

# From BPEL4WS to OWL-S: Integrating E-Business Process Descriptions

Jun Shen<sup>1</sup>, Yun Yang<sup>1</sup>, Chuan Zhu<sup>2, 1</sup>, Chengang Wan<sup>1</sup>

<sup>1</sup> Centre for Internet Computing and E-Commerce  
Faculty of ICT  
Swinburne University of Technology  
PO Box 218, Melbourne, Australia, 3122  
solo.shen@gmail.com, yyang@it.swin.edu.au

<sup>2</sup> Software Development Laboratory  
IBM China  
DeShi Building No.9, East Road, ShangDi  
Haidian District, Beijing, P.R. China, 100085  
zhuchuan@cn.ibm.com

## Abstract

*With the rapid deployment of e-services, many workflow-like e-business process definition languages come into existence. At the same time, Ontology Web Language for Services (OWL-S) aims to build an ontology language to support the integration of various specifications. There has been some work on mapping WSDL (Web Services Description Language) to OWL-S to build a connection between the Web service and service profile. However, in the sense of activity relationships, there has been no effort so far trying to build the OWL-S service model from a workflow process model. Therefore, we design and develop an innovative mapping tool to translate BPEL4WS (Business Process Execution Language for Web Services) to OWL-S. Through this mapping, semantics in the traditional business process specifications can be enriched significantly to enable more flexible and automatic e-service functions by using existing OWL-S tools such as composition and discovery, especially the execution of workflow-based services.*

## 1. Introduction

In the highly competitive business environments, effective information exchange and efficient communication become very critical today. Web services are meeting this demand. Web services are applications that can be published, located, and invoked across the Web. They perform functions like a business process. After deployed, other applications can find and invoke these services. Web services improve the Internet use by enabling program-to-program communication. With the development of the Web services, more and more computers can interact directly and integrate as they are part of one whole system. A Web service is like a unit of work which can complete a specific task. Many tasks can be combined to a

business task in the real world. This allows non-technical people to handle business processes without thinking about application processes. Therefore, once Web services are designed and built by technical people, business process architects can aggregate them to solve the business level problems. Web Services Description Language (WSDL) provides a model and an XML format for describing Web services and locating a Web service or the operation the Web service exposes [2]. WSDL enables dynamic discovery and binding of compatible services which are used in conjunction with registry services. Within the Web services architecture there are three roles: service provider, service requester and service broker; and three basic operations: publish, find and bind. A network component can play any or all of these roles.

Workflow is the solution to get work done from the start. It is composed of process logic and the routing rules. The process logic defines the sequence of tasks and the routing rules that must be followed, as well as deadlines and other business rules implemented by the workflow engine. Workflow-based applications are software that access process definitions and run jobs based on those process definitions via their workflow engine components. They are not only Intranet processes, but also Internet processes. Many enterprises connect their tasks into the Internet through these applications.

There are many flow-like specifications for Web services choreography nowadays, among which BPEL4WS (Business Process Execution Language for Web Services) seems to be the most promising one. However, the semantics of BPEL4WS is very limited especially when it is compared with Ontology Web Language for Services (OWL-S). OWL-S can also act as an inter lingua between the business process definition languages. Therefore, research groups at Stanford University and Carnegie Mellon University have led the work in adapting BPEL4WS or WSDL for semantic Web or OWL-S [3, 5]. Till now, such a work

only maps WSDL service descriptions to the OWL-S service profile, which only deals with inputs, outputs, preconditions and effects (IOPE) of related processes. They promise that the automatic service composition and discovery can be realised by matchmaker inference mechanisms. However, to enable executable service invocation and enactment, it is an important requirement to extend the BPEL4WS workflow model to the OWL-S service model. In this paper, we describe a tool translating BPEL4WS definitions into OWL-S process model specifications. Our innovative work would be a significantly complementary work to the above mentioned efforts.

This paper is organised as follows. Section 2 will revisit some basic concepts of workflow and BPEL4WS, as well as ontology and OWL-S. The detailed translation mechanism will be presented in Section 3, followed by an overview of our translation tool and future work in Section 4 and a conclusion in Section 5, respectively.

