Agents for the Grid: A comparison with Web Services (part II: Service Discovery)

Arturo Avila-Rosas, Luc Moreau, Vijay Dialani, Simon Miles, Xiaojian Liu
Intelligence, Agents, Multimedia
Deptartment of Electronics and Computer Science
University of Southampton, United Kingdom
{ara01r, L.Moreau, vkd00r, sm, xl1}@ecs.soton.ac.uk

ABSTRACT

In order to build an open, large-scale and inter-operable multi-agent system in the context of Grid computing, we are looking at integrating agents technologies with Web Services. In this paper, we address this concern for SoFAR, the Southampton Framework for Agent Research. We focus on all technical aspects of creating, deploying, and publishing agents as Web Services. Not only have we been able to translate SoFAR ontologies and agent behavioural descriptions respectively into XML Schemas and WSDL, but also we have reexpressed in terms of XML Schema validation a pattern matching oriented query language used in discovery mechanism. Using this approach, an agent in the SoFAR framework can be deployed and advertised through a standard discovery mechanism such as UDDI.

1. INTRODUCTION

We are observing the convergence of three major technologies for distributed systems: Grid, Agents and Web Services. The *Grid problem* is defined as flexible, secure, coordinated resource sharing, among dynamic collections of individuals, institutions and resources [5]. Grid Computing and eBusiness share a large number of requirements, such as inter-operability, platform independence, dynamic discovery, etc. In the eBusiness community, Web Services have emerged as a set of open standards, defined by the World Wide Web consortium, and ubiquitously supported by IT suppliers and users. They rely on the syntactic framework XML, the transport layer SOAP [3], the XML-based language WSDL [2] to describe services, and the service directory UDDI [1].

The benefit of open standards has recently been acknowledged by the Grid Community, as illustrated by three projects embracing Web Services in various ways: Geodise (`www.geodise.org`) is a Grid project for engineering optimisation, which makes Grid services such as Condor available as Web Services. MyGrid (`www.mygrid.org.uk`) is a Grid middleware project in a biological setting, which adopts semantic Web techniques to describe services and their coordination through workflows. The Open Grid Service Architecture (OGSA) [4] extends Web Services with support for Grid Services lifetime management.

As far as large scale computing is concerned, the agent community is not at rest. The popularity of the AgentCities initiative (`www.agentcities.org`) promises us a world-wide multi-agent system infrastructure. Additionally, the notion of agent has of late become popular in the Grid community, as exemplified by several workshops and publications on the use of agents in the Grid [6, 8].

In the myGrid project, we are concerned with designing a future-proof middleware, and we are adopting agent-based computing as its underlying paradigm; we are also looking at its integration with Web Services technology in order to promote its open-ness. In a previous publication, we have shown how an agent system could integrate its transport layer with Web Services [6] and we have successfully implemented it in SoFAR, the Southampton Agent Framework for Agent Research [7].

While the transport layer essentially addresses syntactic issues, further semantical considerations are required to provide proper inter-operability. In this extended abstract, we show how agent behaviour can be described, advertised and discovered using Web Services technologies. WSDL is used to describe agent behaviour, which can then be advertised in UDDI registries as regular services. Additionally, we provide an extended UDDI registry, able to support advanced queries used for discovering in agent system.

The rest of this paper is as follows. In Section 2, we describe Web Services, their architecture, and three standards adopted. In Section 3, we present the schemas associated with SoFAR ontologies and explain our implementation of the agent service description. In Section 4, we motivate and introduce mechanisms to deploy and discover agent services through ontology-based query language. Finally, in Section 5 we summarize our discussion.

2. WEB SERVICES

A Web Service interacts with its environment through a collection of operations that are network-accessible through

standardized XML messaging. A Web Service is described by an XML-based service description that covers all the details necessary to interact with the service, including message formats, transport protocols and location.

For an application to take advantage of Web Services, three operations have to take place: publishing service descriptions, looking up service descriptions, and binding (or invoking services) using such service descriptions.

2.1 Web Service Architecture

Three fundamental layers are required to provide or use Web Service. First, Web Services must be network-accessible to be invoked, HTTP is the de-facto network protocol for Internet-available Web Services (SMTP and FTP can also be supported). Second, Web Services should use XML-based messaging for exchanging information, and SOAP is the chosen protocol. Finally, it is through a service description that all the specifications for invoking a Web Service are made available; WSDL is the de-facto standard for XML-based service description.

