SEMANTIC WEB SERVICE EXECUTION FOR WSMO BASED CHOREOGRAPHIES

James Scicluna
Axel Polleres
Digital Enterprise Research Institute (DERI Innsbruck)
Institute of Computer Science, University of Innsbruck,
Technikerstrasse 21a, 6020 Innsbruck, Austria
E-mail: {james.scicluna, axel.polleres}@deri.org

KEYWORDS
Semantic Web, Web Services, WSMO, Web Service Execution, Choreography, Orchestration, Workflow.

ABSTRACT

The Semantic Web is slowly gathering more importance as both academic and industrial organizations are realizing the potential benefit that might be obtained from it. This is especially true in the areas of tourism in which Semantic Web Services can provide a drastically new way on how to find and book related services such as hotel, flights and taxi transfers. However, many aspects of Semantic Web Services are still under development. This short paper presents issues related to choreography and orchestration representation in the Web Service Modelling Ontology (WSMO) and also how such ideas can be applied to an e-tourism use case.

INTRODUCTION

The Web Service Modelling Language (WSMO) is based on the Web Service Modelling Framework (Fensel and Bussler 2002) which specifies the key elements for describing Semantic Web Services. Such elements include Ontologies, Goals, Web Services and Mediators. Ontologies are the key to describe all of these elements through concepts, relations, functions, instances and axioms. Goals specify what the user wants from a particular web service. Web Service descriptions specify what a service can provide in terms of a capability and any number of interfaces. Mediators link heterogenous elements between each other in order to enable heterogenous components to interoperate.

Details of these elements are beyond the scope of this document and we refer the reader to (Feier 2005). Rather, here we want to focus on the Interface of a web service description which enables to model a choreography interface and orchestration for a web service. We will then provide an example of to define such an interface based on an etourism use case (Lopez 2004).

Our use cas is called *Stream Flows! System* (SFS). It is a frequent flyer program designed to be accessible from the internet by its customers. Using this system, customers can buy packages or contract new ones in exchange of points rather than using a credit card. Packages may include flights, hotel booking, car rentals and others. The system is responsible for the continuous maintenance of its packages, that is, once a service is no longer available, the package has to be updated or removed from the system.

A Web Service description in WSMO is defined in terms of non functional properties, a single capability and any number of interfaces. *Non Functional Properties* define information regarding the usage of the service without affecting its functional aspect. In addition to the core non functional properties for the WSMO elements, a web service defines further service specific issues such as quality of service, financial range, robustness and others. A *capability* defines what the service can provide in terms of preconditions, assumptions, postconditions and effects. A *precondition* defines what the web service expects for enabling it to provide its service. An *assumption* describes the expectation of the service on the state of the world when starting an execution of the service. A *postcondition* describes the state of the information space that must be reached by executing the service and an *effect* describes the state of the world that must be reached after executing the service. All these features are defined using logical expressions based on the Web Service Modelling Langue (WSML) (De Bruijn 2005) .

The *interface* of a service defines two pieces of information, namely, a choreography and an orchestration. WSMO allows to define multiple interfaces for a service and hence there could be more than one way in which a user (or an agent) may interact with it. The *choreography* exposes how a client (whether a user or an agent) should interact with the service. The *orchestration* describes how the service uses other web services in order to achieve the required functionality.

CHOREOGRAPHY

WSMO defines a choreography as how services interact with a client which may either be a user or a service. This adheres to the W3C definition. WSMO is concerned with the formal modeling of a choreography which would also enable to perform reasoning over the description (such as to identify whether two web services can talk to each other).

Abstract State Machines as a Formal Model

Abstract State Machines (formerly knows as Evolving Algebras) have been used to describe and validate a wide variety of computing systems. In WSMO, they provide the basis for describing both the choreography and orchestration specifications. In a nutshell, an abstract state machine is defined in terms of a state signature, a state and a number of guarded transitions. A state signature defines the invariant elements of the state description. A state is described by a set of instance elements and guarded transitions are rules that express the changes of states. For further details about how these aspects are defined in a choreography, we refer the reader to (Roman et. al., 2005). Rather, we will provide some examples on how these aspects can be applied In our use case.