## 2. Background

### 2.1. Workflow and BPEL4WS

First of all, more and more companies wish to adopt the workflow technology because workflow-based applications are composed of a two-level programming structure, which makes deployment easier. With this structure, the specification of the logic and the application functions are separated. Therefore, the changing of the model of the process will not affect the associated activity implementations. Secondly, workflow-based applications can be reused. Because activity implementations for process models are typically flow-independent, a particular activity implementation can be used in many different process models. The third benefit is the scalability of workflow-based applications. It allows workflow-based applications to be used not only for small applications, but also for large ones. The workflow-based applications can be developed at two separate levels. One is the development and test of a business process and another is the development and test of the activity implementations, which reduce the complexity of the application from the design, implementation, and testing point of views. This parallel development of the activity implementations, which are made possible by the fixed interfaces to the business process, provides a faster development cycle [10].

Due to the lack of an agreed standard that can describe the public process and composition, we have to face a variety of different workflow standards such as WSFL [8], XLANG [9], BPEL4WS [1] and so forth. BPEL4WS represents a convergence of the ideas in the

XLANG and WSFL specifications. In other words, both XLANG and WSFL are superseded by the BPEL4WS specification. BPEL4WS provides a language to describe the behaviour of business processes and business interaction with their partners. There are two ways to apply BPEL4WS. One is to use the abstract process to describe the business protocol role to keep the private behaviour invisible. The other way is to define the executable business process. It combines the best of WSFL (support for graph oriented processes) and XLANG (structural constructs for processes) into a more handy specification which is able to implement business processes freely.

WSDL has the most influence on BPEL4WS because a BPEL4WS process is built on the basis of the service model which is defined by WSDL. The two key concepts, process and partner, are modelled as WSDL services. A BPEL4WS process is a reusable definition that can be deployed in different ways and in different scenarios, while maintaining the uniform application-level behaviour across all of them. The core of the BPEL4WS process model is the notion of peer-to-peer interaction between services described in WSDL. In the WSDL model the abstract message content of business processes and deployment information are separated as messages and portTypes accordingly. Within its portTypes the interactions among them are defined as operations. In the same way BPEL4WS also follows this model with messages, partners, and communication activities among partners.

With these basic units from WSDL at hand BPEL4WS is going to composing a set of services into a new service, or a process. At the lowest interaction level BPEL4WS uses its primitive activities like <invoke>, <receive>, <reply>, <throw> to model the behaviours among partners and then combine these primitive activities into more complex process units at a higher level by using structural activities like <flow>, <sequence>, <switch>, <while>, <pick> and so on. BPEL4WS allows recursively combining the structured activities to express arbitrarily complex algorithm logic.

The services with which a business process interacts are modelled as partner links in BPEL4WS. Each partner link is characterised by a *partnerLinkType*. During the interaction phase in a business process, there are some data to be maintained for later use. Sometimes we also need to manipulate data for some specific requests. A process definition is made of an activity, a series of partners and containers with specific correlation sets, the definition of fault handlers and compensation handlers [8]. Business processes specified via BPEL4WS prescribe the exchange of messages between Web services. These messages are WSDL messages of operations of the port types involved in the roles of the service links established

between the process and its partners. Some of the messages exchanged will be included in a collection of WSDL messages, that is, containers.

The control flow is a nice hybrid model half-way between block structured and state transition control flow definitions. The model uses “links” to establish the dependencies between block structured definitions. The control flow is very much defined by all the possible activities as defined by BPEL4WS semantics. Activities are the actions that are being carried out within a business process.

Although BPEL4WS has many advantages, it also has some limitations. Because the expressiveness of WSDL service behaviour is restricted to the interaction specification and BPEL4WS uses WSDL portType as service information, BPEL4WS inherits the limitations of WSDL. Furthermore, BPEL4WS can not express the inheritance and relationships among the Web services. It can not provide well-defined semantics for automated composition and execution. Moreover, these languages are based on XML in essence, they are limited in semantic descriptions without enough ontology support.