2.2 Web Service Description

The Web Services Description Language (WSDL) is an XML-based language to describe Web Services and how to access them. A Web Service is seen as a set of end points operating on messages containing either document-oriented or procedure-oriented data. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete set of operations are bundled into abstract endpoints (services). WSDL can be extended in such a way that different message formats and network protocols can be used to describe endpoints and their messages.

A WSDL description is typically composed of an implementation - independent part and an implementation-dependent part. The implementation-independent part of a service definition contains WSDL elements that can be reusable and referenced by multiple service implementations, such as: `<binding>`, `<portType>`, `<message>` and `<types>` elements. The implementation-dependent part contains WSDL elements that describe how a particular service is implemented: `<service>` and `<port>`. We will illustrate them in detail in Section 3.

The W3C XML Schema Definition Language (XSD), uses a uniform XML syntax for describing and constraining the content of XML documents. XML Schemas structure contains datatypes, elements and their content, and attributes and their values. XSD also supports derivation of document types (similar to sub-classing in object-oriented languages), and provides atomic data types (such as integers, floating point, dates) in addition to character data.

2.3 Web Service Publishing and Discovery

The Universal Description, Discovery, and Integration UDDI Project, provides a standardized method for publishing and discovering information about Web Services. Conceptually, three types of information about a service can be registered into a UDDI registry: White pages (basic contact information), Yellow pages (Web Service categorization), and Green pages (behaviours and supported functions of a Web Service in other words, service descriptions).

UDDI defines a complex data structure to store the latter information, including categorization data, such as references to service description documents, that can be used by richer search facilities.

3. MAPPING SOFAR TO WEB SERVICES

To illustrate our point on how Agent-Based Computing may be integrated with Web Services, we consider SoFAR, the Southampton Agent Framework for Agent Research. SoFAR provides an abstract communication model based on an agent communication language (ACL). In particular, SoFAR communication model can be instantiated over the XML protocol SOAP [6]. This communication mechanism constitutes two of the three basic layers (Network, XML-Based messaging, and Service description) required to provide or use Web Service. In this section, we will continue with the construction of this inter-operable base stack, but we now focus on the service description layer.

We define an *Agent Service* as a service that conforms to the set of conventions for Web Services, that is, agent ontologies are defined using XML Schema components, and the agent behaviour is described as a WSDL interface. The benefit is that by publishing agents as service descriptions, other Web Services may make an effective binding and dynamic invocation of the agent seen as a Web Service, regardless of whether it is an agent-based computing functionality behind.

3.1 SoFAR Overview

SoFAR considers a small set of performatives which allows any agent to support common primitive interactions such as querying, notifying, requesting, subscribing, and advertising. Let us note that SoFAR supports not only asynchronous performatives, but also synchronous performatives, which are useful for querying databases and knowledge bases. The following interface taken from [7] describes all the performatives supported by SoFAR agents.

```
interface Agent {
   boolean query_if(Predicate p, Envelope e);
   Predicate[] query_ref(Predicate p, Envelope e);
   void inform(Predicate p, Envelope e);
   void uninform(Predicate p, Envelope e);
   Contract register(Action a, Envelope e);
   void unregister(Contract c, Envelope e);
   Contract subscribe(Predicate p, Envelope e);
   void unsubscribe(Contract c, Envelope e);
   void request(Action e, Envelope e);
}
```

In SoFAR, performatives arguments must be defined in an ontology. Ontologies are organised along a hierarchy based on single inheritance. Terms of ontologies are defined by the unique parent they extend and a (possible empty) set of typed fields they contain. For instance, a Person can be defined as an entity composed of three fields.

```
<term name="Person"    extends="Term">
  <field type="String" name="title"/>
  <field type="String" name="forename"/>
  <field type="String" name="surname"/>
</term>
```

SoFAR has its own mechanism to advertise and discover agents called the RegistryAgent, which is a matchmaker that matches agents' need for services against agents' ability to provide such services. An agent has to advertise its capabilities by sending the RegistryAgent a message that contains the performatives to handle.

SoFAR ontologies specify not only the types of data structures, but also they provide the basis for a query language over sets of such data structures. The query language extends pattern-matching with subsumption and constraints over variables.

3.2 SoFAR Ontologies Schemas

In this Section, we show how to express SoFAR ontology concepts as XML Schemas. The features of XML Schemas we employ are atomic datatypes, simple and complex type definitions, subtyping by extension and restriction, and abstract type definitions.