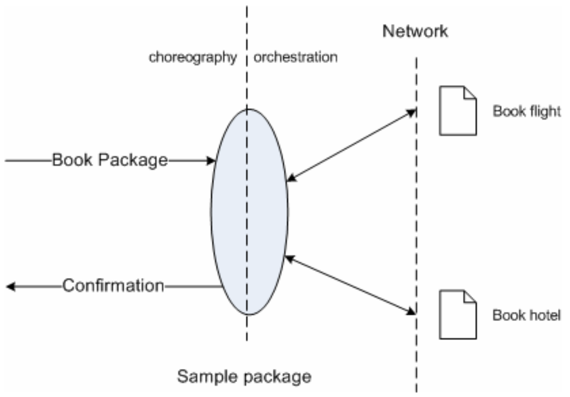

Figure 1: A simple package illustrating the WSMO Choreography and Orchestration Perspectives

Figure 1 depicts a very simple package which defines in its choreography a booking of a flight and then that of a hotel. We will now provide a simple way on how to specify this in terms of an abstract state machine in the choreography of some interface defined in the service. For simplicity reasons and to keep this document concise, we will just describe the choreography definition in the web service. Hence, we will assume that the needed ontologies (for flight, hotel etc) already exist in some ontology. The classes needed are as follows:

- Package Booking (receive a request to book the package)
- Package Confirmation (send a confirmation that package has been booked)
- Flight
- Hotel

In order to identify which concept is an input or an output, we require to extend the concepts needed in order to support the attribute mode whose value determines whether a concept is an input or an output of a choreography. This extended ontology imports all the ontologies which define the classes to be extended in order to be used by the Web Service. Below is an abstract of the extended concepts to be used by the web service.

```
bookPackage subClassOf sfs#packageBooking
  nonFunctionalProperties
    mode hasValue in
    grounding hasValue sfsWsdl#bookPackage
  endNonFunctionalProperties

packageConfirmed subClassOf sfs#confirmedPackage
  nonFunctionalProperties
    mode hasValue out
    grounding hasValue sfsWsdl#bookPackage
  endNonFunctionalProperties
```

A state is described by the attribute values of the instances in the choreography vocabulary (that is, the ontology defining the extended classes needed). The transition rules in the choreography express the current state in the condition and also the updates required in order to move to a next state. In both cases, this is defined in terms of attribute values of the instances used by the choreography as shown below.

```
choreography simplePackageChoreography

  guardedTransitions simplePackageTransitions

    if(bookPackageInstance memberOf
       sfs#packageBooking[
         book hasValue true
       ])
    then
      (confirmedPackageInstance memberOf
        sfs#confirmedPackage[
          confirm hasValue true
        ]).
```

Notice that the concepts bookPackage and packageConfirmed are grounded to a single WSDL Operation. The mode attribute thus identifies whether the concept is an input or an output of the operation. In our very simple example, the choreography requires only one transition since as soon as the booking request is received, what's left is to send back a confirmation. The booking request defines a boolean value which determines whether the booking should be performed or not, if this is so, the confirmed package instance will be sent to the user and its *confirm* value set to true.

ORCHESTRATION

WSMO Orchestration is based on the same Abstract State Machine model. However, the guarded transitions are extended to support links to mediators when performing updates. These mediators can either link to a goal (wgMediator) or to another Web Service (wwMediator). The orchestration thus models what other Web Services are to be used in order to achieve the requirements of the Web Service defining the orchestration.

```
if(flightInstance memberOf sfs#flight[
     booked hasValue true
   ]
then
  _"http://example.org/bookFlightMediator"

if((flightInstance memberOf sfs#flight[
     booked hasValue true
   )]
   and
   (hotelInstance memberOf sfs#hotel[
     reserved hasValue true
   ])
then
  _"http://example.org/bookHotelMediator"
```

Notice that the second rule specifies a condition on both the flight and hotel instances in order to ensure that it is not infinitely fired.

