Semantic Web Service Interaction Protocols: An Ontological Approach*

Ronald Ashri[1], Grit Denker[2], Darren Marvin[3], Mike Surridge[3], and Terry Payne[1]

[1] University of Southampton, Southampton, UK
{ra, trp}@ecs.soton.ac.uk
[2] SRI International, Menlo Park, California, USA
denker@csl.sri.com
[3] IT Innovation, Southampton, UK
{djm, ms}@it-innovation.soton.ac.uk

Abstract. A central requirement for achieving the vision of run-time discovery and dynamic composition of services is the provision of appropriate descriptions of the operation of a service, that is, how the service interacts with agents or other services. In this paper, we use experience gained through the development of real-life Grid applications to produce a set of requirements for such descriptions and then attempt to match those requirements against the offerings of existing work, such as OWL-S [1] and IRS-II [2]. Based on this analysis we identify which requirements are not addressed by current research and, in response, produce a model for describing the *interaction protocol* of a service in response. The main contributions of this model are the ability to describe the interactions of multiple parties with respect to a single service, distinguish between interactions initiated by the service itself and interactions that are initiated by clients or other cooperating services, and capture within the description service state changes relevant to interacting parties that are either a result of internal service events or interactions. The aim of the model is not to replace existing work, since it only focuses on the description of the interaction protocol of a service, but to inform the further development of such work.

1 Introduction

Recent years have seen a redoubling of effort, both within industry and academia, to provide appropriate solutions to the problem of run-time discovery and composition of services [3,4]. Such a capability is central to the needs of a wide range of application domains and its importance is underlined by a significant amount of industrial backing in terms of willingness to agree on underlying standards and to provide tools in support of application development. For example, within the context of e-business it is required to support

* Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168 to SRI. Supported by Engineering and Physical Sciences Research Council Semantic Firewall project (ref. GR/S45744/01)

S.A. McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp. 304–319, 2004.

the dynamic provision of services selected at run-time from an ever-changing pool of competing service providers based on user preferences, or the creation of coalitions of service providers working towards a common goal [5,6]. Within the context of e-science it can enable researchers to better take advantage of Grid infrastructure to create complex solutions involving the use of computational resources and highly specialised services across institutional boundaries [7]. The majority of such efforts focus on the use of Web Services (WS) [8] as the enabling infrastructure, which provides the necessary standardization of message transportation [9], low-level service description [10], service discovery (e.g. UDDI [11]) and service composition (e.g. BPEL4WS [12]). However, WS technologies, *on their own*, are judged as insufficient to provide effective solutions to the problem of *dynamic* discovery and composition, due to the lack of appropriate semantic descriptions of the services involved and insufficient support for flexible run-time dynamic binding of services [13,3].

In response to these limitations, a number of solutions have been proposed, which either attempt to provide an overarching model, which is then linked to the underlying WS infastructure (top-down approach - e.g. OWL-S [1], IRS-II [2]), or attempt to embed appropriate solutions *within* the WS infrastructure (bottom-up approach - e.g. [13], [14]). The latter approach, although it often supports rapid progress, has the obvious limitation of being restricted in the type of solutions it produces by the WS infrastructure it attempts to integrate with. Furthermore, as the infrastructure itself is evolving the solutions will have to adapt to newer versions of standards. The former approach allows for the development of more generalised solutions that can find application outside a WS context and as the WS standards evolve. It supports a clear distinction between an abstract model of service description, which can then be used to support service discovery and composition, and the implementation of that model within the context of a specific set of enabling technologies, such as WS. Nevertheless, even these overarching models still have some way to go before they adequately address the range of issues related to service description.

In this paper we identify some of the shortcomings of current approaches as they relate to the description of the *operational process* of a service, or in other words, its *interaction protocol*, based on a set of requirements derived through an analysis of real-life Grid application scenarios implemented using the GRIA infrastructure [15]. We then propose a model for describing the interaction protocol, which furthers the state of the art through the following specific contributions.

- The ability to describe *multi-party interactions* with respect to a single service, with the different parties distinguished by the *roles* they occupy within the service operation. This is a result of the the need to be able to describe, with respect to a single service, either strict requirements or simply the capability it has to cooperate with other services and *what form that cooperation can take* during operation.
- The ability to describe interactions initiated by the described service as well as interactions that are initiated by other parties. This provides a more

flexible model than what is currently available and enables the support of more sophisticated services, which can, for example, notify clients about the state of progress or call upon other services to perform specific actions.

