In Proceedings of ICEIS'05(International Conference on Enterprise Information Systems),May 23-27,pp. 208-213, Miami-USA, 2005.

APPROACHES OF WEB SERVICES COMPOSITION
Comparison between BPEL4WS and OWL-S

Daniela Barreiro Claro[1 2], Patrick Albers
[1] *4 rue Merlet de la Boulaye, BP 30926 49009 Angers cedex 01, France*
Email: daniela.claro@eseo.fr, patrick.albers@eseo.fr

Jin-Kao Hao
[2]*University of Angers Faculty of Sciences – LERIA 2, Boulevard Lavoisier, 49045 Angers Cedex 01 - France*
Email: hao@info.univ-angers.fr

Keywords: Web Services composition, semantic web, owl-s, bpel4ws

Abstract: Web Services technologies allow interaction between applications. Sometimes a single service given alone does not meet user's needs. In this case, it is necessary to compose several services in order to achieve the user's goal. For composing web services, we developed an example using two main approaches: the first one is BPEL4WS, a Business Process composition, and the other is OWL-S, an ontology specifically for web services composition. In this paper we explain and compare the features of these two approaches and the manner of each one does a web service composition.

1 INTRODUCTION

Many services are available around the Web and people start using them to achieve their goals and facilitate interaction between systems.

A web service can be characterized as a method that is available on Internet and that does not have any graphics interface. For instance we can enumerate many web services like: zip code web service in which an user sends a zip code, and it returns the streets name; booking a flight, in which a user gives the day periods and the web service returns the flight numbers; a fiscal note integration between e-business negotiations, in which an enterprise A sends a purchase note to enterprise B, and this enterprise B can check online the amount and the quantity available and do the payment as soon as possible, minimizing time between e-business (Schroeder, 2004).

Nevertheless there are many services around the web, each one, taken alone, has a limited utility. For example, if a user wants to travel, it is not only sufficient to book a flight, but also to care about reserving a hotel, renting a car, getting entertained, and so on. The user needs to execute all these services manually and these tasks can take long time and effort.

For that reason, the notion of composite services starts being used as a collection of services in order to achieve a particular goal. Indeed, this composition may utilize as many web services as necessary to achieve one goal (Berargi, 2003).

In order to integrate complex services, industry proposed, at first, technologies such as: WSCI (Web Service Choreography Interface), XLANG, WSFL (Web Service Flow Language), WSCL (Web Service Composition Language) and more recently BPEL4WS (Business Process Execution Language for Web Service) (Curbera, 2003). The later is a combination of Microsoft's XLang and IBM's Web Service Flow Language (Peltz, 2003).

On the other hand, the semantic web approach has increased and has taken a considerable position in web services composition. The advance of OWL-S allows an unambiguous web service description avoiding problems of retrieving wrong services (OWL-S, 2004) (McIlraith, 2001).

As far as composing web services, we have implemented an example in which an airplane ticket and hotel can be booked and a car rented through a travel agency. Indeed, we have three autonomous web services, each independently executing an activity. In our composition, we compose these services in a special way: the first two services, bookAirplane and bookHotel will be executed in parallel because the inputs given by the user will be the same for both services. Thus, we can execute them at the same time and profit from a quicker

execution. Conversely, the third service for renting a car needs to be executed after the bookHotel service. This is mandatory because we consider that anybody that will rent a car does not want to reach the car rental company. Thus, thinking like that, we proposed that the service bookHotel will have as output the area where it is located. The rentCar service will have as input this area parameter, so the company's car will be located in the same area as the hotel. As a result, the bookHotel service and rentCar service in our composition must be executed in sequence to assure that the output "area" will be available and passed correctly. The figure 1 shows this schema.

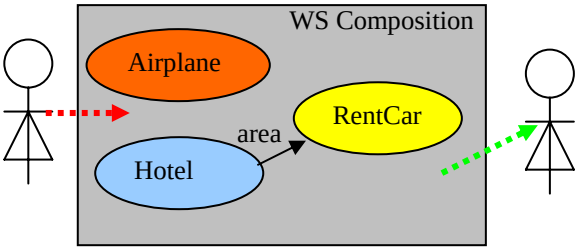

Figure 1: WS Composition example

In this paper we propose a web service composition schema in order to show the characteristics between composing using the two ongoing processes: BPEL4WS and OWL-S.

This paper is organized as follow: the second section describes a composition using BPEL4WS. The third one shows this composition using OWL-S, including semantic effects. The fourth section explains the comparison between these technologies. Finally we show our conclusions.