## 2.2. Ontology and OWL-S

With ontology we can deal with the multiple workflow specifications at a higher level. Therefore each workflow can talk to others without knowing about a totally independent language and it only needs to map itself to the ontology specification. Ontology defines a common vocabulary for stakeholders who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them [6]. Therefore, programmers only need to map the workflow language to the ontology, then they can easily map back to other workflow languages when needed. That was also the main aim of one of our previous projects [7].

W3C’s Web-Ontology (WebOnt) Working Group developed the OWL language. OWL’s ability to express ontological information about instances appearing in multiple documents supports linking of data from diverse sources in an in-principle way. The underlying semantics provides the support for inferences over meta-data that may yield expected results. In particular, the ability to express equivalences using *owl:sameAs* can be used to state that seemingly different individuals are actually the same. *Owl:InverseFunctionalProperty* can also be used to link individuals together. For example, if a property such as “Soc Sec Number” is of the *InverseFunctionalProperty*, then two separate individuals could be inferred to be identical based on having the same value of that property. When

individuals are determined to be the same by such a means, information about them from different sources can be merged. This aggregation can be used to determine the facts that are not directly represented in any single source.

OWL goes beyond languages like XML and it is able to represent machine interpretable contents on the Web. OWL is not only a message format but also a knowledge representation and furthermore there are tools available to reason about it. OWL has three sublanguages which can meet the requirements of different implementers and users to various extents [11]. Class *owl:Thing* is the most basic one in the OWL world as root. Every class in the real situation is a subclass of *owl:Thing*. There is also an exception, which is, if the class contains nothing or is just empty, it is defined as *owl:Nothing*. An individual is a member of a class, so it should contain some actual entities that can be grouped into classes while classes correspond to the naturally occurring things in a domain of discourse. To enrich the meanings of classes, OWL uses properties to give classes and individuals more general and specific facts. Here are two types of properties: *datatype properties*, i.e. relations between instances of classes and literals and XML schema data types; *object properties*, i.e. relations between instances of two classes [11].

OWL-S is designed to carry information about the resources that describe or provide Web services contents in order to make these resources more accessible to automated processes. This means that OWL-S must allow the information from distributed sources to be gathered and related in it [4]. A typical OWL-S ontology begins with a namespace declaration. This set of XML namespace declarations provides a way to interpret identifiers and makes the rest of the ontology presentation much more readable. The second part after the namespaces is a collection of assertions about the ontology which includes comments, version control and inclusion of other ontologies. The last part is the core of OWL-S that merges data from multiple sources and combines with the inferential power of itself. The elements in OWL-S look much like in the object oriented programming style which uses classes, objects (individuals) and variables (properties) to specify things.

## 3. Mapping

### 3.1. Basic analysis

As mentioned earlier, BPEL4WS and OWL-S have totally different ways to describe business processes. From the programming viewpoint, BPEL4WS applies a procedural method while OWL-S is more like an object



one is abstract process and the other is executable process. While in the OWL-S process service, there are three kinds of processes, simple process, atomic process and composite process. BPEL4WS uses the notion of abstract processes to describe the interfaces of business processes. They provide a means of synchronisation with other processes at varying levels of granularity, for purposes of planning and reasoning. In OWL-S the simple processes act as a “view” on either atomic or composite processes. They provide a level of abstraction although sometimes they are optional. So we can reasonably map the abstract process to the simple process in OWL-S. As the core of BPEL4WS, executable processes play an important role in this implementation language. We use both atomic and composite processes in OWL-S to translate them.

Simple processes describe themselves in the same way as atomic processes but the description is only a view. By using properties “*collapsesTo*” and “*expandsTo*” we can explore and realise them according to their true contents: atomic processes or composite processes. Here is an example of a simple process as a view of a composite process.

```
<process:SimpleProcess rdf:ID="AbstractCongoBuy">
  <process:expandsTo rdf:resource="#FullCongoBuy"/>
</process:SimpleProcess>
<process:CompositeProcess rdf:about="#FullCongoBuy">
  <process:collapsesTo
    rdf:resource="#AbstractCongoBuy"/>
</process:CompositeProcess>
```