– The ability to describe the state changes a service goes through that are relevant to the interacting parties, caused either by events internal to the service or as a result of interactions with outside parties. This enables a client to distinguish between the results of actions it initiates, and those that depend on the service itself or other third parties. Furthermore, it enables other parties, such as security devices integrated in the underlying infrastructure [16], to monitor the progress of interaction with a service, so as to ensure that the service is not misused.

Our aim while developing this work has not been to provide yet another alternative for service description, but rather to address specific needs that arose as a result of the demands of real-life grid application that were not addressed by existing work. As such, we view our work as complementary to existing approaches and focused on the description of the interaction protocol of the service, rather than the grounding of such descriptions to WS technologies or generalised service descriptions, since existing methods can easily be used in coordination with our work.

The rest of the paper is structured as follows. Section 2 provides a motivating example that illustrates our requirements for service description. Section 3 discusses existing technologies and their limitations with respect to those requirements. Section 4 presents our service interaction protocol model, using an ontological approach, and illustrates its use with the help of the example presented in Section 3. Section 5 offers two examples of the use of the model. The first discusses how the service description can be used by a semantic web security device to regulate access to services, while the second discusses how the service description can be used to infer what dependencies a service has with respect to other services. Finally, Section 6 indicates the future directions of this work.

2 Motivating Example

Our motivating example is based on a straightforward usage scenario for Grid applications that is supported by the GRIA (Grid Resources for Industrial Applications) infrastructure [15]. It involves a *client* that submits a computation job to a *job service*, where the computation job specifies a particular application to execute, such as a renderer. Furthermore, due to the typically significant amount of data over which the computation will be run the use of a *data staging service* is also required. The data staging service manages the provision of input data to the job service and provides the results to the client once the computation is completed. Below, we discuss in more detail the specifics of the implementation of this scenario within the GRIA infrastructure.

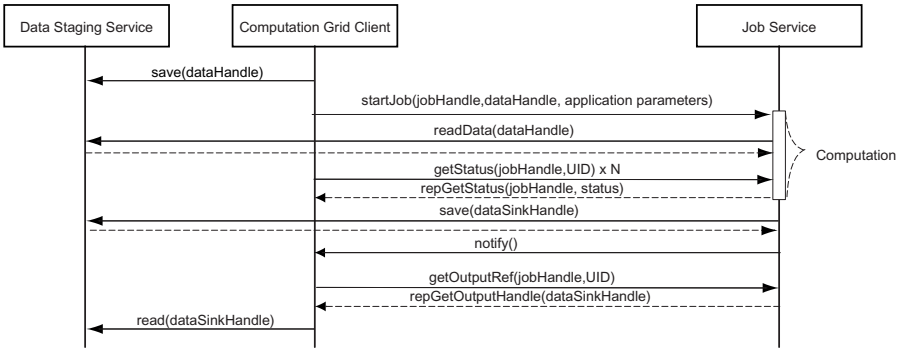

Fig. 1. An Example Grid Workflow

2.1 Grid Application Support Through GRIA

The GRIA framework is a Grid infrastructure developed using just the basic web service specifications, as part of the EC IST GRIA project [17].[1] It provides the necessary infrastructure for exposing computationally intensive applications across the Grid, with ancillary facilities for data staging and quality of service negotiation. A Grid service within GRIA can be considered as a *contextualised* web service, which exposes its functionality through a well-defined interface. It is contextualised since the interactions with the web service are based on a well-defined process, which demands that a context is maintained throughout the lifetime of the interactions. It is this interaction protocol that we aim to make explicit by providing an appropriate ontological model that will allow us to describe it.

In GRIA, a scenario as the one described above actually makes use of a number of other services and systems, both external and internal. Internal systems and services include resource schedulers, accounting systems and databases, while external services include data staging services, certificate authorities, and so forth. GRIA also provides features such as negotiation over the quality of service, long-term accounts with service providers. We do not discuss this issues in detail here, but the interested reader is referred to [15], in which a more complete description of the GRIA system is available.

Here we focus on just the interactions between the client, the job service and the data staging service, which are illustrated in Figure 1. The scenario is described below.

[1] A comparable system to GRIA is the Globus Toolkit 3 reference implementation of OGSI [18]. However, the OGSI model extends web service standards and explicitly introduces Grid-related concepts through those extensions. This is something that GRIA has avoided by adopting only web service standards. WSRF (Web Services Resource Framework - `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`) is closer in nature to GRIA, but has only been released very recently and has no implementations available.

