is created by Protégé ontology editor 3.4 beta and generated semi-automatically. Figure 1 shows the screenshot of the reference ontology editor. It only includes the ontology items that are selected or added by the end users or generated by Ontology Generator Tool (OGT).

Concepts of the reference ontology are selected automatically or manually. Both deletion and selection are realized by labeling concepts in the reference ontology. The selected concepts are integrated into the complete collaboration ontology automatically. The automatic process of integration is controlled by options based on information added during the manual selection. After integration, the rough collaboration ontology will be customized: more ontology items could be added into it from the reference ontology, or ontology items can be deleted. Moreover, ontology items not included in the reference ontology can be added [3].

When the collaboration ontology is generated, the concepts, instances, axioms that don’t occur in the reference ontology may be added to the collaboration ontology.

Taking collaboration ontology as input, OWL2BPEL and OWL2WSDL Translator output *.bpel and *.wsdl files respectively so that corresponding processes and web services can be deployed on Workflow Engine automatically or manually. Presently the deployment of processes is done manually. We will illustrate the OWL2BPEL and OWL-S2WSDL in Section 3 and 4 respectively, moreover the Workflow Engine in Section 5.

An engineer in one domain sends a request to Cross-Domain User Interface (CDUI) and then CDUI encapsulates the request into a SOAP message that can be read by BPEL based workflow engine. Whereafter the Workflow Engine receives the request and initiates corresponding process according to the request and the process will be executed step by step through invoking Web services described by *.wsdl, then a reply to CDUI indicates the end of the process. The Web services invoked above can also invoke other web services and processes, also they may have messages interaction with CDUI when the requests are much more complex. CDUI provides special domain friendly interfaces for engineers from different domains.
All processes and Web services involved in this architecture are semantic enabled which make interactions clear not only in the definition phase but also in the execution phase. So they can get a common “understanding” no matter which domains the providers belong to. The advantage is prominent compared to others (details in Section 7).

The key components of OCDEII will be introduced in the next three sections.

3. OWL-S2BPEL

3.1 OWL-S

The Semantic Web vision is to make Web resources accessible by content as well as by keywords so that they can be understood by machine. Web services play an important role in this: users and software agents should be able to discover, compose, and invoke content using complex services. The DARPA Agent Markup Language (DAML) extends XML and the Resource Description Framework (RDF) to provide a set of constructs for creating machine-readable ontologies and markup information [5]. The DAML program’s Semantic Web contribution is the Web Ontology Language for Services (OWL-S, previously known as DAML-S). OWL-S is a services ontology that enables automatic service discovery, invocation, composition, interoperation, and execution monitoring [6]. OWL-S makes it happen that using Semantic Web techniques to automate dealings with Web services. It provides a framework for describing both the functions and advertisements for Web Services. In this paper, we use OWL-S version 1.2, OWL-S models services using a three-part ontology [7]:

• a profile describes what the service does;
• a process model and corresponding presentation syntax specifies how the service works; and
• a grounding gives information on how to use the service.

OWL-S API provides a Java API for programmatic access to read, execute and write OWL-S service descriptions. OWL-S’s exchange syntax is RDF/XML and many processors work with an RDF based model, in part, to facilitate the smooth integration of OWL-S service descriptions with other Semantic Web knowledge bases. OWL-S API was designed to help programmers to access and manipulate OWL-S service descriptions programmatically.

OWL-S is not intended to replace existing Web Services or Semantic Web standards. The goal is to enhance their utility by remaining compliant with existing standards and adding explicit semantics that is operational and understandable to computer programs.

3.2 BPEL

BPEL (BPEL4WS) is an XML language that supports process oriented service composition, now BPEL is the de facto standard for specifying business processes in a Web services world. In BPEL, the composition result is called a process, participating services are partners, and message exchange or intermediate result transformation is called an activity. A process thus consists of a set of activities. A process interacts with external partner services through a WSDL interface. To define a process, we use

• a BPEL source file (.bpel), which describes activities;
• a process interface (.wsdl), which describes ports of a composed service; and
• an optional deployment descriptor (.xml), which contains the partner services’ physical locations.

BPEL activities can be classified as primitive activities like <receive>, <reply>, <invoke> and <throw> which are used to describe interactions between business partners and structural activities like <sequence>, <flow>, <switch>, <while> and etc which are used to describe workflow in a BPEL process model.
3.3 Mapping from OWL-S to BPEL

Figure 3 illustrates the mapping specification from OWL-S to BPEL and WSDL. The service grounding and model are mapped to WSDL, while the service model is mapped to the primitive activity and structured activity in BPEL respectively according to the corresponding inner structure. We represent the mapping from OWL-S to BPEL in this section and the mapping from OWL-S to WSDL in section 4.2.

