Agent-based Mediation in Semantic Web Service Framework

Renato de Freitas Bulcão Neto[1], Yathiraj Bhat Udupi[2], and Steve Battle[3]

[1] University of São Paulo, São Carlos SP 13560-970, Brazil,
`rbulcao@icmc.usp.br`
[2] North Carolina State University, Raleigh NC 27695, USA,
`ybudupi@csc.ncsu.edu`
[3] Hewlett Packard Laboratories, Bristol BS34 8QZ, UK
`steve.battle@hp.com`

Abstract. In a semantic web service scenario clients and services should interoperate by allowing a service to be delivered via different protocols and data formats. This paper describes a novel solution to protocol and data mediation through a goal-driven, agent-mediated interaction with web services described by OWL-S ontologies. Our contributions include: (i) an OWL-S compiler which mediates between two OWL-S service description interfaces and outputs a script containing a set of executable Nuin plans, and (ii) an agent-based mediator built upon Nuin framework that executes these plans in a event-driven fashion.

1 Introduction

There is a need for richer knowledge-based product and service descriptions to enhance the existing business interactions over the Internet. Current approaches to web-service description (e.g. WSDL) are strongly tied to the message syntax and protocol. This paper describes how we enrich the current web-services model with semantic support for goal-driven, agent-mediated interaction with web-services described by OWL-S ontologies [1]. We present a case study about a software product marketplace for vendors who lack a comprehensive sales infrastructure. A service request made using SOAP-based interactions with the web service enables the client to place an order. In the existing application hard-coded java applets enable the user to communicate with the web-service. For a user with a browser it is simpler to have a web-friendly, resource-oriented interaction [2]. The information in the client request is encoded in the request URL query string. This is unlike posting an explicit request message. This poses a mediation problem, where we need to enable clients and services to interoperate by allowing a service to be delivered via different protocols and data formats. There are two perspectives on the mediation problem, first is protocol mediation: how do we describe one service in terms of another and ensure that it achieves the same goals. The second is data mediation: how do we achieve independence from the syntax of the specific messages allowing us to map from one message format to another.

Our approach is set out in the *web service modelling framework (WSMF)* [3] and it caters to the main objectives of WSMF. It supports rich, declarative service descriptions, which separates the design of the service functionality from its delivery and provides for a framework in which those descriptions are used. Mediation is achieved within an agent-based framework moving from the syntactic domain of messages into a representational framework based on semantic web technologies (RDF and OWL). This agent-based mediator assists the client in achieving specific goals, which seen as key to identifying the tasks and actions to be performed by the service.

Contribution. Our first contribution is the development of an OWL-S compiler which mediates between two different OWL-S service descriptions derived from the *requester* (the client), and the *service provider* interface and outputs an executable Nuin script [4]. Nuin is an agent-framework with emphasis on the building of Semantic Web agents. Our second contribution is the development of the agent-based mediator built upon the Nuin framework that executes the script generated from the compiler in an event-driven fashion. These approaches to solving the mediation problem help us overcome the main barriers to *e-business process automation*.

2 Agent-Based Mediation

This section describes agent-based mediation and event-driven plans, and is demonstrated by an example scenario.

2.1 The Agent Framework in the BDI Architecture

The agent framework which animates the WSMF can be described in terms of a *belief-desire-intention* (BDI) architecture [4]. Various elements of the conceptual architecture are mapped into agent beliefs, desires and intentions. *Beliefs* correspond to the background knowledge of the agent held in its knowledge base (updated with message content at run-time) and its accompanying ontology. *Desires* include information about the client goals, comprising of the information in the service request which is based on the OWL-S profile (including important service parameters). The *intent* of the user is conveyed to the agent through individual requests at the user interface. This way the agent translates the desires and intents of the user into tasks and actions at the provider interface.

The agent executes the various Nuin plans that perform the required protocol and data mediation. These plans coordinate activities across the various plug-in components that support communication with the client and the service provider. Figure 1(a) describes the agent architecture. The *web plug-in* of the agent mediator functions as an adapter between a web server and the agent, lifting HTTP requests into RDF and mapping responses back into HTML. The *service plug-in* of the agent acts as an adapter to an invocation client for the SOAP web services. A *lift module* provides an interpretation of the message content and a translation to or from a common representational form, RDF model, based on XML schema [5]. A request message gets dropped from RDF into XML and conversely responses are lifted from XML back into RDF.

2.2 Protocol Mediation by Process Planning with the OWL-S Compiler

Compilation is an off-line process that generates the Nuin plans required to mediate between the requester and the provider interfaces, and is performed by the OWL-S compiler. Both interfaces include OWL-S service descriptions including descriptions of *inputs*, *outputs*, *preconditions*, *unconditional effects*, *service parameters*, etc. The

