December 2006

Analysis of Business Process Integration in Web Service Context

J. Shen
University of Wollongong, jshen@uow.edu.au

G. Grossmann
University of South Australia

Y. Yang
Swinburne University of Technology

M. Stumptner
University of South Australia

M. Schrefl
University of South Australia

See next page for additional authors

Analysis of Business Process Integration in Web Service Context

Abstract

The integration of Web services is a recent outgrowth of the Business Process integration field that will require powerful meta-schema matching mechanisms supported by higher level abstractions, such as UML meta-models. Currently, there are many XML-based workflow process specification languages (e.g. XPDL, BPEL) which can be used to define business processes in the Web services and Grid Computing world. However, with limited capability to describe the relationships (schemas or ontologies) between process objects, the dominant use of XML as a meta-data markup language makes the semantics of the processes ambiguous. OWL-S (Ontology Web Language for Services) exploits the semantic description power of OWL to build an ontology language for services. It therefore becomes a candidate for an inter lingua. In this paper, we propose an integration framework for business processes, which is applied to Web services defined in OWL-S.

Keywords

Business process management, Web services, Semantic web, Specification integration, UML meta-models

Disciplines

Physical Sciences and Mathematics

Publication Details

This article will be published as: Shen, J, Grossman, G, Yang, Y, et al, Analysis of Business Process Integration in Web Service Context, FGCS (Future Generation Computer Systems): The International Journal of Grid Computing: Theory, Models and Applications, 23(3), 283-294 (in press, to be published 7 March 2007). Original journal from Elsevier available here.

Authors

J. Shen, G. Grossmann, Y. Yang, M. Stumptner, M. Schrefl, and T. Reiter

Analysis of Business Process Integration in Web Service Context ⋆

Jun Shen [a,b,c,*], Georg Grossmann [b], Yun Yang [c], Markus Stumptner [b], Michael Schrefl [b], Thomas Reiter [b]

[a] *School of Information Technology and Computer Sciences*
University of Wollongong
Wollongong, NSW 2522, Australia.

[b] *School of Computer and Information Sciences*
University of South Australia
Mawson Lakes, SA 5095, Adelaide, Australia.

[c] *Faculty of Information and Communication Technologies*
Swinburne University of Technology
Hawthorn, VIC 3122, Melbourne, Australia.

Abstract

The integration of Web services is a recent outgrowth of the Business Process integration field that will require powerful meta-schema matching mechanisms supported by higher level abstractions, such as UML meta-models. Currently, there are many XML based workflow process specification languages (e.g. XPDL, BPEL) which can be used to define business processes in the Web services and Grid Computing world. However, with limited capability to describe the relationships (schemas or ontologies) between process objects, the dominant use of XML as a meta-data markup language makes the semantics of the processes ambiguous. OWL-S (Ontology Web Language for Services) exploits the semantic description power of OWL to build an ontology language for services. It therefore becomes a candidate for an inter lingua. In this paper, we propose an integration framework for business processes which is applied to Web services defined in OWL-S. We first describe the mapping from BPEL to OWL-S and what semantic relationships between Web services look like in OWL-S. Based on these relationships, integration options are applied which create a new composite Web service semi-automatically.

Key words: Business Process Management, Web services, Semantic Web, Specification Integration, UML Meta-Models

1 Introduction

The Web services paradigm is poised to become the dominant form of distributed computing within this decade and beyond. An EDS global consultancy, found that 75% of companies ranging from less than $50 million to more than $1 billion in revenues and across 20 vertical industries have already deployed one or more Web services [1]. Web services involve a family of XML-based protocols to describe, deliver, and interact with services. WSDL is the most important one in our context. WSDL files include a set of standard elements. These elements describe interfaces and usage of a particular Web service [2]. Workflow management systems have become a promising solutions for the organisations that need to automate their business processes [3]. Applying workflow to a business process brings the details of that process into focus and adds the required business rules and business logic to the process.

Typical XML based workflow process definition and execution languages include BPEL4WS (Business Process Execution Language for Web services, BPEL in short) [4], XPDL (XML Process Description Language) [5], ebXML, etc. that can be used to describe workflow systems and business processes in the Web services world. Integration of these languages requires comprehensive and complex mappings between them [6]. Intuitively, UML meta models may meet these requirements to some extent, especially in providing visual forms for models of classes and respective associations. In this paper, we describe a set of business process integration options and a set of additional modeling constructs, especially for the synchronisation of activities and states within a process.not easily described with full semantics. We have developed a transformation tool, BPEL2UML-AD [7], to transform BPEL specifications to UML activity diagrams [8] (referred to as UML-AD for short in the rest of the paper). The advantage of UML-AD is that they provide an effective visual notation and facilitate the analysis of workflow compositions.

⋆ This work was partially supported by Australian Research Council under Discovery Grant DP0210654 and DP0663841, and Swinburne Vice Chancellor's Strategic Research Initiative Grant 2002-2004. We gratefully acknowledge the invaluable feedback on our open source prototype from related research communities. Several higher degree students, B. Lalwani, C. Wan and C. Zhu, contributed their time and efforts to the prototype implementations.

* Corresponding author. School of Information Technology and Computer Sciences, University of Wollongong, Wollongong, NSW 2522, Australia. Tel. +61-2-42213873; fax: +61-2-42214170.

Email addresses: jshen@uow.edu.au (Jun Shen), cisgg@cs.unisa.edu.au (Georg Grossmann), yyang@ict.swin.edu.au (Yun Yang), mst@cs.unisa.edu.au (Markus Stumptner), schrefl@dke.uni-linz.ac.at (Michael Schrefl), reiter@ifs.uni-linz.ac.at (Thomas Reiter).

As part of our analysis of workflow composition, we have identified a set of integration options which can be applied to Web services by mapping backwards from UML-AD to BPEL. This worked started out from an attempt to support the decision between different integration options. A possible basis are the different semantic relationships between process objects. However the description of these relationships cannot be carried over directly to BPEL because BPEL documents solely represents descriptions of activity execution without describing the semantics of involved objects. Therefore we propose to use a mapping from BPEL to OWL-S (Ontology Web Language for services, formerly DAML-S). OWL-S, jointly developed by a consortium including industry and research institutions, is an attempt to provide an ontology for describing Web services [9]. In this paper, we introduce an approach (BPEL2OWL-S) which support the mapping of business processes defined in BPEL onto an OWL-S based process ontology [12].

The paper is organised as follows. Section 2 discusses some typical workflow and Web service specification languages. In Section 3 we analyse integration options on the basis of workflows. In the following section we deal with the mapping of BPEL to OWL-S. Section 5 discusses our findings and related work. The last section concludes our work.