1. The Grid client saves its data to the data staging service supplying the `dataHandle` it obtained, along with a `jobHandle`, from a previous resource allocation stage.
2. The Grid client submits a job request to the job service including the `jobHandle`, the `dataHandle` for the source data and any additional application parameters.
3. During the lifetime of the computation the Grid client can obtain status information as to the progress of the computation.
4. At the end of the computation the job service saves the output data to a data staging service, obtaining a handle to data in response.
5. The job service notifies the client that the computation is complete.[2]
6. The Grid client retrieves a handle to the output data from the job service.
7. The Grid client reads the output data from the data staging service using the output data handle.

2.2 Job Service Description Requirements

Given this relatively straightforward example of service usage within a Grid environment we now turn our attention to identifying what are the necessary requirements for adequately describing the job service in a manner that will make clear the allowed sequence of interactions, the different parties involved and who is the initiator of which interactions. The specific requirements are discussed below.

Interaction Sequence. The most basic feature that a service description should cater for is a description of the appropriate sequence of interactions between the client and the service provider and who is the initiator of those interactions. For example, it should be clear that the client needs to request the `dataSinkHandle` from the job service, once the client has been notified that the computation is complete.

Multiple Parties and Roles. The description needs to clearly identify the various parties involved in the interactions, and who is the initiator of the interaction. Note, that while the client's main goal is to make use of a job service it cannot achieve that without also interacting with a data staging service. The use of the job service is *dependent* on the data staging service. Therefore, the description of the job service *must* make clear this dependence and describe the exact interaction the client and the job service will have with the data staging service. However, since these descriptions cannot identify specific instances of a client or a data staging service, as these are bound at runtime, it should identify the *roles* the various parties occupy. A role can be considered as identifying a well defined behaviour in terms of interaction with the service, such as *client* or *data staging service*.

Service State Changes. It is also desirable to be able to describe state changes that take place due to events that are not related to the interactions

[2] GRIA does not at present support notification because firewall policies at client sites are likely to deny them.

between services, but may impact on what interactions are allowed following that state change. For example, once the computation has completed the client can no longer request the status of the service.

3 Existing Technology and Current Limitations

Given the basic set of requirements identified above, we discuss in this next section the extent to which they are met by existing models for service description or related technologies.

3.1 OWL-S

The OWL-S [1] model attempts to provide a comprehensive approach to service description. The model has found considerable uptake by the Semantic Web Services community and as such has set a certain bar against which any other proposals are typically compared. It is used within the context of prototype applications (e.g. [19,20]) and extensions to the model relating to security have been suggested (e.g. [21]), as well as extensions in order to provide a more principled, theoretically grounded, view of the model [22].

Underpinning OWL-S is a separation of concerns along the lines of a *service profile* that describes what the service does, a *service process model* that describes how the services operates, and a *service grounding* that links the service model to the enabling Web Service infrastructure.

We focus on the descriptive capabilities of the OWL-S *process model*, since that is our central concern. The process model allows the specification of the operation of a service based on the combination of processes using control constructs such as sequence, split, split+join, and so forth. Necessary input, outputs, preconditions and effects can be described for each process, as well as conditional outputs and effects.

The process model, although well suited to deal with a large range of situations, does not adequately fulfill the requirements that we identified in Section 2. The main reason is that there are not constructs to indicate the *direction* of the interaction, i.e. whether the client or the service initiate it, or the fact that other parties may be involved in the interaction. As we discussed, these parties may be *necessary* as was the case in our example or may simply represent parties that could be involved to provide additional support (e.g. the use of a provenance service within the context of a grid application). Finally, there is also no representation of the state of a service.

The lack of such constructs may simply be a result of the fact that the underlying bias of the OWL-S model is to accommodate interaction with *stateless* web services with no individual thread of control, and as such any control of dataflow between processes should be handled entirely by the client.

3.2 IRS-II

The *Internet Reasoning Service-II*(IRS-II) [2] takes a broader view of the entire task of service discovery and composition, based on the UPML framework [23]. The framework distinguishes between *domain models* (describing the domain of application), *task models* (generic descriptions of the task, including inputs, outputs and preconditions), *problem solving methods* (abstract descriptions of the reasoning process), and *bridges* (mappings between different model components of an application). Individual services are described based on their inputs and outputs and the preconditions to their invocation. Services can be coupled by defining problem solving methods that describe their combinations. However, there is no support for complex interactions between services nor a flexible way to describe the operation of single services along the lines of the OWL-S process model. As such, although IRS-II makes a significant contribution towards providing a comprehensive approach to service discovery and composition based on an explicit theoretical model, it does not allow for a great deal of flexibility in describing complex individual services.