2 COMPOSING USING BPEL4WS

The web services composition using BPEL4WS allows the manipulation of services as activities and processes. Actually, the BPEL4WS language is a merge between Microsoft's XLang and IBM's WSFL, but all of them are considered as a web service flow language (van der Aalst, 2003).

As an executable process implementation language, the role of BPEL4WS is to define a new web service by composing a set of existing ones. The interface of the composite service is described as a collection of WSDL PortTypes.

A BPEL4WS process defines the roles involved in a composition as abstract processes. A buyer and a seller can be represented by two roles. They are expressed using partner link definitions. We can have a role for each web service that is composed and does some activity. In order to integrate services, they are treated as partners that fill roles (Mandell, 2003). BPEL4WS depends directly on the WSDL of the service. A business process defines how to coordinate the interactions between a process instance and its partners. Thus, a BPEL4WS process provides one or more WSDL services. The BPEL4WS process is defined only in an abstract manner, allowing only references to service portTypes in the partnerLink (Andrews et al, 2004). Each partner is characterized by a partner link and a role name. In summary, the main idea of business process is to create an organizer that points to each service endpoint that will be actually executed.

2.1 Characteristics

The distinction between roles and partners in a business process is an important characteristic of BPEL4WS. This allows more simple and intuitive integration between enterprises. Another important characteristic of BPEL4WS is the fault handlers. Faults handlers have the ability to catch errors in BPEL4WS. Another characteristic from BPEL4WS is message correlation that allows processes to participate in stateful conversations. It can be used to match returning or known customers to long-running business process. Furthermore, correlation mechanisms allow interaction between a service instance and a partner. BPEL4WS addresses correlations scenarios by providing a declarative mechanism to specify correlated groups of operations within a service instance (Andrews et al, 2004).

In a BPEL4WS process we define the interactions between these activities that compose the service. Thus, there are some types of interaction like sequence, flow, switch, pick, moreover, each one can be combined.

2.2 Implementation

We developed a prototype using BPEL4WS. We created our composition based on our model defined in the introduction. Our composed service has three services: bookAirplane, bookHotel and rentCar. In our example, we have merged our activities execution. We put the sequence (each service is executed in a sequence way) with the flow (the services are executed in parallel). We chose this approach, because two activities could be executed in parallel (S1, S2), but the last one needs a dependency response from one of these two, so it must be executed in sequence (S3 depends on S2). We have defined four partners: the client, the airplane company, the hotel and the car rental

company. Thus, airplane company (S1) and the hotel (S2) could be executed in parallel because they have the same input types given by the client. However, the rentcar (S3) service needs one of the hotels output called "area" as mentioned earlier. In

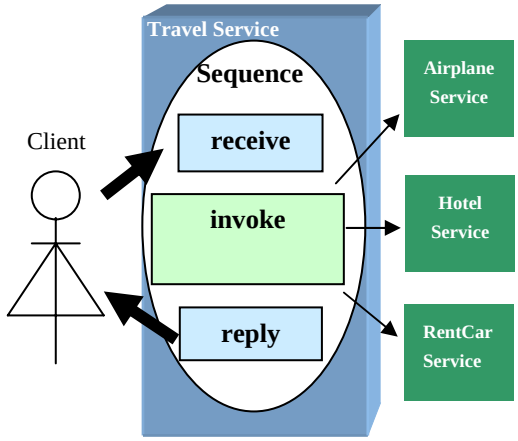

Figure 2: Internal view of Travel Service.

BPEL4WS we define a service, such as *Travel* by describing which others services it contains.

Figure 2, adapted from (Khalaf, 2004), shows the relation between the Travel service and the others that compose it.

After constructing the composition, we need to deploy our travel service; making it available for execution. At this moment, the deployment engine will require the WSDL files that were related on partner's links. As we have an interaction with each service developed, we must have a WSDL for each one. We have to mention in each WSDL the grounding tag in order to actually find the service. Additionally, we invoke the composition using an API created by IBM called BPWS4J1.1 (BPWS4J, 2004). Using this API to execute our composite service, we call a broker, "axisengine", and we use the endpoint given by the Travel deployment to do the connection between the client and services' providers. Using the endpoint, the broker can find the service, and then it can pass the first parameters that are sent by the client.

3 COMPOSING USING OWL-S

The process of composing services using a semantic web language like OWL-S increases the automatic discovery and composition process. In fact, OWL-S is based on ontology and OWL. This means that OWL-S is also based and constructed using resources and hierarchical concepts. With such a language, software agents can find services based on their computer-interpretable description.

The main motivating task for OWL-S was the ability to automatically discover web services. Other motivating tasks are automatic invocation of a service, with which a software agent can interpret markup to understand what input is necessary for the service call, what information will be returned and how to execute the service.