2 The Current State of Workflow and Web Service Description Languages

The current state of the art in workflow description languages in a Web service environment was based on two separate standards, WSFL (Web Service Flow Language) and XLANG. WSFL (xml.coverpages.org/wsfl.html), from IBM, addresses workflow on two levels: (1) it takes a directed-graph model approach to defining and executing business processes; and (2) it defines a public interface that allows business processes to advertise as Web services. XLANG (xml.coverpages.org/xlang.html), from Microsoft, plays the role of notation for Web services based business process automation. As the basis of automated protocol engines, it supports the exchange of messages among various Web services, tracks the state of process instances, and detects errors in message flows to some extent.

BPEL4WS (Business Process Language for Workflow Systems, or BPEL for short) was developed as an attempt to unify XLANG and WSFL and supersedes both these efforts. It allows businesses to describe sophisticated business processes that can both consume and provide Web services. The language is intended to support the modeling of both executable and abstract processes. An abstract process is a business protocol that specifies the message exchange behaviour between different parties without revealing their internal behaviour.

An executable process specifies the execution order between a number of activities that constitute the process, the partners involved in the process, the messages exchanged between these partners, and the fault and exception handling that specify the behaviour to adopt in the cases of errors and exceptions [4]. A BPEL process is a flow-chart, where each element in the process is called an activity. An activity can be either primitive or structured. The set of primitive activities contains: <invoke>, <receive>, <reply>, <wait>, <assign>, <throw> and <empty>. Several structured activities are defined to enable the presentation of complex structures. These are <sequence>, <switch>, <pick>, <flow>, <compensate>, <scope> and <while>. A BPEL process definition provides and/or uses one or more WSDL services, and provides the description of the behaviour and interactions of a process instance relative to its partners and resources through Web service interfaces.

BPEL supports the implementation of any kind of business process in a very natural manner and has gradually become the basis of a standard for Web service description and composition. However, it has several shortcomings that limit the ability to provide a foundation for seamless interoperability. The semantics of BPEL are not always clearly defined, thus complicating the adoption of the language. Major limitations of the BPEL specification have been listed in [10,13]. At the heart of the problem is BPEL's reliance on describing services using pure XML and XML Schema.

Outside the pure Web services domain, the Workflow Management Coalition (WfMC) has been an active driving force in defining standard references to facilitate a process definition language, the interchange of process definitions and the interpretation of process definitions by different workflow management engines, and interoperability across different workflow management systems. The work conducted by WfMC allows developing composite workflow applications across different workflow management systems and organisations which work together as a single logical entity. For this endeavor, WfMC has published XPDL and interoperability specification Wf-XML [5]. XPDL belongs to the family of graph-structured process definition languages. There are also some other specific XML-based languages like e-Business XML (ebXML) and XML Routing Languages, which we have discussed elsewhere [14].

Van der Aalst and Hofstede [15] have analysed workflow patterns to compare the expressiveness of existing business process languages and have examined the properties of BPEL in [16].

3 Analysis of Workflow Composition

In order to integrate different Web service based business process specification languages, we need to analyse constructs of process models at higher abstraction level. We used diagram notations, especially Activity Diagrams (UML-AD) of the Unified Modeling Language (UML) [8], because traditional techniques of structured analysis and design are being increasingly replaced by object-oriented modeling approaches in the development of business information systems. Another reason was that in conjunction with UML, the Model Driven Architecture (MDA) was proposed by the OMG [17]. The main idea of MDA is to separate the platform specific modeling (PSM) and the platform independent modeling (PIM). Transformation tools help to refine a PIM into a PSM. A business process fits well into this concept because it describes the behaviour of a system on a high level and is independent from the implementation like the PIM.

We mapped BPEL to UML-AD [7] and analysed correspondences between business process elements of two separate UML-AD models in case of composition and association where we deal with part-of relationships and domain dependency relationships [18]. A main business process consists of subprocesses which may again have subprocesses of their own. The integration task is to build an integrated hierarchy of such processes and coordinate the control and data flow between them. Some studies addressed the composition of internal and external services and verification of consistency criteria [19]. Others pointed out the actual coordination of processes is still a missing aspect in current integration research and suggested solutions like event-based coordination [20]. In [21] we used a similar integration approach for creating a generalisation model out of two input business processes. In this paper we apply the approach to composite business processes.

For analysing composite business processes we used the life cycle of a composition which consists of the three phases, (a) **construction**, (b) **coordination**, and (c) **destruction**. In the construction phase a composite business process is created out of the separated component business processes, e.g., the business processes "building a car" consists of the business process "building tyres" and "building car body". A "component business process" is a business process which belongs to a component. In the following phase the composite business process must be coordinated with its components depending on the activities and states of the participating objects, e.g., a car accident happened and the composite business process "checking the car" coordinates the component business processes "checking car body" and "checking tyres". The destruction of the composition, e.g., "dismantling the car", terminates the life cycle.

Our integration approach is based on semantic relationships between business process elements. In the following section we apply this approach to the elements of Web services and link integration options to these elements which are described in Section 3.2.

3.1 Semantic relationships of composite Web services

Diagram notations used for conceptual behaviour modeling are based on two complementary notational primitives, states and activities. UML behaviour representation notations either emphasise states (in the case of state machine diagrams) or activities (in the case of activity diagrams) and attempt to minimise the use of the respective opposite primitives. While these representations are useful for particular facets of the software development process they typically cannot be used in pure form without overly restricting the modeler, resulting in the introduction of different kinds of "pseudostates" (really, particular types of transitions) in Statechart modeling and different types of "locations" (really, intermediate states) in Activity Diagrams. Instead, we use a symmetric notation that permits both states and activities to occur explicitly in the same diagram. Activities represent a situation of a business process where actions are executed which cannot be interrupted. We assume that an activity needs time and the duration of execution is unknown. In contrast, states represent a situation where the business process remains in a waiting position to receive an event which invokes an activity.

In [18] we identified semantic relationships between activities and states of business processes during the lifetime of composition. Here, we are going to apply this approach to Web services and demonstrate it on the example of a composite Web service T called "Travel service" which plans and coordinates travels to a specific destination D by invoking the following other Web services:

- "Flight booking service" ($F1$) books flights to D.
- "Hotel booking" ($H1$) books a hotel in D.
- "Limousine booking service" ($L1$) books a limousine for picking up a person from airport in D.
- "Restaurant booking service" ($R1$) books dinner in a restaurant in D.