Like the important influence of WSDL upon BPEL4WS, it is also an essential part for OWL-S. WSDL defines almost all the data types of the service which are used directly in the process. Message is what WSDL uses to define and carry data types which are the basic interaction unit like a word we talk to others [5]. Here is an example. A message named Order is defined in the *Definitions.wsdl* (namespaces ns9 and ns10). In *DemoProcess.wsdl*, this message is incorporated in an operation as an attribute.

```
<portType name="BuyerPT">
  <operation name="create">
    <input message="def:Order"/>
    <output message="def:Status"/>
  </operation>
  <operation name="agree">
    <input message="def:ID"/>
    <output message="def:Status"/>
  </operation>
</portType>
```

In a BPEL4WS process a variable is defined to carry this data type for future use and the variable acts as a shortcut in the process to access this data type.

```
<variables>
  <variable name="order"
    xmlns:ns9="http://bpeldemo.ibm.com/definitions/"
```

```
  messageType="ns9:Order"/>
<variable name="status"
  xmlns:ns10="http://bpeldemo.ibm.com/definitions/"
  messageType="ns10:Status"/>
</variables>
```

By now, mapping from BPEL4WS to OWL-S is only to concern about the process transformation. Therefore we can ignore how BPEL4WS imports and uses the data types and extracts them from WSDL directly for OWL-S to use. Along with the process mapping result, the data flow OWL-S file serves as an assistance to specify the messages exchanged within the whole business process. In each process, atomic or composite, where there is data exchanging, the data flow uses the *sameValues* property to specify where the data in the process comes from.

We can see how the data flow from one activity to another from the description of the composite process. In the scope of *sameValues*, the two *valueOf* properties specify the data positions respectively, the composite process and its atomic process where the data are defined. Also from the viewpoint of data flow, the relationship between the atomic process classes beneath the process can be implicitly indicated. After the description of the composite process, the related atomic processes are described one by one. From these descriptions we can see that these atomic processes are actually the sub class of the atomic processes which are derived from the *operations* in WDSL, the data source.

### 3.3. Activities

This sub-section addresses the key part of the whole mapping specification because the activities are elementary blocks of the business process. Based on the BPEL4WS activities types, we divide this mapping part into two parts: primitive and structural activities.

All the primitive activities in BPEL4WS are mapped to the atomic process in OWL-S but there are some slight differences among them. Although they occur in BPEL4WS it is WSDL which actually defines them initially in its *operation* part resided in *portType*.

The <reply> activity allows the business process to send a message only. So it only has the output. We can find this basic activity appearing quite often in BPEL4WS which is equivalent to <output> in a WSDL *operation*. We can get this <reply> information either in BPEL4WS or in WSDL *portType*. However the former provides more information about this activity in the whole business process while the latter only keeps the basic message content. In OWL-S, we map it to the atomic process with property *hasOutput*. During the whole mapping phase, we actually extract the atomic processes from the *operations* in WSDL first while the atomic process for the <reply> activity is a sub class of the atomic process derived from the *operation* which

contains the *output* where `<reply>` originally comes from. All these relationships are in black boxes, but in the data flow, they can be explicitly shown up. The mapping of `<receive>` is quite similar to `<reply>`. The only difference, due to their characteristics, is that `<receive>` merely has inputs. Due to `<invoke>`'s own characteristics, its mapping is the combination of `<reply>` and `<receive>`. But not every `<invoke>` has both *inputVariable* and *outputVariable* attributes. The following example of `<invoke>` has both inputs and outputs (refer to `Invoke_1` for validation in Figure 2). Target `<invoke>` in BPEL4WS is:

```
<invoke name="Validation_Invoke"
  partner="ValidationService"
  xmlns:svc="http://bpeldemo.ibm.com/services/"
  portType="svc:ValidationUtilityPT" operation="validate"
  inputVariable="order" outputVariable="status"/>
```

The source in WSDL is:

```
<portType name="ValidationUtilityPT">
  <operation name="validate">
    <input message="def:Order"/>
    <output message="def:Status"/>
  </operation>
</portType>
```

