Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 181 (2007) 19–33
www.elsevier.com/locate/entcs

JOLIE: a Java Orchestration Language Interpreter Engine

Fabrizio Montesi[1], Claudio Guidi[2], Roberto Lucchi[2]
Gianluigi Zavattaro[2]

[1] *Corso di Scienze dell'Informazione di Cesena, University of Bologna, Italy*
[2] *Department of Computer Science, University of Bologna, Italy*

Abstract

Service oriented computing is an emerging paradigm for programming distributed applications based on services. Services are simple software elements that supply their functionalities by exhibiting their interfaces and that can be invoked by exploiting simple communication primitives. The emerging mechanism exploited in service oriented computing for composing services –in order to provide more complex functionalities– is by means of *orchestrators*. An orchestrator is able to invoke and coordinate other services by exploiting typical workflow patterns such as parallel composition, sequencing and choices. Examples of orchestration languages are XLANG [5] and WS-BPEL [7]. In this paper we present JOLIE, an interpreter and engine for orchestration programs. The main novelties of JOLIE are that it provides an easy to use development environment (because it supports a more programmer friendly C/Java-like syntax instead of an XML-based syntax) and it is based on a solid mathematical underlying model (developed in previous works of the authors [2,3,4]).

Keywords: SOA, coordination, orchestration, Java, service, engine

1 Introduction

Service oriented computing is an emerging paradigm for programming distributed applications based on services. Services are simple software elements that supply their functionalities by exhibiting their interfaces and that can be invoked by exploiting simple communication primitives, the so-called One-Way and Request-Response ones. Services can be composed each other in order to design more complex services by exploiting *orchestrators*. The orchestrators, indeed, are able to invoke and coordinate other services by exploiting typical workflow patterns such as parallel composition, sequencing and choices. Furthermore, composition can be also achieved

⋆ Research partially funded by EU Integrated Project Sensoria, contract n. 016004.

[1] Email: famontesi@gmail.com
[2] Email: cguidi@cs.unibo.it, lucchi@cs.unibo.it, zavattar@cs.unibo.it

doi:10.1016/j.entcs.2007.01.051

by following a different approach, that is choreography, that allows to design a distributed system in a top view manner [2,8]. The most credited technology that deals with service oriented computing is *Web Services* which aims at guaranteeing interoperability among different platforms and whose specifications are defined by means of the XML language. One of the most important specification is WSDL [10] that defines a language for designing a Web Service interface. An interface allows to access service functionalities by means of operations. An operation represents the basic interaction modality of a service and it can be a One-Way operation, where an invoking message is sent to the service, or a Request-Response one, where an invoking message is sent to the service assuming that a response message will be subsequently sent back from it. Web Services can be composed following both orchestration and choreography approaches. As far as orchestration is concerned here we cite WS-BPEL [7], as far as choreography is concerned we cite WS-CDL [9].

In our previous works [2,3,4] we have analyzed orchestration and choreography as synergic approaches for distributed system design by following a formal approach. Our formal investigation aims at supplying a precise formal framework on which we can develop designing tools for service oriented computing systems where orchestration and choreography languages play a fundamental role. In particular, we have formalized both choreography and orchestration languages by means of two process calculi and we have presented a formal notion of conformance between them based on bisimulation. As it emerges by those works the orchestration represents w.r.t. choreography a refinement step towards the implementation of service oriented applications. Informally, if on the one hand choreography does not produce executable systems, on the other hand the orchestration makes it possible to program each service involved in the application. For the sake of brevity, we do not report the formal definition of the syntax and semantics of our orchestration language (for a closer look to the language we remind to the previous works). We simply report a small example in order to give the flavour of the kind of calculus we have developed. Assume a buyer service requests for the price of a particular kind of good to a seller service by sending a message on a Request-Response operation. Then, it invokes a purchase order by sending a message on a One-Way operation. We can model such a service dialog as follows:

$$Buyer ::= [good := apple; \overline{price}@S(good, price); ...; \overline{apple}@S(250), \mathcal{S}_B]_B$$

$$Seller ::= [price(good, eur, good = apple?eur := 100) \mid apple(n); ..., \mathcal{S}_S]_S$$

The buyer is a service located at site B where the *good* variable is initialized to the value *apple*. $\overline{price}@S(good, price)$ means that the buyer invokes the Request-Response operation *price* at the service located at site S sending the variable *good* and storing the response into the variable *price*. Then, the buyer performs some internal computation (that we do not specify for the sake of brevity). Finally, it performs $\overline{apple}@S(250)$ that represents the invocation of the One-Way operation *apple* to the service located at S in order to initiate a purchase order of 250 apples. The ; is a sequential composition operator which means that all the statements must be executed one after the previous one has completed. The seller is a service located