For each Web service, there exists a compensatory Web service which cancels the specific booking, i.e., $F2$ cancels a flight, $H2$ cancels a room reservation, etc. $F1$, $H1$, $D1$, $R1$, and their respective compensation services are either atomic processes or composite processes.

Construction: The task of T in the construction phase is (1) to identify inter dependencies of services and (2) which is the optimal configuration of them. $F1$ and $H1$ depend on each other, e.g., if there is no flight available on a day

it is not necessary to book a hotel for that day. We define this dependency as *component commit cn_commit*$(F1, H1)$. It holds if $F1$ and $H1$ need to be executed successfully or none of them. $L1$ depends on date and time of the flight booked by $F1$. However $L1$ does not play an important role in the schedule because the person can be picked up by a taxi as an alternative. A *cn_commit*$(F1, L1)$ relationship is not necessary. Instead we define that $F1$ must be invoked before $L1$ by setting the *component history* relationship *cn_history*$(F1, L1)$. Finally, we can construct a relationship where one activity is actually defined as a subactivity of the other, a relation we refer to as *activity decomposition*: *subact*$(T, F1)$ holds if Web service $F1$ is executed as part of the execution of T, e.g., a hotel reservation as part of an overall trip reservation.

Coordination: During the coordination phase a composite Web service might need to coordinate other Web services because of incoming messages, e.g., the whole trip will be cancelled after T has booked the flight, the hotel, and the limousine. In this case all involved Web services $F2$, $H2$, and $L2$ need to be invoked. These relationships are often based on *subact* relationships as described above.

In [18] *interprocess dependencies* are discussed which deal with state conditions of business processes. In BPEL states are determined by the value of variables which are replaced by message types in OWL-S. *Interprocess dependencies* are split into three categories: (1) *Composite state dependencies* hold if a component Web service (e.g., the atomic process "receive credit details" A in $F1$) can only be executed if the respective composite Web service, is in the respective composite state, e.g., T in "credit card details are available". We denote such a dependency by *cs_ipd*(A, T, S) where S is a specific state. (2) *Component state dependencies* are similar to the previous example but here a component Web service must be in a specific state before a composite Web service can be executed, e.g., *cn_ipd*$(C, F1, S)$ holds if the atomic process "invoke flight cancel service" C in T can only only be executed if $F1$ is in the state "Flight booked successfully" S. (3) *State component state* condition defines the influence of a component Web service state on the state of it's respective composite Web service, e.g., the flight to D was cancelled because of technical problems of the airplane and $F2$ was executed successfully. Hence the state of T has to be changed because the travel schedule is not valid anymore. We define this relationship as *st_cn_ipd*$(C, F2, S)$ where C is a process in T, S represents a state of $F2$.

Destruction: The last phase of the composition life cycle forms the dissolution of the composition. In the example of T it means either that the booked services were used by the client or cancelled before. In this phase we identified the same semantic relationships as in the *construction* phase, i.e., *component commit*, *component history*, and *subact*.

In the next section we discuss integration options which are later linked to the previous defined semantic relationships.

3.2 Analysis of Integration Options

In [18] we elaborated the differences between synchronisation and constraints, where the semantics of the element activity and state are crucial. However, in Web service execution semantics, states do not exist explicitly, whereas in BPEL a state is implicitly determined by the values of variables. In Section 3.4 variables are mapped to OWL-S process ontology as message types. Therefore the following analysis of integration options which is based on synchronisation and constraints is applicable to Web services if we consider states as received messages and activities as atomic processes.

We have identified three scenarios where integration is required, (1) **synchronisation of activities**, (2) **synchronisation constraints between activities and states**, and (3) **constraints between states**. Synchronisation deals with the coupling of two Web services, e.g., through message exchange and constraints defined conditions under which an atomic process can be executed, e.g., if a specific message was received previously.

To explain this, we use the same example as in the previous section, the composite Web service T which consists of the atomic processes $T_1, \ldots, T_m$ and the component Web service F holding the atomic processes $F_1, \ldots, F_n$.

Synchronisation of activities: In the case of activity synchronisation we observe both the control flow and the orthogonality of the relationship between T and F. We identify two simple integration options which integrate two activities, (1) *blocking* and (2) *non blocking*. In the case where several activities need to be integrated we propose a further four complex integration options (3) *future synchronisation*, (4) *ordering*, (5) *execute if available*, and (6) *cancel if unsuccessful*. They may apply a combination of simple integration options as defined below:

(1) **Blocking (*block*)**: A *blocking* synchronisation holds if T sends a message M to F and wait for an incoming message from F as an answer to M. The integration option *block* is defined as $block(T_x, F_y)$ where T_x is the atomic process which sends M and F_y the atomic process which receives M.
(2) **Non blocking (*nblock*)**: A non blocking synchronisation between T and F exists if T sends a message M to F and does not wait for an incoming message from F. The option *non blocking* is defined as $nblock(T_x, F_y)$ where T_x sends M and F_y receives it.
(3) **Future synchronisation (*future*)**: The future synchronisation offers

the opportunity to synchronise the control flow of two composite processes at a later point in time, e.g., T_1 sends a message to F_1 and T continues execution to a synchronisation point, e.g., T_m and waits for an incoming message from F.

This integration option is defined as $future(S, E)$ where S is a pair of atomic processes which start the synchronisation defined as $S = \{T_1, F_1\}$ and realised through $nblock(T_1, F_1)$. The second pair of activities $E = \{F_n, T_m\}$ ends the synchronisation and is realised through $nblock(F_n, T_m)$. Two constraints must be satisfied during runtime, (1) during the execution the instance of T must pass T_m after T_1, and (2) the instance of F must enter F_n after F_1.

(4) **Ordering (*order*)**: If T has to exchange several messages with other services it might be necessary to order the communication with them, e.g., T might to finish booking the hotel and flight first before booking the limousine. The integration option is defined as $order\_act(I)$ where I is a set of integration options in a specific order. If A is the set of all integration options identified within a Web service then $I \subseteq A \times A$.

(5) **Execute if available (*avail*)**: This option ensures atomicity of activity execution. Atomicity might be necessary between the invocation of atomic processes by T, e.g., the atomic processes in $F1$ and $H1$ "confirm booking". In [18] we use a control flow template which first checks the availability of a successful execution of an activity and then executes it. The option *avail* is defined as $avail(E)$ where E is a set of activities that need to be executed successfully, e.g., $E = \{F_1, H_1\}$.

(6) **Cancel if unsuccessful (*cancel*)**: An alternative solution to *avail* is proposed by *cancel*. The difference is that instead of checking the availability of all activities in E they are executed first and if one of them fails all other activities are cancelled, similar to a rollback. It is defined as $cancel(E)$.