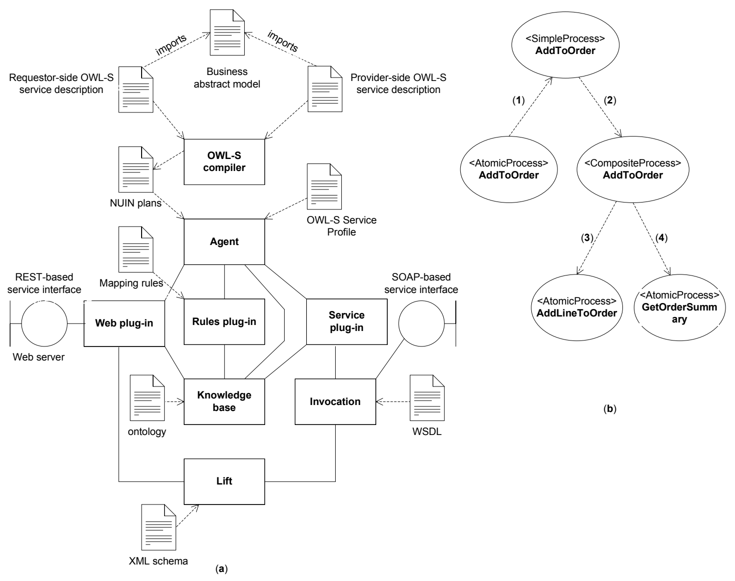
Business abstract model
imports
imports
Requestor-side OWL-S service description
Provider-side OWL-S service description
OWL-S compiler
NUIN plans
OWL-S Service Profile
Agent
Mapping rules
REST-based service interface
SOAP-based service interface
Web plug-in
Rules plug-in
Service plug-in
Web server
Knowledge base
Invocation
ontology
WSDL
Lift
XML schema
(a)
<SimpleProcess> AddToOrder
(1)
(2)
<AtomicProcess> AddToOrder
<CompositeProcess> AddToOrder
(3)
(4)
<AtomicProcess> AddLineToOrder
<AtomicProcess> GetOrderSummary
(b)

Fig. 1. (a) Agent mediation architecture. (b) An example of event-driven intent invocation.

service descriptions at both interfaces need not have a one-to-one mapping between them. Where the immediate effects of actions at the two interfaces do not correspond exactly, we define composite processes that have the required combined effect. Also, an *abstract business process model* represents the abstract view of the provider interface. This model is imported by the concrete processes of the two interfaces. The primitive parts of the abstract process are of type *OWL-S SimpleProcess* allowing us to describe a business process independently of its realization.

The OWL-S compiler reads the above descriptions and outputs a set of executable Nuin plans. The compiled output is modular in that each plan corresponds to an atomic, composite or simple process. Each atomic process corresponds to an invocation or receipt of a message (that may require a response). Each composite process corresponds to a breakdown of the plan into smaller tasks. Nuin supports backtracking enabling us to back out of a plan where the preconditions do not hold. Simple processes realized in different ways at the two interfaces, create the bridge necessary for protocol mediation. They are the point where the agent recognises the user intent and then forms its own intent to act.

2.3 Data Mediation using Mapping Rules

The agent-based mediator equipped with a *Rules plug-in* performs data mediation, which realizes the mapping between the incoming and outgoing message content and their common ontological conceptualizations. Mapping rules are applied to the content stored in the knowledge base representing previously received and lifted message content. The rules plug-in is based on the Jena rules engine and the mapping rules are expressed in the Jena rules language [6].

2.4 Event-driven Intent Invocation

The agent plans are designed to allow for event-driven triggering of plans. At the requester interface, the web plug-in extracts the query parameters and raises an event that signals the receipt of the message. The event triggers a plan corresponding to an atomic process which, in effect, recognises the event as a user action. In turn the atomic process plan signals the user action with another event. This event may trigger composite process plans that recognise more complex actions. At the point where the occurrence, or recognition, of a process on the user-side corresponds to an equivalent process available on the provider-side, the agent can form the intent to act. Once the invocation of a process against the provider has completed it it necessary to complete the user-side action by returning the appropriate HTTP response. This is viewed as a continuation of the action that raised the original event.

Figure 1(b) describes a scenario of the invocation of an atomic process at the requester-side interface being translated to a composite process with its component atomic processes in the provider-side interface. The user simply wants to add an item to his shopping basket; to *AddToOrder*. Step (1) is an event that alerts the agent to the user action to add an item to the order. It will include an input that identifies the required product. We see that the *AddToOrder* simple process is realized in different ways on the requester and provider sides. They are processes with equivalent effects. At step (2) the agent forms its own intent to *AddToOrder* at the provider interface. This happens to be a composite process plan, invoked from the simple process plan. This invokes the individual atomic process plans *addLineToOrder* and *getOrderSummary* in steps (3) and (4) respectively. Note that given the intention to *AddToOrder* in step (2) we drive the process in a top-down way. However, prior to recognising an intention we drive the process from the bottom-up.

3 Conclusion

This paper described an agent-based solution to protocol and data mediation following the major objectives set out by the WSMF: services, mediation, ontology, and goals. We have shown how OWL-S service descriptions may be used within an agent-based framework to support this ontology-based mediation.

References

[1] OWL-S Coalition. OWL-S 1.0 Release. At http://www.daml.org/services/owl-s/1.0/, 2003.
[2] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD dissertation, University of California, Irvine, USA, 2000.
[3] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. At http://informatik.uibk.ac.at/users/c70385/wese/, 2002.
[4] I. Dickinson and M. Wooldridge. Towards Practical Reasoning Agents for the Semantic Web. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 827–834, Melbourne, Australia, 2003.
[5] S. Battle. Round-tripping between XML and RDF. In *International Semantic Web Conference*, At http://iswc2004.semanticweb.org/posters/PID-BRRGVFRE-1090254811.pdf, 2004.
[6] HP Labs Semantic Web Team. Jena Semantic Web Framework. At http://jena.sourceforge.net/, 2003.