Additionally, the composed web service is actually an abstract service. In fact, the composition file has only the service calls. In OWL-S each service that is part of composition has the same structure of the composed one.

3.1 Characteristics

OWL-S is composed of three other structures called: service Profile, service Model and service Grounding, used to describe different aspects of the service, see figure 3 (OWL-S, 2004).

The service Profile is responsible for presenting the service to other services or agents that want to use it. It describes the service in order to facilitate the search process, specifying what organization provides the service and what functions the service provides. The service Model describes the service with regards to its inputs, outputs, effects and preconditions parameters. Furthermore, the process model is the core of OWL-S architecture, it defines how the process will be executed. Services can be composed using a combination of atomic, simple or composite services. This implies that a composition can have services that are themselves composed. Additionally, in the service model we can say how the services will be executed: in a sequence manner (*sequence*) or in a parallel manner (*split/split+join*) or some other way (OWL-S, 2004).

The service grounding is responsible for giving the endpoint of a service. A service grounding can be thought of a mapping between an abstract and a concrete specification (OWL-S, 2004). It is also in the grounding that we put the reference to each WSDL document.

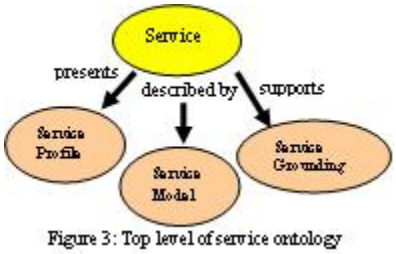

Figure 3: Top level of service ontology

3.2 Implementation

In our implementation using OWL-S composition, we defined the Travel service as being

composed of three atomic services called Airplane, Hotel and RentCar services. We have also determined that the first two services will be executed in parallel using a "split+join", but the last one should be executed on a *sequence* form.

We must define the OWL file for each atomic service. Furthermore, in these files we must put the grounding reference positioning exactly where the service is running. The Travel.owl file is only an abstract service where we define the input/output parameters and also which service will effectively do the tasks. Figure 4 shows the internal view of Travel service.

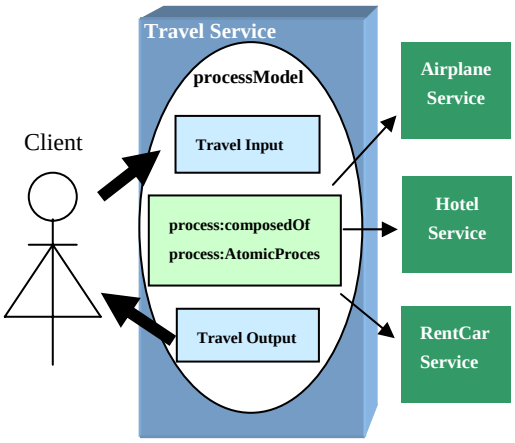

Figure 4: Internal view of Travel Service.

Furthermore, we can pass parameters between the services that compose the abstract service, for example, the "area" parameter is passed from the hotel service to the RentCar service.

After creating the OWL-S file containing the three services above, we need to execute the travel service, sending it the parameters: *date_arrival*, *date_departure*, *destination_city* and *country*. As a result we will obtain the flights, hotels and car rental information. In order to execute the travel service, we have used OWL-S API (Mindswap, 2004). For a client side, we defined an endpoint called *Travel* as the name of our service. Continue the execution, we invoke the Travel service and the OWL-S works on executing the others services that belongs to this composition.

4 COMPARISON BETWEEN BPEL4WS AND OWL-S

Nowadays the ongoing approaches for composing web services are: BPEL4WS and OWL-S. There are some differences between them concerning composition. However, many concepts that are used in these two implementations are similar. Here, we compare them based on the way used to compose web services. To create a composite service we must first create the services that will actually execute or do the tasks. The same services of the composition can be used from BPEL4WS or OWL-S without any changes. In fact, the manner how the service was actually created does not matter. After that, we can create an abstract service that will coordinate these services. However the process is executed, with both approaches we have an API that invokes the abstract service and the other invocations will be done by each one of the technologies used. We have used for executing web services composition an API but in fact, any user can create her/him own manipulation form.

In order to make a distinction between what is or is not a service, in BPEL4WS we use the notion of activities to represent a service while implementing. In BPEL4WS the activity that actually calls a service is *invoke*. For instance, the activity *invoke* has the partner link, actually the service's point, as mentioned in the second section of this paper. However, in BPEL4WS *invoke*, *receive*, *reply* are also considered as activities. Conversely with OWL-S, a service could be seen as an AtomicProcess, because it is where we put the service's resource invocation. As a result, we consider that a service in BPEL4WS is a set of activities including *invoke*. On the other hand, in OWL-S, we consider that a service is represented by atomicProcess or derivations. In the section above we will highlight some features that are different in both approaches.