The semantics of BPEL is very limited especially compared with OWL-S. With the attributes of OWL-S and BPEL mentioned above, we present a mapping method from OWL-S to BPEL. The semantic attributes will be endowed to BPEL after mapping.

The mapping method from OWL-S to BPEL is shown in Figure 4. OWL-S has three kinds of processes, simple processes, atomic processes and composite processes. BPEL has two kinds of processes, abstract processes and executable processes. Abstract processes provide means of synchronization with other processes at various level of granularity for the purpose of planning and reasoning. Simple processes in OWL-S also play the same role as BPEL abstract processes by defining a level of abstraction. So we map simple processes to abstract processes. BPEL primitive activities can be used to perform a Web service operation by sending and receiving appropriate messages. OWL-S perform control construct is used to perform atomic process. So we map OWL-S perform control construct to BPEL primitive activities. OWL-S has sequence control construct, which is used to perform the child atomic or composite processes in a sequence. Structured Activities in a BPEL process model describe the order in which a set of the child primitive or structured activities is performed. For example structured activity describes that the child primitive or structured activities within a sequence activity are performed in a sequence. Due to their logical matching behavior, OWL-S control constructs within an OWL-S composite process are mapped to BPEL structured activities. A composite process is not a behavior that a service will do, but a behavior (or set of behaviors) the client can perform by sending and receiving a series of messages [5]. In the mapping from OWL-S composite processes to BPEL structrue activities, we map OWL-S sequence to BPEL sequence; split to flow; if-then-else to switch and repeatwhileto to while respectively.

Take the foregoing simplified collaboration scene to illustrate this mapping. Mike is a mechanical engineer and Eric is an electronical engineer. They are working together to design a kind of hub. When Mike changed the position of a hole in the main circuit board, Eric should know the change in order to make some changes in the design of corresponding circuits. Eric send a request message to CDUI, then the message transfers to workflow engine and initiates a process which will invoke some web services to fulfill the request from Eric and reply a result to Eric. The OWL-S statement of the scene is as follows:

```
...<process:CompositeProcess
  rdf:about="http://importnet.imi.uni-karlsruhe.de/importnet/requestdesignchange.owl#requestchange">
  <process:composedOf>
    <process:Sequence>
      <process:components>
        <process:ControlConstructList>
          <list:first>
            <process:Perform>
              <process:process
                rdf:resource="http://importnet.imi.uni-karlsruhe.de/importnet/requestdesignchange.owl#mechanicalchangerequestProcess"/>
            </process:Perform>
          </list:first>
          <list:second>
            <process:Perform>
              <process:process
                rdf:resource="http://importnet.imi.uni-karlsruhe.de/importnet/requestdesignchange.owl#mechanicalchangereplyProcess"/>
            </process:Perform>
          </list:second>
        </process:ControlConstructList>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
...`

The OWL-S statement is transformed to the BPEL message flow in Figure 4.
After mapping, we get the BPEL statement of the same scene:

```xml
<process xmlns="http://importnet.imi.uni-karlsruhe.de/importnet/requestdesignchange"
         xmlns:ns1="http://importnet.imi.uni-karlsruhe.de/importnet/requestdesignchange"
         xmlns:ns2="http://importnet.imi.uni-karlsruhe.de/importnet/requestdesignchange"/>

<partnerLinks>
  <partnerLink myRole="electronicalengineer"
               name="electronicalchangerequestLinkType"
               partnerLinkType="ns1:electronicalchangerequestLinkType"/>
  <partnerLink name="mechanicalchangereplyLinkType"
               partnerLinkType="ns2:mechanicalchangereplyLinkType"
               partnerRole="mechanicalengineer"/>
</partnerLinks>

<variables>
  <variable messageType="ns1:requestchangeMessage"
            name="requestchangeMessage"/>
  <variable messageType="ns1:returnchangeMessage"
            name="returnchangeMessage"/>
</variables>