3.3 Web Services Choreography

Web Services Choreography [24] represents a relatively recent effort within the W3C, with the goal of providing a language that describes interactions between services. The underlying motivation is to provide a language that better reflects long-term, complex interactions between *stateful* services. The WS-Choreography model supports the definitions of roles, the definitions of message exchange based on either a request or a request+response protocol and the exchange of state information. The language makes use of pi-calculus as a grounding to formal, conceptual model [25].

In essence, a choreography document describes the interactions between a number of participants, with each participant occupying specific roles. The choreography is divided into work units, with work units containing activities. A number of process flow constructors enable the combination of activities. Finally, choreographies provide for progressively more specific definitions starting from an abstract definition that does not include an WSDL or SOAP constructs to portable and finally concrete implementations.

WS-Choreography addresses several of the issues that we aim to address, such as describing more complex interactions between stateful services through an abstract model that can then be grounded to specific WS technologies such as BPEL4WS. However, WS-Choreography does not address the need for describing *individual* services so as to then support the automatic composition of more complex interaction protocols, a goal that we aim to support. Furthermore, although the need for integration with Semantic Web technologies is stressed by the WS-Choreography working group it is currently not addressed. As such, our work can be seen as complementary to this effort, since we focus on descriptions of individual services and their needs in terms of interaction with a client or other supporting services. Such description could then be used to facilitate the composition of choreographies as envisaged by WS-Choreography.

3.4 BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) [12] defines a notation for specifying business processes composed of Web services. It supports the description of both executable process models (i.e. the actual behaviour of the web service) and business protocols (i.e. just the mutually visible message exchange). The language includes constructs for declaring process flow including iteration and choice as well as provides facilities for failure handling and compensation. A single process imports and exports functionality through WSDL interfaces. In this way, the process acts as a Web service itself and can invoke other Web services as part of the process. BPEL4WS also supports the declaration of partners and their roles, while state that should be public within a business process can be exposed through message correlation constructs.

In general, BPEL4WS is a flexible language, with facilities for the addition of new constructs and attributes to the existing BPEL4WS standard without rendering existing engines incapable of utilising a BPEL document. However, BPEL4WS remains a low-level language that is heavily dependent on WSDL constructs, since it is aimed to tightly integrate within the general framework of Web Service standards. Furthermore, although semantic annotations could be added this would again be restricted to annotations of WSDL constructs, rather than more generalised notions. Our aim, as discussed in Section 1 is to take a top-down approach so that we are not restricted by low-level standards such as BPEL4WS and to provide a model that describes the interactions with respect to a single service.

Nevertheless, we realise that in order for our model to find practical implementation within a wider setting that includes service composition dependent on description of individual services, it should be amenable to a grounding to a BPEL4WS description.

4 Service Description Through Interaction Protocols

As pointed out in Section 2, the desirable features for describing interaction protocols of services include multi-party interaction, explicit representation of who initiates the interaction, and representation of the relevant state changes of the service. Parts of these requirements can be found in some existing approaches as discussed in Section 3. For example, WS-Choreography defines roles and request and response protocols. Nevertheless, there exists no approach that captures all required features that resulted from our requirement analysis for real-world grid services. Therefore, we propose a new model for service description, basing it on an initial proposal for the definition of Interaction Protocols in DAML [26] and extending it with additional concepts such as events to reflect internal state changes of services, roles, and different types of messages to express direction of message flow (incoming vs outgoing).

The approach we take is describing the interactions with a service as an interaction protocol, using a *service-centric* point of view. The protocol defines

what are the appropriate messages that can be exchanged at any given moment from any of the parties involved in the interaction, and it is *service-centric* in the sense that we focus on just those interactions that include the service we wish to describe. For example, the service description of the job service need not include the first and the last interaction in Figure 1 between the client and the data staging service. That interaction should be described by the service description of the data staging service, which the client should have access to as well (this point is discussed further in Section 5). The reason for such a service-centric view is simply because a service can only define what interaction *it* can participate in and should avoid making assumptions about the nature of interactions that it is not party to.

4.1 Interaction Protocol Model

The proposed interaction protocol model is described in this section. The model is defined as an OWL ontology (see `www.csl.sri.com/users/denker/sfw/wf/ip.owl`) and Figure 2 illustrates the main elements. In the figure, squares depict classes, arrows depict object properties, and the dashed boxes depict enumeration classes of individuals.