Synchronisation and constraints between activities and states: The integration of activities and states of two Web services is divided into (1) synchronisation and (2) constraints:

(1) **Synchronisation**: In this case an activity sets another Web service into a specific state. As mentioned before the state of a Web Service is determined by a message it sends. So for setting a Web service W into a specific state S, a message including S must be sent which must then be answered by W with S, e.g., the atomic process "cancel flight" in F sends a message "flight is not valid anymore" to T. As a result the travel schedule of T is not valid anymore and T sends back a confirmation F. This integration option is defined as $setInState(F_y, T, M)$ where F_y is an atomic process in a Web service which sets another Web service T into a state defined by M.

(2) **Constraints**: There are two types of constraints depending on whether

an activity may start or must start. The first is defined as $isAbleToStart(T_x, F, M)$ where T_x is an atomic process in a Web service, F is the Web service which must be in a specific state represented by the message type M, e.g., T_x is "change flight", F is the flight booking service and M is the message "flight was booked successfully". In this example it is only possible to change a flight after a previous booking was successful and a booking exists already. T has to send a message to F to ask for the state before executing T_x if M was not received beforehand.

The second is defined as $mustStart(T_x, F, M)$ where T_x is an atomic process which must start if Web service F is in a specific state determined by message M, e.g., if a flight is cancelled by F then T must inform the owner of the ticket about it. T_x is the atomic process which informs the owner and M the message "flight is cancelled".

Constraints between states: The last scenario deals with the relationship between states. A state of a component object may change the state of composite object, e.g., the state "tyre is flat" changes the state of car to "car is not ready to drive". This constraint is similar to the constraint defined by $mustStart()$ but in this case there is no activity involved. We define this constraints as $isInState(S, D)$ where S and D represent two different states. An object O must be in state S if there is an object in D which depends on O.

We linked each integration option as a preferred or alternative way of integration to the semantic relationships defined in Section 3.1. A table of all possible connections between semantic relationships and integration options can be found in [18]. The advantage of this approach is that the user need only deal with Web service relationships and the choice of the preferred integration option (if the situation is such that alternative options exist). However, the user (developer) does not need to deal with the technical integration of Web services.

3.3 Integration of Web services described in OWL-S

For applying this business process integration approach on Web services we need to map the relationships and integration options to a language which supports the description of semantics and can be interpreted by computer systems. We decided to use OWL-S as a suitable language used for Web services for the reasons mentioned in Section 1. This section deals with the implementation of the previous described integration approach in OWL-S.

An ontology defines a common vocabulary for individuals, organisations, and applications that need to share information in a domain. Having an ontology in the domain will reduce the number of combinations in the integration

mapping, as we only need to map the business processes defined in any of the languages to the ontology which can then be mapped back easily to other languages, for example, which are mentioned in the end of section 2 (also refer to [14]). Many researchers had made efforts to establish an unified resource model for Web entities [22]. Based on that, a Knowledge Grid environment has been set up beyond a peer-to-peer platform, where semantic overlay layer plays a critical role in offering better scalability and performance for query of semantic objects [23]. Recently, an ontology oriented Web service modeling framework was proposed in [24], and a special Web service modeling onotology working group (www.wsmo.org) has been formed. A comparison between WSMO and OWL-S can be found in [25].

NIST (National Institute of Standards and Technology) has proposed one specific process-oriented ontology language, namely Process Specification Language (PSL), which is based on the Knowledge Interchange Format (KIF) the predates OWL and OWL-S [26]. Like PSL, OWL-S is an attempt to provide an ontology for Web services, in this case within the context of the OWL language.

According to OWL-S, service descriptions are divided into three parts, which are characterised by the kind of knowledge provided about a service [9]. The service *profile* describes what the service required from users or agents and what it provides to them. The service *model* describes the service's process model (the control flow and data-flow involved in using the service). Specifically, the OWL-S service model defines three types of processes (atomic, simple and composite). It is designed to enable automated composition and execution of services and of the three parts is the one most closely related to the BPEL process model. The service *grounding* connects the process model description to communication level protocols and message descriptions in WSDL. These components are annotated with classes of well-defined types that make the service descriptions machine-readable and unambiguous. Additionally, the ontological structure of types allows type definitions to draw properties from the hierarchical inheritance and relationships to other types.

We have chosen OWL-S as a candidate for describing business processes and their relations in a semantic context. As an immediate application scenario of such mapping, we have an in-house peer-to-peer decentralised workflow-based e-service system, SwinDeW-B, which was based on SwinDeW [27,28]. Without OWL-S semantics' support, neither XPDL nor BPEL could organise well-formed negotiation and coordination between peers, who enacted real tasks and activities. Thus, the integration of BPEL and OWL-S introduced in this paper becomes more essential for peers to play complementary roles in semantic Web service contexts [29]. In following we address the mapping from BPEL to OWL-S and the implementation of it.

3.4 Mapping from BPEL to OWL-S

We will concentrate on the OWL-S service model (also known as process model - process ontology). The top level class of OWL-S process ontology is ***process***. A process can have any number of inputs and outputs, preconditions, effects and participants. There are three disjoint subclasses of the ***process*** class. Atomic processes can be directly invoked, have no sub-processes and execute in a single step (from the perspective of the service requester). Composite processes can be decomposed into other (atomic or composite) processes, which are linked by control constructs such as *sequence* or *if-then-else*. In contrast to atomic processes, they cannot be directly invoked. And simple processes cannot be directly invoked, but, like atomic processes, they are viewed as having single-step executions. The features that distinguish and differentiate BPEL from OWL-S in terms of expressiveness, semantics, automated composition and execution, fault handling and querying mechanisms were identified in [12], and some difference between BPEL and DAML-S, predecessor of OWL-S, had been compared in [13]. The mapping of necessary WSDL elements to the OWL-S process model is used and extended in our mapping tool BPEL2OWL-S, an extension of initial implementation work described in [11].

Workflow and e-service languages like XPDL and BPEL share commonalities in basic structures and elements, and OWL-S acts as a direct successor of DAML-S and PSL. Therefore, without losing generality, the following subsections will describe the detailed mapping from BPEL elements and semantics to OWL-S process ontology.

Processes and Business Partners:

A BPEL executable process does not represent any abstract view of the process and can be directly invoked. Hence it will be mapped onto an OWL-S atomic or composite process, depending on the internal activity of the executable process. OWL-S atomic processes can be directly invoked and composite processes can be made explicitly invocable by setting the ***invocable*** property of the composite process to true, thus, making this type of mapping valid. A BPEL process can have only one main activity in it. Thus, the selection of the OWL-S process (atomic or composite) for the mapping will depend on this activity.