The mapping result in OWL-S looks like

```
<process:AtomicProcess rdf:ID="Invoke_1">
  <process:hasInput>
    <process:Input rdf:ID="Invoke_1_Order">
      <process:parameterType rdf:resource="#Order"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:UnConditionalOutput
      rdf:ID="Invoke_1_Status">
      <process:parameterType rdf:resource="#Status"/>
    </process:UnConditionalOutput>
  </process:hasOutput>
</process:AtomicProcess>
```

The `<throw>` activity is used to generate an exception message in the business process. Because OWL-S has not specified how to handle the fault currently, we just temporarily treat `<throw>` exactly the same as `<reply>` with attribute *faultVariable* corresponding to the *outputVariable* within `<reply>`. Like its name, `<empty>` activity actually does nothing in the business process. However there is a reason why it exists in OWL-S. The reason is that `<empty>` can be used in the situation of synchronisation in BPEL4WS when many concurrent processes need to be managed. Therefore it is mapped to an "empty" atomic process in OWL-S that has nothing in it.

As the structural activity is composed of more than one activity that can be primitive activities or structural activities or both, we map the structural activities to the composite processes in OWL-S accordingly. Composite processes play the same role in OWL-S just like the function of structural activities in BPEL4WS. A composite process contains more than one sub process

(atomic or composite process). In order to describe composite processes precisely OWL-S has a minimal set of control constructs. The basic primitives are *Sequence*, *Split*, *Split + Join*, *Unordered*, *Condition*, *If-Then-Else*, *Repeat-While*, *Repeat-Until*. OWL-S has not specified the function and use of the *condition* class so far. We just put the content of *condition* the same with corresponding *condition* in BPEL4WS without any changes.

The `<sequence>` activity contains one or more activities that are executed sequentially. The activities are executed in the order that they appear within the `<sequence>` element. When the final activity in the `<sequence>` has been completed, the `<sequence>` activity itself is completed [1]. We map the `<sequence>` activity to the composite process with control construct *sequence*. So far the preconditions and effects of *sequence* are not defined in the current version of BPEL4WS but they are not very important to the mapping of `<sequence>`.

It is an OWL-S convention that within the definition of a composite process, all the outputs and inputs that belong to the atomic processes residing in this composite process should be claimed one by one. Because there is no directly corresponding control construct in OWL-S to model the logic of `<switch>` activity, we use multiple *If-Then-Else* to fulfil this logic, which comes from the basic programming knowledge. A `<switch>` in BPEL4WS (refer to `Switch_4` for solvency check in Figure 2):

```
<switch name="Solvency_Switch">
  <case condition="bpws:getVariableData('status', 'status',
    '//type')=3">
    <sequence name="Solvency_Sequence">
      .....
    </sequence>
  </case>
</switch>
```

The correspondent part in OWL-S looks like:

```
<process:CompositeProcess rdf:ID="Switch_4">
  <process:composedOf>
    <process:If-Then-Else>
      <process:ifCondition rdf:resource=
        "bpws:getVariableData('status', 'status',
          '//type')=3"/>
      <process:then>
        <process:CompositeProcess rdf:about="#Sequence_6"/>
      </process:then>
      <process:else>
      </process:else>
    </process:If-Then-Else>
  </process:composedOf>
</process:CompositeProcess>
```

In OWL-S, we use control construct *Repeat-While* to model the `<while>` activity with *whileCondition* holding attribute *condition* in the `<while>` activity. The

<pick> activity will run one of its event handlers within itself when this handler is activated. We map it to the composite process with the *Choice* control construct just from the original and explicit format. It means that this mapping can explain all the actions that the <pick> activity has but cannot specify the situation which has been clearly handled by BPEL4WS by using *onMessage*, *onAlarm* and other event handlers.