The appropriate interactions with a service are described through an *interaction protocol*. This protocol corresponds to a set of *states*, where each state has a set of *options*. Options describe either the messages each party taking part in the interaction can perform while the interaction protocol is at that state or what events can take place at that state. Once one of the possible options in a state is taken our model describes which state follows that option, which in turn describes the next set of available options and so forth. Each option has a label to describe that option as either an *event* or a *message*.[3] In addition to labels attached to options, we further refine the model to include three types of options: *request*, *request+response*, and *internal*. A request option corresponds to the exchange of one message, whereas a request+response option includes two sequential message exchanges, the first one being the request message and the second one being the response message. Naturally, request and request+response options have messages associated with them. An internal option corresponds to an internal state change and always has an event associated with it.

Each message has a *name*, a *type*, a *role*, and *content*. The name of the message serves to identify the message within the domain of discourse. For example, if the service use an agent communication language, along the lines of the FIPA ACL [27], the name would correspond to the performative in question. If the service message exchanges correspond to lower-level WSDL messages, the name of the message would correspond to the WSDL operation.[4] The message type specifies whether the message is *incoming* (In) or *outgoing* (Out), so that we

[3] Restrictions like these are specified in the OWL ontology using appropriate operators.

[4] In this case some type of *grounding* between the interaction protocol and WSDL messages is required, similarly to the OWL-S approach.

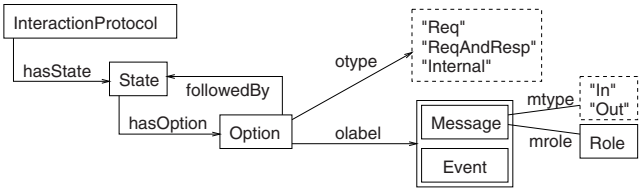

Fig. 2. Overview: (Partial) Interaction Protocol Ontology

can distinguish between messages that are initiated by the service or by other parties. The role defines the role of the party sending or receiving the message. Note that since the interaction protocol is service-centric we always know one of the parties involved in the interaction, i.e. the service under description, so the role serves to identify the other party involved. Finally, events are simply identified by a *name*.

In the case of the request option, the message may be either ingoing or outgoing. Similarly, in the request+response option, one message has to be of type ingoing message and the other of type outgoing, though there is clearly no restriction on whether the request or the response message are ingoing or outgoing. This enables the description of two directions of request+response interactions, one initiated by the service itself, the other initiated by an outside service.

With request+response we are modeling *two* subsequent states, with restrictions imposed on those states. Each state has only one option and each option has one message associated with it, to one state. In order to achieve this, in our ontology we require that a request+response option is followed by exactly one state. The successor state is in any case a state that has only one option. Thus, there are two possibilities (modeled using the union operator): (a) The message associated with the request+response option is an ingoing message, and the successor state is a state with one request option that has an outgoing message associated to it, or (b) the message associated with the request+response option is an outgoing message, and the successor state is a state with one request option that has an ingoing message associated with it (see Figure 3 for the definition).

The definition in Figure 3 guarantees the kind of request+response option that we aim for. In the future we aim to further refine the definition of the content class, including specification of multiple, typed parameters.

4.2 Applying the Model

The interaction protocol of the job service described in Section 2 is illustrated in Figure 4 and can be described using our OWL ontology for interaction protocols (see `www.csl.sri.com/users/denker/sfw/wf/ip-ex.owl` for OWL specification). In the figure states are numbered $S1, S2, .., S11$, while options are represented by the arrows that lead from one state to the other. Attached to options are descriptions of the events or messages concerned. Finally, request+response

```
<owl:Class rdf:about="#ReqAndRespOption">
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Restriction>
   <owl:onProperty rdf:resource="#followedBy"/>
   <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
       <owl:onProperty rdf:resource="#olabel"/>
       <owl:allValuesFrom rdf:resource="#IngoingMessage"/>
      </owl:Restriction>
      <owl:Restriction>
       <owl:onProperty rdf:resource="#followedBy"/>
       <owl:allValuesFrom rdf:resource="#StateWithOneReqOptionWOutgoingMessage"/>
      </owl:Restriction>
     </owl:intersectionOf>
    </owl:Class>
    <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
       <owl:onProperty rdf:resource="#olabel"/>
       <owl:allValuesFrom rdf:resource="#OutgoingMessage"/>
      </owl:Restriction>
      <owl:Restriction>
       <owl:onProperty rdf:resource="#followedBy"/>
       <owl:allValuesFrom rdf:resource="#StateWithOneReqOptionWIngoingMessage"/>
      </owl:Restriction>
     </owl:intersectionOf>
    </owl:Class>
   </owl:unionOf>
  </owl:Class>
 </owl:intersectionOf>
</owl:Class>
```

Fig. 3. Request+Response Definition

options are shown with the intermediate state, as described above, within a dashed line boundary.