On the other hand, a BPEL abstract process cannot be directly invoked and also represents the abstract view of the process and hence will be mapped onto OWL-S simple processes. An OWL-S simple process can be thought of as a view on either an atomic or composite process [9]. Simple processes provide a means of characterising other processes at various levels of granularity, for the purpose of planning and reasoning. They give additional characterisation

of how they work, in terms of other processes (atomic or composite) and are not directly invocable (abstract process).

Variables and Data Flow:

In WSDL2DAML-S [11], only the mapping of port types and operations to their corresponding DAML-S atomics processes is presented. It does not reflect the mapping of WSDL messages in DMAL-S process ontology. We have extended the mapping of WSDL2DAML-S by including the WSDL messages in our OWL-S based workflow process ontology.

WSDL messages are used to represent the abstract definition of the data being transmitted in and out of the processes [2]. WSDL messages consist of one or more logical parts. Each part is associated with a type from some type system (data type defined or built-in XML Schema: XSD). Since we know the data types of parts of the messages by the ***type*** attribute of the part element, all the messages will be represented as OWL classes. The type of the class will not be restricted to any particular data type but will use the OWL object (i.e. ***Thing***) as the data type. Part of each message will be mapped onto the properties of their corresponding OWL class and the data type of the property will be based on the type specified for that part. Variables in BPEL provide the means for holding messages that constitute the state of a business process. The messages held are often those that have been received from the partners or are to be sent to the partners via primitive activities. The type of each variable may be a WSDL message type, an XML schema simple type or an XML schema element. The ***messageType***, ***type*** or ***element*** attributes are used to specify the type of a variable. Variables that are defined by the ***messageType*** attribute represent a WSDL message thus share the same data type of the WSDL message it refers to. Hence, such types of variables are mapped onto the data types in the OWL-S process ontology (same as WSDL messages mapped). Variables defined by the ***messageType*** attribute (representing a WSDL message type) in a BPEL process will be mapped in the same way as the WSDL messages mapped onto the data types in OWL-S process ontology using the OWL class.

A Data Flow OWL file will be produced as one of the outputs for this whole mapping process. Data Flow OWL contains the process annotations for Process OWL which will relate various process parameters to each other as defined in Process OWL, which defines the concrete activities in the process model. Therefore the data flow is separated from the process definition. The inputs and outputs of the atomic processes derived from <receive>, <reply> and <invoke> activities are described accordingly in Data Flow OWL. The OWL-S ***valueOf*** class is used to refer to their respective super class atomic process's inputs and outputs. When composing atomic processes into composite processes (i.e. when a composite process has atomic sub-processes in

it), it is crucial that the inputs and outputs of the sub-processes are related to each other. This is addressed using the OWL-S ***valueOf*** class (represents that two parameters used for referencing are equal), used similarly as above for referencing the inputs and outputs of the derived atomic processes from primitive activities to their corresponding super class atomic process's inputs and outputs. Furthermore, we represent the referencing of the inputs and outputs of the derived composite processes from structured activities in Data Flow OWL using the OWL-S ***valueOf*** class. Thus, the complete data flow for both atomic processes and composite processes can be derived from BPEL primitive and structured activities respectively.

Activities:

BPEL primitive activities are simple activities with single step executions. These activities are commonly used as request-response operations, for assignments and for throwing exceptions in the process. OWL-S atomic processes share the same concept. They are the basic units of implementation, a black-box representation which does not describe how such processes work. They are normally used together with other types of processes (simple or composite) to represent the workflow and business logic in the process. All primitive activities can be mapped onto atomic processes.

BPEL structured activities describe how a business process is created by composing the primitive and structured activities. They prescribe the order in which a collection of activities (both primitive and structured) takes place and express the control patterns, data flow, handling of faults and external events. The structured activities include: ordinary sequential control between activities which are provided by <sequence>, <switch> and <while>; non-deterministic choice based on external events which are provided by <pick>; concurrency and synchronisation between activities which are provided by <flow>.

OWL-S composite processes are composed of sub-processes (atomic or composite) and share the same concept of BPEL structured activities. Thus, the structured activities in BPEL will be recursively mapped onto OWL-S composite processes. Every composite process has a control construct associated with it. The control constructs (sequence, choice, repeat-while, etc.) are closely related to BPEL structured activities. The mapping of the structured activity will be based on the type of the activity which will be used to determine the type of the control construct for the derived composite process. The inputs and outputs of composite processes are derived from the corresponding inputs and outputs of atomic sub-processes and will be computed normally. OWL-S has not specified the function and use of the ***condition*** class so far. Our current practice is to merely carry over the ***condition*** content from BPEL into the OWL-S ***condition*** for reference purposes, without any changes in surface

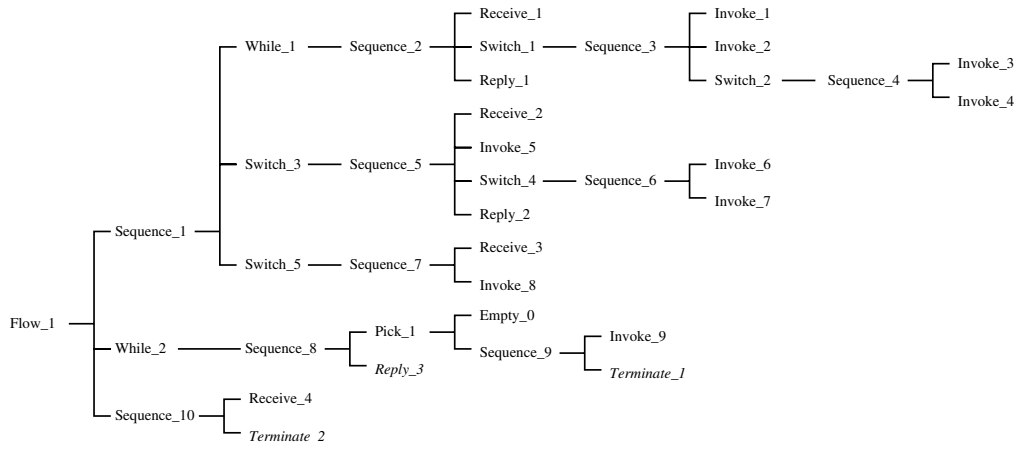

Fig. 1. An example of BPEL flow.

syntax.

In the following section we apply these mapping mechanisms on a case study.