The substitution of the <flow> activity in OWL-S is the composite process with the *Split* control construct. However *Split* only takes care of concurrency but still waits for the further specification about waiting and synchronisation. Suppose that the <flow> activity appear in BPEL4WS as (refer to Flow\_1 of main flow in Figure 2):

```
<flow name="Main_Flow">
  <sequence name="CREATION_SEQUENCE">
    .....
  </sequence>
  <while name="SEARCH_CYCLE" condition="0=0">
    .....
  </while>
  <sequence name="TERMINATION_SEQUENCE">
    .....
  </sequence>
</flow>
```

The mapping in OWL-S looks like:

```
<process:CompositeProcess rdf:ID="Flow_1">
  <process:composedOf>
    <process:Split>
      <process:components rdf:parseType="Collection">
        <process:CompositeProcess rdf:about="#Sequence_1"/>
        <process:CompositeProcess rdf:about="#While_2"/>
        <process:CompositeProcess rdf:about="#Sequence_10"/>
      </process:components>
    </process:Split>
  </process:composedOf>
</process:CompositeProcess>
```

### 3.4. Limitations and clarification

There are some limitations in our mapping specification. Some of them have been already encountered earlier. This sub-section will outline them and show the temporary means that we handle them.

By now *condition* is still a “place-holder” which awaits further work from the OWL-S community. However *condition* is the key part for some important logical control constructs like *If-Then-Else*, *Repeat-While* and *Repeat-Until*. This directly leads to the incomplete mapping of some activities like <switch> and <while>. Concurrency is another stated unfinished issue which is vital to manage a large scale business system. Because of some heavy traffic there are surely some concurrent processes and the ability to deal with

them which is an important criterion to measure the quality of the whole system.

Compared to BPEL4WS, which is a complete workflow language, OWL-S needs time and efforts to get mature. Therefore some functions and activities in BPEL4WS are far beyond the current OWL-S capability like fault handling, value assignment, correlation set and so on. Some of these issues have been addressed in the discussions in recent semantic Web services conferences and workshops. Nevertheless, BPEL4WS and OWL-S are yet to be widely accepted and deployed until further elaboration of business process semantics. Our work is a cornerstone towards that target.

## 4. The Prototype Tool and Evaluation

A mapping tool has been developed in Java. It supports multiple WSDL files which are accompanied with a BPEL4WS file. WSDL files are distinguished in terms of slave WSDL and master WSDL. The master WSDL is the main one that all slave WSDL(s) refer to. WSDL serves as the foundation of this mapping process. It describes all the data used in the business process. We extract the data from WSDL and define them in OWL-S, for example the messages and atomic processes defined beforehand. From the OWL-S process file, there may be some data never used at all but it is the most effective and efficient way to deal with the data in WSDL. By doing so, we need not go back to WSDL to search the source every time when treating an activity. If we consider the WSDL as the flesh, BPEL4WS will be the skeleton. BPEL4WS explains a business process specifically and accurately. Accordingly we build our process structure in OWL-S corresponding to BPEL4WS closely but in a totally different way. BPEL4WS, as a typical workflow language, uses flow to describe the business process while OWL-S, as an ontology language, converts the process to a hierarchical structure which is similar to the object-oriented programming concept.

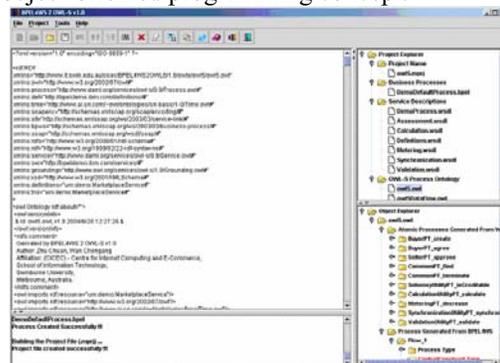


Figure 3. The interface of the mapping tool  
Figure 3 is the GUI of our tool. On the right hand

side there are two sub-windows, one is project explorer and the other is object explorer. From the project explorer all the files in this project can be accessed and the source can be seen in the main window on the left when double click on that file. The object explorer is used to explore the elements of any file that shows up in the main window. Because all the files are in the XML format which is hierarchical, object explorer extracts and illustrates objects in the file in a tree format. Currently this tool does not support multiple BPEL4WS files. Since OWL-S is a recent Web service ontology language, its specification is not completed yet. Therefore there are very few validation tools to check the syntax. Some OWL-S validators are too simple to really validate a complex OWL-S file.