The information contained within the model allows us to identify the parties/roles involved, the direction of the interaction, the allowed sequence of interaction as well as relevant state changes of the service which affect the allowed interactions. As a result, when a client attempts to use this service and obtains a description of the service it will reveal that the job service requires a data staging service, and how the interaction with the data staging service takes place. Furthermore, the client can see that the service will attempt to notify it when the computation has completed, implying that the client should be able to accept such messages (we discuss this issue further in Section 5.1). Finally, the client can also identify that there is an event generated by the service after which it can no longer request the status. As an example of the use of the ontology, see Figure 5 specification of the first state transition between states $S1$ and $S2$.

5 Use of Models

In this section we discuss how the service description model can be used to enable reasoning in two crucial cases: first, within the context of providing adaptive

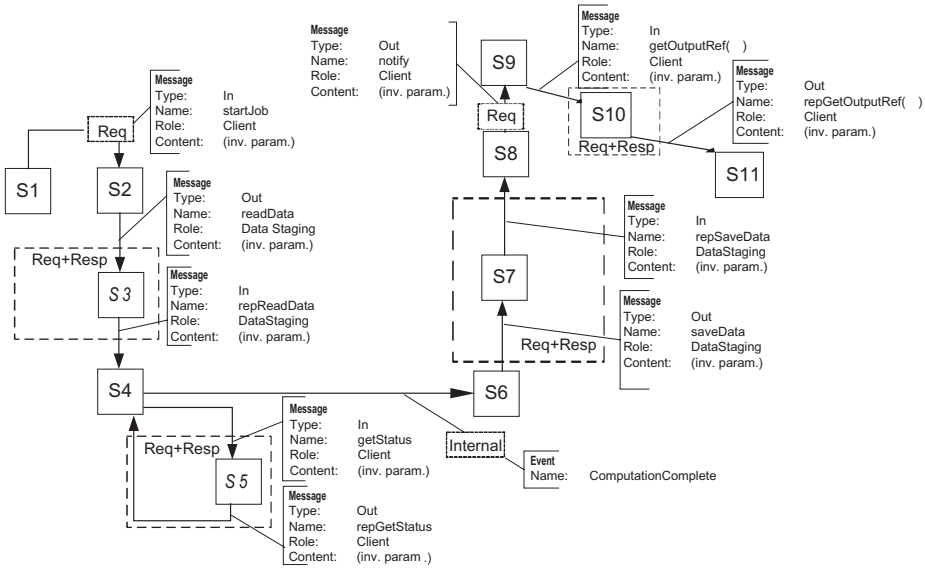

Fig. 4. Job service interaction protocol

security devices that make use of semantic descriptions; and, secondly, within the context of service composition.

5.1 Securing Services

In a previous paper we outlined the requirements for an adaptive security device (a *semantic firewall*), which makes use of semantic descriptions of services to dynamically control lower-level security devices, such as traditional firewalls, so as to enable only *appropriate interactions* [28]. Appropriate interactions are defined as those that are expected given the current context and satisfy any policy requirements associated with the interactions. In that paper, we indicate that OWL-S would be considered as a language for providing the required semantic descriptions of services. However, as discussed earlier some crucial features are missing that would enable its use.

The service description model described here addresses these limitations and enables a *semantic firewall* to establish at runtime which are the appropriate interactions at each stage of the process. Domains offering services to clients can ensure that their services are only used as the service developer defined through the interaction protocol of the service and prevent inappropriate messages from going through. Furthermore, client domains can establish exactly which interactions are initiated by the service and directed towards the client. This will solve a significant shortcoming of existing grid systems where clients are typically behind firewalls that disable any externally initiated messages. Consequently, clients are currently forced to *poll* the job service as to whether the computation has been

```
<!-- Roles for job and client services --> <ip:Role rdf:ID="J"/>
<ip:Role rdf:ID="Client"/>

<ip:Content rdf:ID="ContM1"/>

<ip:Message rdf:ID="M1">
  <ip:mtype rdf:resource="&ip;#In"/>
  <ip:mrole rdf:resource="#Client"/>
  <ip:mname rdf:datatype="&xsd;String">startJob</ip:mname>
  <ip:content rdf:resource="#ContM1"/>
</ip:Message>

<ip:Option rdf:ID="O1">
  <ip:followedBy rdf:resource="#S2"/>
  <ip:otype rdf:resource="&ip;#Req"/>
  <ip:olabel rdf:resource="#M1"/>
</ip:Option>

<ip:State rdf:ID="S1">
  <ip:hasOption rdf:resource="#O1"/>
</ip:State>

<ip:State rdf:ID="S2">
  <ip:hasOption rdf:resource="#O2"/>
</ip:State>

<ip:InteractionProtocol rdf:ID="JobServiceInteractionProtocol">
  <ip:hasState rdf:resource="#S1"/>
  <ip:hasState rdf:resource="#S2"/>
  ...
  <ip:hasState rdf:resource="#S11"/>
</ip:InteractionProtocol>
```