<sequence>
  <receive operation="requestchange"
           partnerLink="mechanicalchangereplyLinkType"
           portType="ns1:mechanicalchangereplyPT"
           variable="requestchangeMessage"/>
  <invoke inputVariable="requestchangeMessage"
          operation="mechanicalchangerequest"
          outputVariable="returnchangeMessage"
          partnerLink="mechanicalchangereplyLinkType"
          portType="ns2:mechanicalchangereplyPT"/>
  <reply operation="requestchange"
          partnerLink="mechanicalchangereplyLinkType"
          portType="ns1:mechanicalchangereplyPT"
          variable="returnchangeMessage"/>
</sequence>
</process>
```

4. OWL-S2WSDL

Implementing a BPEL process in the form of *.bpel file format, which is based on ontology, is not enough to make the processes executable. We also need to build transformation rules for mapping between OWL based knowledge and WSDL definition.

4.1 WSDL

WSDL is a language with XML format to describe Web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bounded to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A WSDL document is simply a set of definitions. There is a definitions element at the root, and definitions inside. Services are defined using the following elements in WSDL 1.1 [8]:

- **types**, a container for data type definitions using some type system (such as XSD).
- **message**, an abstract, typed definition of the data communicated.
- **operation**, an abstract description of an action supported by the service.
- **portType**, an abstract set of operations supported by one or more endpoints.
- **binding**, a concrete protocol and data format specification for a particular port type.
- **port**, a single endpoint defined as a combination of a binding and a network address.
- **service**, a collection of related endpoints.

4.2 Mapping from OWL-S to WSDL

OWL-S and WSDL are interoperable but have differences. OWL-S specifies abstract type by OWL class while WSDL use XML Schema specifying abstract type. OWL-S doesn’t contain the binding information, whereas WSDL doesn’t include semantic information which has been embedded in OWL-S. So the WSDL file mapped from OWL-S could cover semantic and binding information simultaneously. Figure 5 shows the relationship between OWL-S and WSDL [9]. WSDL operations correspond to OWL-S atomic processes. The inputs and outputs of an OWL-S atomic process correspond to messages in WSDL. The grounding ontology of OWL-S is used to specify how the abstract information exchanges that are detailed by atomic process descriptions are realized by concrete information exchanges between the consumer requesting a service and the service provider that it has chosen to use [9]. We map each OWL-S atomic process to a WSDL operation (defining input and output messages), and map each OWL-S process input and output to WSDL elements of the input and output messages for the corresponding operation through grounding. By declaring these correspondences, the grounding efficiently provides a bridge between the syntax and protocol oriented world of WSDL, and the
world of OWL that relies on semantic elements defined by description logics.

![Diagram of OWL-S and WSDL](image)

**Figure 5. Relationship between OWL-S and WSDL**

Considering the scene in Section 3.3, the initiated process after receiving the request from Eric will invoke a Web service which is used to return the change made by Mike, and the corresponding WSDL is also mapped from OWL-S as follows:

```
<wsdl:message name="requestchangeMessage">
  <wsdl:part name="changelocation" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="requestService">
  <wsdl:operation name="request">
    <wsdl:input name="requestchange" message="tns:requestchange"/>
    <wsdl:output name="returnchange" message="tns:returnchange"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="requestServiceDocLitBinding" type="tns:requestService">
  <wsdl:operation name="request">
    <wsdl:input name="requestchange"/>
    <wsdl:output name="returnchange"/>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="requestServiceLitService">
  <wsdl:port name="requestService" binding="tns:requestServiceDocLitBinding">
  </wsdl:port>
</wsdl:service>
```

5. BPEL Based Workflow Engine

We use open source ActiveBPEL Community Edition Engine [10] as workflow engine to implement a BPEL process (*.bpel). BPEL not only provides a mechanism to orchestrate Web services, BPEL processes themselves are Web services when they are deployed in workflow engine. There are some key point steps for the workflow implementation in our architecture [3]:

- Constructing a WSDL definition (*.wsdl): It defines available functions and the mechanism for accessing BPEL processes. This covers the implementation details of a process regarding how to get it, what messages it uses, what operations it performs, and how to format and configure the input and output data;
- Constructing a PDD definition (Process Deployment Descriptor) (*.pdd): This file describes the details of service binding, and the process version for a BPEL process optionally. The service binding details
describe the relationship between the partners defined in a BPEL file and the implementation required to interact with actual partner endpoints (Web services). The descriptions include the binding information for each partner role or my role defined in a partner link;

- Creating deployable BPR (Business Process Archive) (*.bpr): This file is created using Java’s jar tool and contains various files created and placed into the appropriate archive directory locations;
- Deploying the BPR file (BPEL process) as a Web services: Deploying a Web service represents enabling a Web service to execute on a specific Web service. In general, this means copying the Web service entry points and all relevant information to the target Web server.

6. System Implementation

A prototype system is being realized in the project ImportNET to validate OCDEII. The main functions of the system include: creating reference ontologies based on DOLCE top ontology library [4] and generating collaboration ontologies by Protégé; mapping *.owl file to *.bpe and *.wsdl files automatically; building a workflow engine based on ActiveBPEL Community Edition Engine [10].

![Figure 6. CDUI for OCDEII](image)

As we have mentioned in the motivating scenario, Mike could view the changes which Eric has made after the system finished the request. Figure 6 shows the Cross-Domain User Interface (CDUI) of ImportNET in which OCDEII is being used in testing phase. The mapping details from *.owl file to *.bpe and *.wsdl files will be displayed in the left area of Figure 6 when we press the button “Mapping Monitor”. The reply result is displayed with graphic interface in the right area of Figure 6. The red circle figures out the hole which had been changed by Mike. Now Eric could know the change clearly and continue his work on the design of corresponding circuits.

7. Related Work

Many studies have been dedicated to cross-domain enterprises integration and interoperability under SOA. Tang presented a flexible service-oriented architecture for cross-enterprise business integration and cooperation [11]. Zhao, et al. first reviewed the Web Service-Resource Framework (WS-RF) with emphasis on the concepts of Web Services and static resource modeling and outlined a service-oriented architecture that showed how various types of manufacturing resources are virtualized as Web services [12]. But most of these works ignored the semantic information in integration.

Some others approached to develop semantic Web services by which the Web services are annotated based on shared ontologies. LSDIS Lab in University of Georgia discussed one approach that added semantics to WSDL using DAML+OIL ontologies [13]. Tsai et al. gave an ontology based dynamic process collaboration framework which supported the modeling and composition of complex SOA systems with automated code generation and code deployment in [14]. Martin et al. elaborated OWL-S particularly in [9, 15] and semantic was added to Web services by extending BPEL and WSDL with OWL-S. Aslam and Shen presented a method mapping BPEL to OWL-S and implemented a mapping tool [16, 17]. To the best of our knowledge, these work focused on adding semantic to BPEL and WSDL which was useful in bringing semantics to processes and Web services but not efficiently in reflecting the original extractive semantic from realistic world. Bordbar, et al. proposed a framework SiTra for the transformation from OWL-S to BPEL [18], their work concentrated on adding semantics to BPEL but it was insufficient in the mapping from OWL-S to BPEL.

In this paper, we have presented an approach to establish a system which can map ontologies from different domains to processes and Web services. Our objective is to propose an approach to bridge real word semantic to process semantic directly.

8. Conclusions and Future Work

In this paper we bring ontology to solve semantic inconsistent problem in the integration and interoperability among cross-domain enterprises. An architecture (OCDEII) is proposed to support this ontology based integration and interoperability. Firstly, reference ontologies corresponding to each domain were built; secondly collaboration ontologies are semi-automatically generated from reference ontologies; then mapping methods which mapping OWL-S files generated
from collaboration ontologies to BPEL and WSDL files were given respectively. Finally, we implemented a prototype system to fulfill some functions in our architecture and the system is being used in the project ImportNET.

We use WSDL 1.1 when mapping between OWL-S and WSDL. In WSDL 2.0 [19], there are specifications presenting a standard RDF and OWL vocabulary to express WSDL, so that WSDL 2.0 documents can be transformed to RDF and merged with other Semantic Web data. Consequently, the mapping from OWL-S to WSDL 2.0 will mostly avoid the semantic loss and make WSDL more readable among cross-domain enterprises, so we will work on the mapping between them in the future. In addition, we do not give a mapping for service profile ontology. Therefore our method is not a full mapping for OWL-S. Service profile contains three aspects of service: functional aspect, classification aspect and non-functional aspect, we plan to map these aspects to the interaction part (messages) in the next step.

Acknowledgements

We wish to thank all partners involved in the project ImportNET, especially Prof. Shuangxi Huang, Milan Marinov, Dr. Alexander Mahl, Wernher Behrendt, Manuela Plößnig, Diana Bischof, Suan Terge, Miklós Szőts, they have taken an active part in the work leading to the elaboration of this paper, even if they might not have directly contributed to writing parts of this paper. We also thank Stanford Center for Biomedical Informatics Research, this work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine. We gratefully acknowledge funding from European Commission within the Sixth Framework Programme (Grant No. FP6-033610), National High-Tech R&D Plan of China (Grant No. 2006AA04Z166) and National Natural Science Foundation of China (Grant No. 60674080 and 60504030).

References