Orchestration of semantic web services in IRS-III

Orchestration of Semantic Web Services in IRS-III*

Roberto Confalonieri[1,2], John Domingue[1] and Enrico Motta[1]

[1]Knowledge Media Institute, The Open University, Milton Keynes, UK
{r.c.confalonieri, j.b.domingue, e.motta}@open.ac.uk
[2]Department of Computer Science, Università di Bologna, Bologna, Italy
confalon@cs.unibo.it

Abstract. In this paper we describe our orchestration model for IRS-III. IRS-III is a framework and platform for developing WSMO based semantic web services. Orchestration specifies how a complex web service calls subordinate web services. Our orchestration model is state-based: control and data flow are defined by and in states respectively; web services and goals are modeled as activities and their execution triggers state changes. The model is illustrated with a simple example.

1 Introduction

Web services are the new way of interacting with and using the web; users are expected to seek for appropriate web services that help them to achieve their goals. This process of manual search may be automated if web services are augmented with semantic descriptions and infrastructures for supporting them are developed [3]; IRS-III [2] is a framework and implemented infrastructure which supports the creation of semantic web services according to the WSMO ontology [6].

Web service interfaces play an important role in the composition and execution of web services. *Choreography* describes the interaction process between web services. *Orchestration* represents the workflow steps of a composite web service fulfilling its capability as its decomposition. Whilst choreography for IRS-III has been almost defined [1], orchestration has not.

In this paper we present an ontology for modeling the orchestration of a composite web service in IRS-III and an interpreter that executes it. The model is in OCML [4].

The paper is organized as follows: Section 2 briefly overviews orchestration. Section 3 describes our model and implementation issues through a simple example. Section 4 contains conclusions and describes future work.

2 Orchestration

Orchestration is a process-centric view of the interactions between the composite web service and the sub services that it relies upon. It includes complex process se-

*This work is supported by the DIP (Data, Information and Process Integration with Semantic Web Services) and AKT (Advanced Knowledge Technologies) projects. DIP (FP6 - 507483) is an Integrated Project funded under the European Union's IST programme. AKT is an Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

mantics (loops, conditions, fork...) and/or workflow steps that are outsourced to external services. In a nutshell orchestration describes how the service works from the provider's perspective, i.e. how a service makes use of other services represented by activities in order to achieve its capability [5, 7]. This can be done in two ways: the specific sub services are fixed in the activities at design-time (static composition); activities are dynamically bounded by declaring them as goal descriptions on the basis of any goal-service discovery mechanism (automatic composition). An activity is expected to use a particular interface for the sub services it binds at run-time; the interface contains details about services it uses to solve the goal currently being processed. Heterogeneity mismatches between the used interface and the one needed have to be resolved through mediation.

3 State-based orchestration in IRS-III

We choose a state machine representation for the orchestration of semantic web services in IRS-III. In our approach states define the control flow, transitions represent *activities* and activity execution triggers state change. An activity can be a web service (simple or composite) or a goal as IRS-III supports capability-driven invocation of web services.

According to the WSMO standard model, a web service interface description is composed of choreography and orchestration; an orchestration has a *problem solving pattern* (fig.1).

```
currency-converter-orchestration (orchestration)
  has-problem-solving-pattern :value currency-converter-psp

currency-converter-psp (problem-solving-pattern)
  has-start :value currency-converter-psp-start
  has-end :value currency-converter-psp-end
```

Fig. 1 Orchestration definition of the currency converter composite web service

In our *orchestration ontology* a problem solving pattern consists of a set of classes modeling states and activities, with a *start-state* and *end-state* classes connected by control construct state classes. Start-state and end-state represent respectively the entry and exit data flow points for the data of the composite web service being orchestrated (fig.2).

```
currency-converter-psp-start (start-state)
  has-input-role :value has_source_currency
                 :value has_target_currency
                 :value has_amount
  has_source_currency :value(wsmo-orchestration-role-value
                 currency-converter-web-service 'has_source_currency)
  has_target_currency :value (wsmo-orchestration-role-value
                 currency-converter-web-service 'has_target_currency)
  has_amount :value (wsmo-orchestration-role-value
                        currency-converter-web-service 'has_amount)
  has-later-state :value exchange-rate-sequence-state
```

```
currency-converter-psp-end (end-state)
  has-output-role :value has-currency-conversion
  has-currency-conversion :value (wsmo-orchestration-role-value
                                  multiply-activity 'multiply-output)
```