Fig. 5. Ontology use example for job service description

completed. By deploying semantic firewalls and making use of the interaction protocol model described to identify when and which incoming messages should be expected we can overcome this problem.

5.2 Inferring Service Dependencies

Application development using the GRIA infrastructure reveals that services are often developed with other required services in mind. For example, in our motivating scenario the required service was the data staging service. By describing such dependencies through our interaction protocol, clients, having identified a required service, can determine exactly what dependencies the service has on any other services and use that information to attempt to discover suitable services to fulfill those requirements. The use of *roles* to define such required services can aid the discovery process by indicating *classes* of services that can conform to such behaviour.

6 Conclusions and Future Work

In this paper we have present a novel model for describing how a service can interact with clients and other services. This model was developed in response to the identification of a set of requirements based on *real-life* experience with

Grid application development. Existing approaches for service description, most significantly OWL-S, do not adequately fulfill those requirements and as such this work can be seen as informing and furthering the discussion on what is an appropriate model for describing services. In particular, our model offers three specific features that are crucial for enabling the types of Grid-based scenarios we aim to support. Firstly, we are able to describe *multi-party interactions*, with respect to a single service and the different *roles* those parties occupy. Secondly, we can differentiate between interactions initiated by the service and those initiated by other parties. Finally, we can describe relevant service states that influence the subsequently allowed interactions. As discussed in Section 5 such a model can play a crucial role in enabling the development of security devices that take advantage of semantic web-based descriptions of services [28] as well as more sophisticated compositions of services. We now aim to develop appropriate reasoning engines that can be integrated within GRIA for handling such interaction protocol descriptions, with the particular aim of using them within a *semantic fireall*. In addition, we will investigate the integration of policies within the description as well as the verification of an interaction protocol against domain-wide policies. Finally, any high level description techniques adopted to describe and analyse interaction protocols must eventually be grounded in existing and emerging security specifications like WS-Security [29], WS-SecurityPolicy [30] and web service standards such as BPEL4WS [12]) in order to enable their enforcement at the message level. In consequence, further research will include investigation of techniques for mapping high level descriptions on to these Web service standards.

References

1. Ankolenkar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., Drew McDermott, S.A.M., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In Cruz, I.F., Decker, S., Euzenat, J., McGuinness, D.L., eds.: The First Semantic Web Working Symposium, Stanford University, California (2001) 411–430
2. Motta, E., Dominique, J., Cabral, L., Gaspari, M.: IRS-II: A Framework and Infrastructure for Semantic Web Services. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: The Semantic Web - ISWC03. Volume 2870 of LNAI., Springer (2003) 306–318
3. Sycara, K., Paolucci, M., Ankolenkar, A., Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services. Journal of Web Semantics **1** (2003) 27–46
4. Staab, S., van der Aalst, W.M.P., Benjamins, V.R., Sheth, A.P., Miller, J.A., Bussler, C., Maedche, A., Fensel, D., Gannon, D.: Trends and Controversies: Web Services: Been There, Done That? IEEE Intelligent Systems **18** (2003) 72–85
5. Oliveira, E., Rocha, A.P.: Agents Advanced Features for Negotiation in Electronic Commerce and Virtual Organisations Formation Processes. In Dignum, F., Sierra, C., eds.: Agent Mediated Electronic Commerce, The European AgentLink Perspective. Volume 1991. Springer (2001) 78–97