In Figure 1, we model two concepts `Term` and `Person`. The concept `Term` is root of the hierarchy (any concept or relation in SoFAR ontology is an extension of `Term`) and is also abstract (for which there cannot be any instance).

```
<xsd:complexType name="Term" abstract="true"/>

<xsd:complexType name="Person">
 <xsd:complexContent>
  <xsd:extension base="Term">
   <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="forename" type="xsd:string"/>
    <xsd:element name="surname" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Figure 1: Ontology as XML Schema

3.3 Agent Service Description

The next layer in our abstract architecture is the agent service description layer, for which we use WSDL. Let us consider an agent called `PersonAgent` which possesses the ability to support queries about persons through a `query_ref` performative; such a performative returns all the elements that match the argument of the query.

The first element in a WSDL document, after the root element (`<definitions>`), is the `<types>` element. A complete WSDL description of `PersonAgent` can be found in Appendix. Although the `<types>` element encloses data type definitions used to describe the messages exchanged, we prefer defining all terms involved in `PersonAgent` description in a separated document. So, our WSDL document contains an `<import>` element that contains a reference to a schemas file where all the types used are defined.

```
<import namespace= "http://sofar.ecs.soton.ac.uk/schemas"
location= "http://sofar.ecs.soton.ac.uk/Person.xsd"/>
```

The performative `query_ref` is considered as an operation with request-response messages; it means that the endpoint receives a message, and sends a correlated message as a response. So, there are two `<message>` elements for such a performative. Each `<message>` contains zero or more `<part>` elements. A `<part>` corresponds to an argument or a returned value in the respective method call. Each `<part>` must have the same name and data type as the argument or return value it represents. Input and output parameters of the method with signature:

```
Predicate[] query_ref(Person person,
                      Envelope envelope);
```

have been modelled as follows:

```
<message name="query_refRequest">
   <part name="person" type="Person"/>
   <part name="envelope" type="Envelope"/>
</message>

<message name="query_refResponse">
   <part name="result" type="ArrayOfPredicates"/>
</message>
```

In this definition, an `Envelope` denotes a data-structure that contains meta-information about the message (similarly to a message header).

An `<operation>` element in WSDL is the equivalent of a method signature. An operation specifies which `<message>` element is the input and which `<message>` is the output. The collection of all operations exposed by the service are grouped into a `<portType>` element.

```
<portType name="PersonAgentPortType">
   <operation name="query_refOperation">
     <input message="tns:query_refRequest"/>
     <output message="tns:query_refResponse"/>
   </operation>
</portType>
```

So far, we have defined operations and messages in an abstract way, the next step is to provide a reference to the external SoFAR framework in order to define how a SOAP client will reach the implementation of our Agent Services. In WSDL, such a reference is referred to as a binding between abstract definitions and their implementation. An element `<binding>` defines the message format and protocol details for operations and messages supported by a particular portType.

```
<binding name="PersonAgentSoapBinding"
         type="tns:PersonAgentPortType">
 <soap:binding style="document"
       transport="http://.../soap/http/"/>
  <operation name="query_refOperation">
    <soap:operation soapAction="" />
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></input>
     </output>
   </operation>
</binding>
```

Finally, we identify the Agent Service logically, and attach the port with the physical location of the agent, here an http server running at port 50000 on the machine `sofar.ecs.soton.ac.uk`.

```
<service name="PersonAgentService">
 <port binding="tns:PersonAgentSoapBinding"
          name="PersonAgentPort">
   <soap:address
       location="http://sofar.ecs.soton.ac.uk:50000
                   /soap/servlet/rpcrouter"/>
 </port>
</service>
```

4. PUBLISHING AND DISCOVERING AGENT SERVICES

We are now capable of publishing our Agent Service Description into a UDDI registry. But just a minimal part of our WSDL description can be published into the UDDI structure. UDDI structure provides a taxonomy to categorize the service and only keeps references to our WSDL description. We use an extension of UDDI registry where we be able to store additional information (meta-data) about agents and an ontology-based pattern-matching in order to accommodate the kind of searching that is required to locate an agent service according to the performative it supports. In this extended abstract, we focus on the ontology-based query language.

In Section 3.1, we have introduced the matchmaker as a mechanism to advertise and discover agents in SoFAR. Now, we bring those ideas to Web Services, by using pattern-matching among type definition schemas (ontological terms). Let us consider an instance of `Person` defined in the previous Section, which is modelled in figure 2.