Fig. 2 Start-state and end-state definitions of the currency converter orchestration; input and output-role value slots reflect input and output-role of the currency converter web service

Control states are wrappers for activities (fig.3) and they represent the control flow as an execution path in the model. We've defined three control construct states:

- *sequence-state*: the sequence construct is the elementary unit of orchestration as web services and goals are represented by activity in sequence-states;

```
exchange-rate-sequence-state (sequence-state)
  has-activity :value exchange-rate-activity
  has-later-state :value multiply-sequence-state

multiply-sequence-state (sequence-state)
  has-activity :value multiply-activity
  has-later-state :value currency-converter-psp-end
```

Fig. 3 Sequence state definitions of the currency converter orchestration: the currency-converter web service is composed by two activities to be executed in sequence; the interpreter selects the next state through the `has-later-slot`

- *conditional-state*: the conditional construct checks if a certain condition is true or false and selects the appropriate execution branch; the condition is an OCML relation, namely a *kappa-expression*, which ranges on either outputs of previous activities or inputs of the composite web service orchestrated; loops as *do-while* and *repeat-until* can be represented in terms of the conditional state;
- *fork+join-state*: the fork+join construct consists of concurrent execution of a bag of activities whose results have to be joined.

The data flow, i.e. how the data are used before and after the execution of activities, is defined in the activity classes by means of `has-input-role` and `has-output-role` value slots (fig.4).

```
multiply-activity (activity)
  activity-type :value multiply-goal
  activity-ontology :value wsmo-multiply
  has-input-role :value multiply-input1
                 :value multiply-input2
  has-output-role :value multiply-output
  multiply-input1 :value (wsmo-orchestration-role-value
                          currency-converter-psp-start 'has_amount)
  multiply-input2 :value (wsmo-orchestration-role-value
                          exchange-rate-activity 'has_exchange_rate)
  multiply-output :type number
```

Fig. 4 Multiply activity definition as a goal: the multiply-output slot is a relation for the output data flow in the model; exchange-rate activity definition is similar

We have adopted a consumer-pull convention for data. The data are stored locally and the consumer later retrieves the data needed through a *wsmo-orchestration-role-value* OCML function (fig.4). Web services may have heterogeneous input and output. For their composition either to the binding a mediation mechanism for the data is required. The aforementioned function therefore plays a double role in the model:

- *data binding*: the first argument of the function specifies which activity or state class the current activity relies upon,
- *data mapping*: the second argument of the function specifies the output-role needed and the evaluation of the function assigns a value to the input-role of the current activity.

A composite web service invocation results in instances of the appropriate orchestration classes being created at runtime. The OCML model is interpreted by an orchestration engine written in Common Lisp. The interpreter in a straightforward way reads state by state, instantiates the state it is currently processing and invokes the appropriate handler. At instantiation time the input-roles needed for the current activity are retrieved; each handler is responsible for executing, monitoring the activity and storing its result properly before selecting the next state. The "storage" is represented by an OCML relation of the form `(output-role instance-class-name output-role-value)` asserted as a fact to the orchestration ontology at runtime (fig.4).

4 Conclusions and Future Work

The orchestration engine is stateless as the web services in IRS-III; our model does not satisfy the orchestration requirements outlined in [5] as our intention was first to be able to run a composite web service in IRS-III. Our plan is to investigate more on semantic aspects related on the automatic composition following an approach based on parametric design; to add *fork* and *join* control construct, to make the engine statefull and to integrate orchestration with the choreography model presented in [1].

References

1. Domingue, J., and Galizia, S. Towards a Choreography in IRS-III. Proc. of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30, 2004.
2. Domingue, J., et al. IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. Proc. of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30, 2004.
3. McIlraith, S., Son, T. C., and Zeng, H. Semantic Web Services. IEEE Intelligent Systems, Mar/Apr. 2001, pp.46-53.
4. Motta, E. An Overview of the OCML Modelling Language. KEML98.
5. Peltz, C. Web Service Orchestration and Choreography. IEEE Journal, Computer 36 (10). 46-52 October, 2003.
6. Web Service Modeling Ontology – Standard, available at: http://www.wsmo.org/2004/d2/
7. Orchestration in WSMO, available at: http://www.wsmo.org/temp/d15/v0.1/20040529/