6. Anuj K. Jain, Manuel Aparico, M.P.S.: Agents for process coherence in virtual enterprises. Communications of the ACM **42** (1999) 62–69
7. de Roure, D., Jennings, N.R., Shadbolt, N.: The Semantic Grid: A future e-Science infrastructure. In Berman, F., Fox, G., Hey, A., eds.: Grid Computing: Making the Gloab Infrastructure a Reality. Wiley (2003) 437–470
8. W3C: Web Services Activity. (http://www.w3.org/2002/ws/)
9. W3C: Simple Object Access Protocol. (http://www.w3.org/TR/SOAP/)
10. Christensen, E., Curbera, F., meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. Technical report, (W3C)
11. Bellwood, T., Cl'ement, L., von Riegen, C.: UDDI Version 3.0.1- UDDI Spec Technical Committee Specification. Technical report, OASIS (2003)
12. (Ed.), S.T.: Business Process Execution Language for Web Services Version 1.1. Technical report, BEA Systems, IBM, Microsoft, SAP, Siebel Systems (2003)
13. Mandell, D.J., McIlraith, S.A.: Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: The Semantic Web - ISWC03. Volume 2870 of LNAI., Springer (2003) 227–241
14. Verma, K., Akkiraju, R., Goodwin, R., Doshi, P., Lee, J.: On Accommodating Inter Service Dependencies in Web Process Flow Composition. In Payne, T.R., Decker, K., Lassila, O., Mcilraith, S., Sycara, K., eds.: First International Semantic Web Services Symposium (2004 AAAI Spring Symposium Series), AAAI (2004)
15. Taylor, S., Surridge, M., Marvin, D.: Grid Resources for Industrial Applications. In: 2004 IEEE International Conference on Web Services (ICWS'2004). (2004)
16. Ashri, R., Payne, T., Marvin, D., Surridge, M., Taylor, S.: Towards a Semantic Web Security Infrastructure. In Payne, T.R., Decker, K., Lassila, O., Mcilraith, S., Sycara, K., eds.: First International Semantic Web Services Symposium (2004 AAAI Spring Symposium Series), AAAI (2004)
17. GRIA: Grid Resources for Industrial Applications. (http://www.gria.org)
18. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T., Snelling, D., Vanderbilt, P.: Open Grid Services Infrastructure. Technical report, Global Grid Forum (GGF)
19. Richards, D., Sabou, M.: Semantic Markup for Semantic Web Tools: A DAML-S Description of an RDF-Store. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: The SemanticWeb - ISWC 2003. Volume 2870., Springer (2003) 274–289
20. Paolucci, M., Ankolenkar, A., Sinivasan, N., Sycara, K.: The DAML-S Virtual Machine. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: The SemanticWeb - ISWC 2003. Volume 2870., Springer (2003) 290–305
21. Denker, G., Kagal, L., Finin, T., Paolucci, M., Sycara, K.: Security for DAML Services: Annotation and Matchmaking. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: The SemanticWeb - ISWC 2003. Volume 2870 of LNCS., Springer (2003) 335–350
22. Mika, P., Sabou, M., Gangemi, A., Oberle, D.: Foudations for DAML-S: Aligning DAML-S to DOLCE. In Payne, T.R., Decker, K., Lassila, O., Mcilraith, S., Sycara, K., eds.: First International Semantic Web Services Symposium (2004 AAAI Spring Symposium Series), AAAI (2004)
23. Fensel, D., Benjamins, V.R., Motta, E., Wielinga, B.J.: UPML: A Framework for Knowledge System Reuse. In Dean, T., ed.: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1999) 16–23
24. Burdett, D., Kavantzas, N.: Ws choreography model overview. Technical report, W3C (2004)

25. Milner, R.: Communicating and Mobile Systems: The Pi Calculus. Cambridge University Press (1999)
26. Toivonen, S., Helin, H.: Representing interaction protocols in DAML. In van Elst, L., Dignum, V., Abecker, A., eds.: Agent Mediated Knowledge Management, International Symposium AMKM 2003, Stanford, CA, USA, March 24-26, 2003, Revised and Invited Papers. Volume 2926 of LNCS., Springer (2004) 310–321
27. FIPA ACL Message Structure Specification. Technical report, Foundation for Intelligent Physical Agents (2002)
28. Ashri, R., Payne, T., Marvin, D., Surridge, M., Taylor, S.: Towards a Semantic Web Security Infrastructure. In Payne, T.R., Decker, K., Lassila, O., Mcilraith, S., Sycara, K., eds.: First International Semantic Web Services Symposium (2004 AAAI Spring Symposium Series), AAAI (2004)
29. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004. Technical report, OASIS (2004)
30. Nadalin, A.: Web services security policy. Technical report, IBM, Microsoft, RSA Security and Verisign (2002)