4.1 Semantic features

The semantic feature is characterized as giving the resource meaning interpreted by a hierarchical structure. Moreover, using semantic description, we can allow automatic discovery of services by agents. Many features from the service are described in OWL-S using the service profile (as described earlier), containing information as functional description, enterprise contacts, category that the service belongs to. Using this feature or more, using a language that has this characteristic we can avoid problems like similar service descriptions that use an ambiguous word, like *bank*, standing for its economical meaning, or *bank*, sand accumulated on a river (Claro, 2004). In other words, related to semantics features, the OWL-S approach treats and allows automatic web services discovery. Conversely, BPEL4WS using BPWS4J does not allow that. Indeed, there are some works increasing the functionalities of BPEL4WS semantic features, as seen in (Mandell, 2003) that enable automatic

semantic discovery automatic. However, it is clear that this is an add-on to BPEL4WS specification.

4.2 Correctness

In order to increase the discovery process of BPEL4WS, augmenting services probability to be adequate to a specific task, we have proposed to improve the quality model proposed by many authors (Sreenath, 2004) (Zeng, 2004). Correctness is quality criterion that measures how adequate the service is to execute a specific task. We can treat that using a probability that measures how many times an *s* service was invoked and was appropriate to be executed. Indeed, we measure if the service executed correctly the task that was determined for it. In fact, this criterion is necessary only when using BPEL4WS for composing web services, because it does not treat semantics concepts. As a result we measure the success of the service's execution for a limited time. Our criterion can be defined as:

$$\sum_{s=1}^{n}\sum_{t=1}^{m} x_{st}\, f_{st} \geq l$$

where x_{ij} is 0 or 1 if service *s* executes task *t.* The measure of how adequate a service is for a task is given by f_{st} which means the integral of the success frequency. And *l* is the minimal boundary of adequate measure, normally given by a system administrator.

4.3 Fault Handler

Fault handler is a BPEL4WS characteristic that is very important concerning developing compositions. It means the ability to catch errors that have occurred for any reason when executing the process or when services are being invoked. The handling of some situations usually affects a set of activities that are associated with each other. In BPEL this is done by enclosing them in a *scope* structure activity. A scope provides the context for the activities nested within it and is where fault handlers are defined (Khalaf, 2004). A handler must contain a *reply* activity to notify a partner if an error has occurred. As explained above, this feature is not yet implemented in OWL-S.

4.4 Compensation

The compensation mechanism is present in BPEL4WS composition and can undo some committed task when we need to cancel some procedures. In fact, some activities that have been completed might later be undone because they belong to a long transaction. This can be easily compared to the *rollback* function in a transactional database environment. It is implemented as an activity that we have to put into our block where we probably want to assure that if something occurs, the others will be undone. Indeed, compensation features work also with the notion of *scope*. We can define a scope of activities and then we say that the compensation will work in this scope. It is important that the scopes that might be undone have a name because that is how the compensate activity identifies them (Khalaf, 2004) (Khalaf & Nagy, 2004). When a compensation activity is reached, it runs the compensation handlers on a specified scope.

4.5 IOPE

The IOPE (Input, Output, Preconditions and Effects) is one important feature present in OWL-S as process' parameters. The input and output specifies the data transformation produced by a process. The input specifies the information that the process requires for its execution. For atomic process this information must come from the client. Concerning composed services, some inputs come directly from the client but others come from previous services execution. If a process has a precondition, it cannot be performed successfully unless the precondition is true. Additionally, the performance of a process may result in changes of the world's state (effects). And the acquisition of information by the agent performing it (output). Thus, effects describe conditions in the world while output describes information (OWL-S 1.1beta, 2004). In BPEL4WS we do not have the concept of preconditions and effects, we only work using inputs and outputs.

4.6 Basic structures

Structured activities (BPEL4WS), also called control constructs (OWL-S) are the mechanisms used to order the service calls and thus their executions. Using these structures, it is possible, for instance, to determine if a service will be executed in sequence or in parallel. Some of these structures are similar in OWL-S and in BPEL4WS, for instance, *Sequence* which represents the fact that the services will be performed sequentially in the same order they are listed. The *While* (BPEL4WS) and *Repeat-While* (OWL-S) have similar functionalities, both test the condition before executing the loop. OWL-S offers another similar structure *Repeat-Until* which executes the loop and then tests the condition. In spite of parallel executions, OWL-S offers *Split+Join* and BPEL4WS treats it using *Flow*, but both provide concurrency and synchronization.