4 Case Study

In this section we will present a case of mapping from BPEL to OWL-S through an example [12]. Figure 1 shows an example BPEL flow which we will use for mapping illustration. Note that there are two <terminate>s in the figure. Because OWL-S has not specified how to terminate a process yet, we temporarily ignore the <terminate> activity. That is why in the final OWL-S process there is not atomic process named ***terminate*** as a counterpart of the one in the BPEL process.

Simple processes are used to abstract other processes (atomic or composite). The level of abstraction again depends on the main activity of the BPEL abstract process and will be based on the detailed mapping of the activities. A simple process that abstracts an atomic process is realised by that process which is done by using the ***realizedBy*** property of the simple process and the ***realizes*** property of the underlying atomic process. A simple process that is abstracted to a composite process is expanded to that process which is done by using the ***expandTo*** property of the simple process and the ***collapsesTo*** property of the underlying composite process [9]. Using the top level attribute (i.e.,***abstractProcess***) of the process definition, we can identify whether the BPEL process is an abstract or executable process, thus, allowing us to map the process onto an appropriate OWL-S process.

Business partners' definitions are optional and need not cover all partner relationships/links defined using ***partners*** and ***partnerLinks*** elements in the

process definition. All the partners, if defined in the BPEL process, will be represented as the participants of the main OWL-S process by using the participant property of the OWL-S process. A participant of an OWL-S process can be any kind of OWL object (i.e. ***Thing***). However, the ***participant*** property can be specialised by restricting it to agents, objects, entities, etc. [9].

4.1 Primitive Activities

The <receive>, <reply> and <invoke> activities of BPEL use the ***portType*** and operations defined in the WSDL document to send and receive messages and invoke operations in the process. As the ***portType*** and operations in WSDL are mapped onto the OWL-S atomic processes together with their corresponding inputs and outputs, we will also map these three activities onto the atomic processes. We make these processes as the sub-types of those defined atomic processes in Process OWL, representing corresponding ***portType*** and operations in these activities.

The <receive> activity allows the business process to do a blocking and waits for a matching message to arrive. Assumed that ***portType*** and operations from the related WSDL are mapped onto the corresponding OWL-S atomic processes; the <receive> activity will be mapped onto an atomic process which will be the sub-class of an atomic process derived from the WSDL. The super class, an atomic process, will be identified by using the ***portType*** and ***operation*** attributes of the <receive> activity. This will allow us to map the activity onto the accurate atomic process and sub-type process. Since the <receive> activity only receives the message, such a message will be mapped as an input of the atomic process derived from the <receive> activity with no outputs for the process. The type of this input will be based on the ***variable*** attribute of the <receive> activity which specifies the name of the variable defined in the BPEL process. As we use variables defined in the BPEL process as data types in Process OWL, we can identify the data type (in Process OWL) of the input of the atomic process derived from the <receive> activity using the variable name. Thus, this input of the atomic process will then be the same as one of the inputs for the super class, atomic process. The <reply> activity allows a business process to send a message only and hence will be mapped onto an OWL-S atomic process with outputs only. The mapping of the <invoke> activity combines the mapping of both <receive> and <reply> activities as discussed above. The derived atomic process will have both the inputs and outputs based on the ***inputVariable*** and ***outputVariable*** attributes of the <invoke> activity respectively. The examples can be found in [12].

The <throw> activity generates a fault from within the business process [4].

The type of the fault generated is associated with the ***faultVariable*** attribute of the <throw> activity. The ***faultVariable*** attribute specifies the variable name which is represented as a data type in Process OWL. The <throw> activity will be mapped onto an OWL-S atomic process with one output of the same data type as of the fault variable specified in the <throw> activity. There will be no inputs for the atomic process derived as this activity is used to generate faults and send fault messages. The derived atomic process will be of its own type and will not inherit from any atomic processes derived from WSDL as the <throw> activity does not specify the port type and operation for communication.

4.2 Structured Activities

4.2.1 Sequence, Switch, While, Pick

The <sequence> activity contains one or more primitive or structured activities that are performed sequentially based on the order listed within the <sequence> activity. The <sequence> activity completes when the final activity listed in the <sequence> activity is completed. An OWL-S composite process with control construct of type ***Sequence*** will be used for this mapping. It will list the sub-processes (atomic or composite) which will be performed in the order in which they are listed. The type of the sub-processes will be based on the internal activities of <sequence> in BPEL. The components of the derived composite process can be either atomic processes or composite processes depending upon the internal activities in <sequence>. All the sub-processes of the derived composite process are recursively declared depending on the type of the process in the process.

The <switch> activity supports the conditional behaviour in the pattern that occurs quite often. The activity consists of an ordered list of one or more conditional branches defined by the ***case*** elements, followed optionally by an ***otherwise*** branch. The activity of the branch whose condition holds is performed and the <switch> activity gets completed. If no branch condition holds then the activity of the ***otherwise*** branch is performed. An OWL-S composite process with control construct of type ***Choice*** will be used for this mapping. An OWL-S ***Choice*** control construct allows the selection of a process from the list of its sub-processes.

The <while> activity supports the repeated performance of a specified iterative activity in the business process. The iterative activity is performed until the given condition no longer holds. An OWL-S composite process with control construct of type ***Repeat-While*** will be used for this mapping. ***Repeat-While*** allows a sub-process to iterate until the ***whileCondition*** is true. The

iterative activity of the <while> activity will be mapped onto the sub-process in the ***Repeat-While*** control construct based on the type of the activity and the condition in the <while> activity will be mapped onto ***whileCondition*** of the ***Repeat-While*** control construct.

The form of the <pick> activity is a set of branches of the form event/activity, and exactly one of the branches will be selected based on the occurrence of the event associated with it before any others. After the <pick> activity has accepted the event, the activity of the appropriate branch is performed and no more events are accepted by the <pick> activity after this. Thus, only one activity is performed from the set of activities in <pick>. The <pick> activity has one or more ***onMessage*** events, that contains one activity each and zero or more ***onAlarm*** events which also contains one activity each [4]. The mapping of this activity is done in the same way as for the <switch> activity. An OWL-S composite process with the ***Choice*** control construct will be used for this mapping.

4.2.2 Flow

The <flow> activity provides concurrency and synchronisation. The standard attributes and elements for the activities nested within a <flow> activity are especially significant because they exist to provide flow-related semantics. The most fundamental semantic effect of grouping set of activities in a <flow> is to enable concurrency. A <flow> activity completes when all the activities in the <flow> have completed. More generally, a <flow> activity creates a set of concurrent activities directly nested within it. It further enables the expression of synchronisation dependencies between activities that are nested directly or indirectly within it. The ***link*** construct is used to express these synchronisation dependencies [4]. Furthermore, the standard elements (***source*** and ***target***) are used to link the activities within the <flow> activity.