This tool and its predecessor, which translates BPEL4WS to DAML-S [7], can be downloaded from our Website ([www.it.swin.edu.au/centres/ciccc/bpel2owls.htm](http://www.it.swin.edu.au/centres/ciccc/bpel2owls.htm)) for demonstration and test. The generated OWL-S files can be applied in many existing tools [4]. Our future target is to map other popular workflow process language such as XPDL [12].

## 5. Conclusion

In business process reengineering projects, one of the most critical issues is data integration and interoperability. To solve this problem we need some abstraction from a higher level to integrate various individual workflow or Web service specifications. That is the reason why OWL-S is so attractive even it is still in its infancy. Semantic Web is the new vision of the Internet because it is more meaningful for computers to implement automated interactions. People can make full use of the power of information networks through semantic Web.

Our work is meeting this objective. BPEL4WS is one of the popular workflow process languages towards Web services, so we map it to OWL-S ontology to incorporate richer semantics description capabilities. The mapping methodologies and tools developed in this paper can play significant roles in the integration efforts of business process descriptions.

As the future of OWL-S is promising, this project needs to undergo constant updates to keep up with OWL-S advancing steps. We believe once OWL-S becomes mature enough this whole project will contribute greatly to the integration of Web service languages, thus semantic Web based Web service architectures will be more powerful and intelligent, and better accepted by enterprises, especially nowadays business process and service outsourcing become more popular so that the heterogeneity hampers well-designed service dissemination.

## Acknowledgements

This work is partly supported by Swinburne Vice Chancellor's Strategic Research Initiative Grant 2002-2004, as well as the National Natural Science Foundation of China under grants No.60273026 and No.60273043. We gratefully acknowledge invaluable feedback on our prototype from related research communities. We also thank the anonymous reviewers for giving us instructive advices on our work to improve this paper.

## References

- [1] T. Andrews, F. Curbera, et.al. *Specification: Business Process Execution Language for Web Services Version 1.1*, <http://www-106.ibm.com/developerworks/Webservices/library/ws-bpel>, 2003
- [2] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. *Web Services Description Language (WSDL) 1.1*, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, 2001
- [3] D.J. Mandell and S. McIlraith. *Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation*. Proc. of 2nd International Semantic Web Conference (ISWC2003), Lecture Notes in Computer Science, Vol. 2870, pp.227-241, 2003
- [4] OWL-S 1.1. *Upper Ontology for Services*, <http://www.daml.org/services/owl-s/1.0/>, 2004
- [5] M. Paolucci, N. Srinivasan, K. Sycara and T. Nishimura. *Toward a Semantic Choreography of Web Services: From WSDL to DAML-S*. Proc. of 1st International Conference on Web services (ICWS'03), pp.22-26, 2003
- [6] J. Shen and Y. Yang. *Extending RDF in Distributed Knowledge-Intensive Applications*. Future Generation Computer Systems, 20(1): 27-46, 2004
- [7] J. Shen, Y. Yang and B. Lalwani. *Mapping Web Services Specifications to Process Ontology: Opportunities and Limitations*. Proc. of 10th IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS'04), pp. 229-235, 2004
- [8] J. Snell. *The Web Services Insider, Part 4: Introducing the Web Services Flow Language*, <http://www-106.ibm.com/developerworks/library/ws-ref4/index.html>, 2001
- [9] S. Thatte. *XLANG: Web Services for Business Process Design*, [http://www.gotdotnet.com/team/xml\\_wsspecs/xlang-c/default.htm](http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm), 2001
- [10] M. Virdell. *Business Processes and Workflow in Web Services World*, IBM DeveloperWorks, <http://www-106.ibm.com/developerworks/Webservices/library/ws-work.html>, 2004
- [11] W3C. *OWL Web Ontology Language Guide*, <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>, 2004
- [12] WfMC. *Workflow Process Definition Interface-XML Process Definition Language, V 1.0*. Lighthouse Point, FL, [http://www.wfmc.org/standards/docs/TC-1025\\_10\\_xpdl\\_102502.pdf](http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf), 2002