The roles of the mediators linked by the orchestration differ depending on the type of mediator used. A *wgMediator* must:

1. Mediate data between the ontologies used by the source service and the target goal.
2. Link the capability of the source web service to the one in the target goal such that it reduces the information space needed by the orchestration at a given state

The purposes of a *wwMediator* are as follows:

1. The normal data mediation which is needed to resolve heterogeneities between the ontologies used by the source and targets
2. Link the orchestration to the target choreography of the web service such as to enable invocation of the target web service
3. Perform Process Mediation in case the orchestration and the target choreography don't match at the protocol level

LANGUAGE FOR A WORKFLOW SPECIFICATION

The language for choreography and orchestration in WSMO formally models such aspects. However, such a model is not executable in terms of concrete Web Service calls (such as via SOAP). Therefore, a more expressive workflow language is needed in order to enable actual execution of an orchestration. This language must enable to perform execution at the semantic level in order to enable reasoning about parameters (such as simple subsumption) and also reasoning for service composition purposes. An example of such a workflow language is outlined below.

```
inputs
  bookingRequest impliesType sfs#packageBooking

outputs
  confirmation impliesType sfs#confirmedPackage

initial_state
  bookingRequest = EMPTY

flow
  get(bookingRequest)

  if(bookingRequest.book == true)
    then
      confirmation.confirm hasValue true
    else
      confirmation.confirm hasValue false

  put(confirmation)
```

First, the inputs and outputs are declared. All of the elements needed are instances of ontologies. Where necessary, variables are used but this is not the case. The initial state is declared by setting the input to get to EMPTY. The <*flow*> element defines the behavioral flow of the service. To receive data, the method *get* is used. This method required that execution stops until the required data is obtained. Once the booking request is received, it is checked if the *book* attribute is set to true and the *confirm* attribute of the confirmation element is set appropriately. Once the package has been booked, the confirmation element is sent to the client using the *put* statement.

Another approach in defining "semantic" workflows is that of providing semantic information for existent technologies and specifications. An example for such an approach is the one taken in (Mandell and McIlraith) providing a description of how BPEL4WS can be extended in order to provide the semantic information needed for automatic discovery and composition.

FUTURE WORK

Our main future efforts will involve in a better specification for Choreography and Orchestration in WSMO. Furthermore, a more clear and executable workflow language will be specified and mapped to the ASM model defined in WSMO. Also, the mediators which are used to link from an orchestration to a goal (*wgMediator*) or another Web Service (*wwMediator*) need to be further defined in order to enable interoperability between their source and target components.

CONCLUSION

We hereby provided a very short description of the current work which addresses the choreography and orchestration specifications in WSMO based on Abstract State Machines and how we intend to provide a workflow language in order to enable execution of the web service. Also, a simple example was provided to show how these aspects can be applied in an eTourism use case. Our main focus was how WSMO can model such scenarios and enable also actual execution of the underlying service.

ACKNOWLEDGEMENT

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the Vienna city government under the CoOperate programme and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects [EQUATION] and TSC.

REFERENCES

De Bruijn J. 2005. "The WSML Family of Representation Languages" *WSML Working Draft*, March 2005.

Feier Cristina. 2005 "WSMO Primer". *WSMO Working Draft*, March 2005..

Fensel D.; Bussler C. 2002 "The Web Service Modeling Framework". January 2002.

Gurevich Y. 1993 "Evolving Algebras: Lipari Guide, 1993".

Lopez O. 2004 "Requirement Profile 1 & Knowledge Objects". *Infrawebs Working Draft* December 2004

Mandell D. and McIlraith S. "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation"

Roman D.; Feier C.; and Scicluna J. "Ontology-based Choreography and Orchestration of WSMO Services" *WSMO Working Draft.* March 2005