An OWL-S composite process with the ***Concurrent-Sync*** control construct will be used for this mapping. An OWL-S ***Concurrent-Sync*** control construct is a sub-class of the ***Split-Join*** control construct which allows concurrent execution of a set of sub-processes, with barrier synchronisation. A complete execution of all the sub-processes in the ***Concurrent-Sync*** is required to complete the process the same as the <flow> activity in BPEL. Here is an example: the <flow> activity appears in BPEL (refer to Flow 1 in Figure 1):

```
<flow name="Main_Flow">
    <sequence name="CREATION_SEQUENCE">
        ...
    </sequence>
```

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

It is mapped to the following OWL-S expression:

```
<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>
```

4.3 Implementation

The transformation from BPEL to OWL-S has been implemented in the BPEL2OWL-S tool [12]. It maps specifications from BPEL to OWL-S and uses the same BPEL and/or WSDL inputs and produces the outputs as Process OWL and Data Flow OWL files. The classes, reusable packages, and examples are available from (www.it.swin.edu.au/centres/cicec/bpel2owls.htm).

Some of the distinct features of the BPEL2OWL-S tool are described there. Object Explorers are used to represent the object view of the input (BPEL and WSDL) and the output (Process OWL) files. The tool allows easy navigation and representation of the hierarchical inheritance and relationships between the objects and their properties using a tree structure. A Project Validator is used to associate a correct service WSDL description for a given BPEL process in mapping. The tool provides help with brief descriptions for the important functionalities of the tool and how to interact with the tool. Users can also simultaneously view the tree structure representing the object view of the source files (inputs and outputs) in the tool interface as well as other Web browsers. The tool supports multiple WSDL files which are along with a BPEL file. WSDL files are distinguished in terms of slave WSDL and master WSDL. The master WSDL file is the main one that all slave WSDL files refer

to. WSDL serves as the foundation of this mapping process. It describes all the data that are used in a business process.

5 Discussion and Related Work

While OWL-S offers more powerful description capabilities for business processes than UML, by now ***condition*** is still a "place-holder" which awaits further work from the OWL-S community (namely surface syntax). However ***condition*** is a 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 unfinished issue which is vital to manage a large scale business system. Because of the great traffic there are surly some concurrent processes and the ability to deal with them is an important criterion to measure the quality of the whole system. Compared to BPEL, a complete workflow language, OWL-S needs time and efforts to get mature. Therefore some functions and activities in BPEL, like fault handling, value assignment, correlation sets and so on, are far beyond the current OWL-S capability.

Furthermore, while mapping the <flow> activity to the OWL-S Concurrent-Sync composite process, we cannot represent the synchronisation of and links between the sub-processes in the derived composite process due to the lack of waiting and synchronisation features in the current version of OWL-S. All the above findings can potentially contribute to the ongoing development of both BPEL and OWL-S. Tools developed in a Web service and Semantic Web context are making the integration easier, but are still far from practical applicability. Current work on the OWL-S process ontology [24,30] still lacks crucial concepts required for the composition of complex Web services and business processes. These restrictions also hampered the work of [10,11] who tried to adapt BPEL and WSDL for Semantic Web or OWL-S-based applications. They map WSDL service descriptions to an OWL-S service profile, which only deals with inputs, outputs, preconditions and effects (IOPE) of related processes.

Ultimately, these restrictions need to be overcome in order to represent complete workflow and business logic of BPEL in OWL-S process ontology, as what has been aimed at in [10]. Moreover, to build modern Semantic and Knowledge Grid applications [31,32], a translatable workflow ontology is an essential foundation. Therefore, we need to consider enhancing our tool in the near future, in order to improve the efficiency and flexibility in the mapping. The list below identifies the issues of crucial BPEL language constructs that need to be considered in future mappings to achieve the accuracy and reliability in mapping of the BPEL specification to OWL-S based workflow process

ontology: (1) effective use of variables and assignment statements for mapping of the <assign> activity; (2) ability to explicitly terminate a business process; (3) allowing effective synchronisation and blocking in the business processes; (4) describing the relationships between the various partners of the process; and (5) support for fault and error handling.

Nevertheless, the derived mapping specifications and implemented tools in this paper are based on and strictly conform to the language specifications and standards. Our work significantly extended the work reported in [11]. With the reconstruction of MyBPEL and the exploitation of new versions of UML [8], we are improving BPEL2UML-AD to incorporate new semantics capabilities suitable for business process meta-modeling. At the same time, the BPEL2OWL-S tool has been announced and released for world-wide researchers' reference and testing. To date it has attracted many responses and we believe it will contribute to the evolution of process ontology models, i.e. OWL-S, which has just reached a new beta version very recently [9]. For example, there are arguments about at which abstract level, class or instance, we should map processes onto OWL-S descriptions. These discussions are out of the scope of this paper and can be found in [14], where foundations of RDF(S) are revisited. We are also implementing mapping from XPDL to newer versions of OWL-S.

6 Conclusion

In this paper, with the analysis of business process integration options, we have examined the impact of ontology based models on process integration. The lessons come from the earlier but unsatisfactory partial transformation tool, BPEL2UML-AD, which maps BPEL to the traditional UML-based meta-modeling framework. We realise that OWL-S enables definitions of the Web services content vocabulary in terms of objects and complex relationships between them including classes, sub-classes, cardinality restrictions, hierarchical inheritance. It also provides a shared set of terms describing the application domain with a common understanding for sharing information and knowledge and with well-defined semantics. Using OWL-S as ontology in our project overcomes not only the limitations and weaknesses of BPEL but also some of the limitations of XML, and solves data integration and interoperability problems and issues faced today in the Web Service world.

We have demonstrated the mapping from the BPEL specification to an OWL-S based workflow process ontology by extending the work done in previous WSDL-to-OWL-S mappings, and by developing a GUI-based BPEL2OWL-S tool to support this mapping. By integrating the separate efforts on BPEL and OWL-S, deployment and enactment of Web services environments in the work-

flow style will become more practical. By improving our mapping specification from BPEL to OWL-S, we believe that we can achieve the accuracy, efficiency and reliability in our mapping. We will be able to represent the complete and accurate business logic of a BPEL process in OWL-S based workflow process ontology. These efforts further enable our integration framework of business process definitions and their application in new environments like Grid and peer-to-peer based e-services deployment and coordination.

Both language specifications, BPEL and OWL-S, are new, work-in-progress and evolving and perceptible languages. This will allow us to track the changes in the specifications of these languages in order to extend our mapping between these standards and specifications. In the near future, we will enrich service semantics in the peer-to-peer context and enhance the SwinDeW-B prototype with more advanced features.