Another structure that we can compare is *Choice* (OWL-S) and *Switch* (BPEL4WS) which each call a structure from a given set of possibilities.

5 CONCLUSION

In this paper we have exposed the two ongoing approaches used to compose web services, also we have explained their features and compared their main characteristics. We have created an example that we developed using BPEL4WS and OWL-S allowing the comparison between these two languages. We have highlighted some important points from each approach giving a user the notion of what and how he/she can use and which features are present or not. Although they are evolving technologies, the BPEL4WS has some features not envisioned by OWL-S Coalition and vice-versa. This is because, even though, both languages have the same goal concerning composition, they have different goals about the manner to compose. In our opinion, OWL-S is more worried about automatic service discovery for composition whereas BPEL4WS is more concerned about composition process and how the composition is actually done. Additionally, we have proposed some new features to compensate the missing one in each language, such as our correctness characteristic.

As future works, we have planned to further develop and increase the quality model. Another work envisioned is to measure the execution time of each approach.

ACKNOWLEDGEMENT

Daniela Barreiro Claro is supported by a research scholarship given by the Région des Pays de la Loire (2003-2006).

REFERENCES

Andrews, T.,Curbera, F., Dholakia, H. et al. 2003. Specification: BPEL4WS Version 1.1. *Retrieved November 26, 2004, from http://www-128.ibm.com/developerworks/webservices/library/ws-bpel/.*

Berargi, D., Calvanese, D., De Giacomo, G. & Mecella, M.,2003. Reasoning about actions for e-Services. *In ICAPS Workshop on Planning for Web Services, Trento, Italy.*

BPWS4J, 2002. BPWS4J API. *Retrieved November 26, 2004, from http://www.alphaworks.ibm.com/tech/bpws4j.*

Claro, D.B., Albers, P., Hao, J-K. 2004. Web Services Composition using Reactive Planning. *In ICKEDS'04 International Conference on Knowledge Engineering and Decision Support, Porto, Portugal.*

Curbera, F., Khalaf, R., Mukhi, N., Tai,S. & Weerawarana, S. 2003. The next step in Web Services. *In Communication of the ACM, October, Vol.46, Nº10.*

Khalaf, R., 2004. Business Process with BPEL4WS: Learning BPEL4WS, Part2. *Retrieved October 27, 2004, from http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol2/*

Khalaf, R.; Nagy, W.A. 2004. Correlation, Fault Handler and Compensation: Business Process with BPEL4WS, Learning BPEL4WS Part 6. *Retrieved October 27, 2004, from http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol6*

Mandel, D.J., McIlraith, S.A.2003. Adapting BPEL4WS for the Semantic Web Bottom-up Approach to Web Service Interoperation. *In proceedings of the Second International Semantic Web Conference(ISWC 2003), Sanibel Island, Florida.*

McIlraith, S., Son; T.C., Zeng, H. 2001. Semantic Web Services. *In IEEE Intelligent Systems, Stanford University* .

Mindswap G., 2004. Maryland Information and Network dynamics lab semantic web agents projects. *Retrieved October 28, 2004, from http://www.mindswap.org/2004/owl-s/api/index.shtml*

OWL-S, 2004. OWL-S: Semantic Markup for Web Services. OWL Services Coalition. *Retrieved November 08, 2004, from http://www.daml.org/services/owl-s/1.0/.*

OWL-S 1.1beta: Semantic Markup for Web Services, *Retrieved November 08, 2004, from http://www.daml.org/services/owl-s/1.1B/owl-s/owl-s.html.*

Peltz, C., 2003. Web Services Orchestration and Choreography. *Computer Magazine, Octobre 2003, Volume 36, Number 10, pages 46-52.*

Schroeder, R., Claro, D. B., Matarazzo, C., 2004. ERP Systems Integration using Web Services. *UDESC – State University of Santa Catarina. Joinville, Brazil(in portuguese).*

Sreenath, R.M., Singh, M.P., 2004. Agent-based service selection. *In Web Semantics: Science, Service and Agents on the World Wide Web, pp.261-279.*

van der Aalst, W.M.P.,2003. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems, 18(1):72-76.*

Zeng, L.,Benatallah,B., Dumas,M., Kalagnanam,J., Sheng, Q.Z., 2003. Quality Driven Web Services Composition, *In proceedings of the Twelfth International Conference of WWW, May 20-24, Budapest, Hungary.*