```
Person("Dr",?String,?String)
```

If we regard the latter term as a query to an agent, it has the following meaning: tell me the set of Persons with the title of `Dr`. In order to decide if `PersonAgent` is capable to handle queries of this type, such a term must match the `Person` schema that it is associated with the `PersonAgent` description. This operation is similar to *schema validation*, that is defined as the process of parsing an XML document in order to determine whether it is well-formed and follows the constraints of some schema. Deciding if a pattern matches

```
<Person>
  <title>Dr</title>
  <forename>
     <String variable='true'/>
  </forename>
  <surname>
     <String variable='true'/>
  </surname>
</Person>
```

Figure 2: Ontology-based Query as XML Schema

a term can be implemented by checking that the pattern is validated by the XML Schema specifying that term. Ongoing work focuses on extending this matching algorithm to support constraint resolution.

We are proposing an extension of UDDI which contains WSDL descriptions of all agents that have been registered. It enables us discovery of agents supporting a specific capability using the ontology-based pattern matching described.

5. DISCUSSION AND CONCLUSION

Our paper has shown that the concept of agents can be closely aligned with that of a Web Service, in that an agent can be described as a Web Service and discovered using an standard mechanism as UDDI.

We have demonstrated that XSD is expressive enough to describe SoFAR ontologies; and through validation process, ontology-based pattern-matching can be implemented in order to adapt UDDI to the sort of searching that is required to find an agent service.

Using WSDL gives the agent the ability to describe and to advertise its capabilities. Our agent Web Service descriptions and its terms, that are expressed using ontologies as a semantic enhancement to WSDL and UDDI, enable dynamic discovery and invocation of services by software through common terminology and shared meaning. This is a vital property in an open systems such as the Grid.

In the future, we plan to look at the orchestration of agent services, based on workflow languages such as WSFL and XLANG. This will allow us to study the suitability of WSDL for describing semantically-composable agent services. Such an orchestration activity can make heavy usage of ontology-based metadata about the quality of service offered by agents, for which we consider formalisms such as OIL, DAML+OIL, and RDF.

6. ACKNOWLEDGEMENTS

This research is funded in part by EPSRC myGrid project (reference GR/R67743/01) and EPSRC comb-e-chem (reference GR/R67729/01).

The first author acknowledges the funding of the Mexican Petroleum Institute (IMP).

7. REFERENCES

[1] Uddi standards. http://www.uddi.org.

[2] W3c wsdl spec. http://www.w3c.org/TR/wsdl.

[3] Xml protocol working group. http://www.w3c.org/2000/xp/Group/.

[4] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid — An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Argonne National Laboratory, 2002.

[5] Ian Foster, Carl Kesselman, and Steve Tuecke. The anatomy of the grid. enabling scalable virtual organizations. *International Jounral of Supercomputer Applications*, 2001.

[6] Luc Moreau. Agents for the Grid: A Comparison for Web Services (Part 1: the transport layer). In *IEEE International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May 2002.

[7] Luc Moreau, Nick Gibbins, David DeRoure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley, and Mark Weal. SoFAR with DIM Agents: An Agent Framework for Distributed Information Management. In *The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, pages 369–388, Manchester, UK, April 2000.

[8] Omer F. Rana and Luc Moreau. Issues in Building Agent based Computational Grids. In *Third Workshop of the UK Special Interest Group on Multi-Agent Systems (UKMAS'2000)*, Oxford, UK, December 2000.

APPENDIX

```
<?xml version="1.0" encoding="UTF-8"?>
 <definitions xmlns="http://schemas.xmlsoap.org/wsdl" xmlns:xsd1="http://sofar.ecs.soton.ac.uk/schemas"
              xmlns:tns="http://sofar.ecs.soton.ac.uk/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              name="PersonAgent" targetNamespace="http://sofar.ecs.soton.ac.uk/PersonAgent.wsdl">

 <import namespace="http://sofar.ecs.soton.ac.uk/schemas"
    location="http://sofar.ecs.soton.ac.uk/Person.xsd"/>

 <message name="query_refRequest">
   <part name="person" type="Person"/>
   <part name="envelope" type="Envelope"/>
 </message>

 <message name="query_refResult">
  <part name="result" type="ArrayOfPredicates"/>
 </message>

 <portType name="PersonAgentPortType">
    <operation name="query_refOperation">
      <input message="tns:query_refRequest"/>
      <output message="tns:query_refResult"/>
   </operation>
 </portType>

 <binding name="PersonAgentSoapBinding" type="PersonAgentPortType">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http/"/>
      <operation name="query_refOperation">
         <soap:operation soapAction=""/>
            <input>
               <soap:body use="literal"/>
            </input>
            <output>
               <soap:body use="literal"/>
            </output>
     </operation>
 </binding>

<service name="PersonAgentService">
 <port binding="tns:PersonAgentSoapBinding" name="PersonAgentPort">
   <soap:address location="http://sofar.ecs.soton.ac.uk:50000/soap/servlet/rpcrouter"/>
 </port>
</service>

</definitions>
```