References

[1] J. Rosenberg, The Critical Need for Monitoring and Analysis, Web Services Journal, 3(11), http://webservices.sys-con.com/read/39904.htm (2003).

[2] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, W3C, http://www.w3.org/TR/wsdl (2001).

[3] S. Weerawarana, F. Curbera, Business Process: Understanding BPEL4WS, Part 1, http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/ (2002).

[4] T. Andrews, F. Curbera, et.al., Business Process Execution Language for Web Services Version (BPEL4WS) 1.1, IBM, BEA, Microsoft, SAP, Siebel, http://www.ibm.com/developerworks/library/specification/ws-bpel/ (2003).

[5] WfMC, Workflow Process Definition Interface (XPDL) 1.0, WfMC, http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf (2002).

[6] A. Calì, D. Calvanese, D. G. Giuseppe, M. Lenzerini, P. Naggar, F. Vernacotola, IBIS: Semantic Data Integration at Work, in: Proc. of the 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), Vol. 2681 of Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 79–94.

[7] T. Reiter, Transformation of Web Service Specification Languages into UML Activity Diagrams, Master's Thesis, Johannes Kepler University of Linz, Austria, and University of South Australia, Australia (Mar 2005).

[8] OMG, Unified Modeling Language Specification Version 2.0, OMG, http://www.omg.org/uml/ (2004).

[9] Defense Advanced Research Projects Agency (DARPA), OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/ (2004).

[10] D. Mandell, S. McIlraith, Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, in: Proc. of the 2nd International Semantic Web Conference (ISWC 2003), Vol. 2870 of Lecture Notes in Computer Science, 2003, pp. 227–241.

[11] M. Paolucci, N. Srinivasan, K. Sycara, T. Nishimura, Toward a Semantic Choreography of Web Services: From WSDL to DAML-S, in: Proc. of the 1st IEEE International Conference on Web Services (ICWS 2003), Las Vegas, Nevada, USA, 2003, pp. 22–26.

[12] J. Shen, Y. Yang, C. Zhu, C. Wan, From BPEL4WS to OWL-S: Integrating E-Business Process Descriptions, in: Proc. of the 2nd IEEE Conference on Services Computing (SCC 2005), Orlando, Florida, USA, 2005, pp. 181–188.

[13] S. McIlraith, D. Mandell, Comparison of DAML-S and BPEL4WS (initial draft), Tech. rep., Knowledge Systems Laboratory, Stanford University, http://www.ksl.stanford.edu/projects/DAML/Webservices/DAMLS-BPEL.html (2002).

[14] J. Shen, Y. Yang, Extending RDF in Distributed Knowledge-Intensive Applications, Future Generation Computer Systems 20 (1) (2004) 27–46.

[15] W.M.P. van der Aalst, A.H.M. ter Hofstede, YAWL: Yet Another Workflow Language, Information Systems 30 (4)(2005) 245–275.

[16] P. Wohed, W.M.P. van der Aalst, M. Dumas and A.H.M. ter Hofstede, Analysis of Web Services Composition Languages: the Case of BPEL4WS, in: Proc. of the 22nd International Conference on Conceptual Modeling (ER 2003), Vol. 2813 of Lecture Notes in Computer Science, 2003, pp. 200–215.

[17] A. Kleppe, J Warmer and W. Bast, MDA explained - The Model Driven Architecture: Practice and Promise, Addison-Wesley Publishing Company, 2003, Object Technology Series.

[18] G. Grossmann, Y. Ren, M. Schrefl, M. Stumptner, Behaviour Based Integration of Composite Business Processes, in: Proc. of the 3rd International Conference on Business Process Management (BPM 2005), Vol. 3649 of Lecture Notes in Computer Science, 2005, pp. 186–204.

[19] M. Preuner, M. Schrefl, Requester-Centered Composition of Business Processes from Internal and External Services, Data and Knowledge Engineering 52 (1) (2005) 121–155.

[20] M. Snoeck, W. Lemahieu, F. Goethals, G. Dedene, J. Vandenbulcke, Events as Atomic Contracts for Component Integration, Data and Knowledge Engineering 51 (1) (2004) 81–107.

[21] G. Grossmann, M. Schrefl, M. Stumptner, Classification of Business Process Correspondences and Associated Integration Operators, in: Proc. of the 5th International Workshop on Conceptual Modeling Approaches for e-Business (eCOMO 2004), Vol. 3289 of Lecture Notes in Computer Science, 2004, pp. 653–666.

[22] H. Zhuge, Resource Space Model, Its Design Method and Applications, Journal of Systems and Software 72 (1) (2004) 71–81.

[23] H. Zhuge, X. Sun, J. Liu, E. Yao, X. Chen, A Scalable P2P Platform for the Knowledge Grid, IEEE Transactions on Knowledge and Data Engineering 17 (12) (2005) 1721–1736.

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

[25] R. Lara, D. Roman, A. Polleres, D. Fensel, A Conceptual Comparison of WSMO and OWL-S, in: Proc. of the 2nd European Conference on Web Services (ECOWS 2004), Vol. 3250 Lecture Notes in Computer Science, 2004, pp. 254–269.

[26] C. Schlenoff, G. Tissot, J. Valois, J. Lubell, J. Lee, The Process Specification Language (PSL) 1.0, National Institute of Standards and Technology (2000).

[27] J. Yan, Y. Yang, G. K. Raikundalia, SwinDeW - A Peer-to-Peer based Decentralised Workflow Management System, IEEE Transactions on Systems, Man and Cybernetics, Part A, accepted for publication, http://www.it.swin.edu.au/personal/yyang/papers/SwinDeW-TSMC.pdf.

[28] J. Shen, Y. Yang and Q. H. Vu, SwinDeW-B: A P2P Based Composite Service Execution System with BPEL, in: Proc. of Workshop on Dynamic Web Processes (DWP 2005) within the 3rd International Conference on Service Oriented Computing (ICSOC), available as IBM RC23822, Amsterdam, the Netherlands, 2005, pp.73–84.

[29] J. Shen, Y. Yang, J. Yan, A P2P based Service Flow System with Advanced Ontology-based Service Profiles, Advanced Engineering Informatics, to appear.

[30] S. Narayanan, S. McIlraith, Analysis and Simulation of Web Services, Computer Networks 42 (5) (2003) 675–693.

[31] H. Zhuge, Semantics, Resource and Grid, Future Generation Computer Systems 20 (1) (2004) 1–5.

[32] H. Zhuge, The Knowledge Grid, World Scientific Publishing Co. Singapore, 2004.