Cooperation and Agreement between Semantic Web Services

Shamimabi Paurobally, Valentina Tamma and Michael Wooldridge
Department of Computer Science, University of Liverpool, Liverpool L69 7ZF
{sha,valli,mjw}@csc.liv.ac.uk

Abstract

Semantic web services frameworks need to support dynamic cooperation, negotiation and coordination between web services for efficient resource and task allocation and execution. Although the OGSI standard includes a web services agreement model, there is no concept of negotiation through protocols and strategies. Even allowable actions are limited to two types. In this paper, we discuss our recent work on adapting and refining some of the extensive research in the multi-agent systems community to facilitate dynamic and adaptive negotiation between semantic web services.

1. Introduction

Grid computing research is investigating applications of virtual organisations for enabling flexible, secure and coordinated resource sharing among dynamic collections of individuals, institutions and resources. The aim is to support globally distributed collaboration between web services through information management and a service oriented approach [5]. Adding ontologies and negotiation capabilities to grid services would help in resource brokering, inter-grid interoperation and agent-like planning for scheduling.

However, current web services work fails to benefit from the agent communities research in negotiation for coalition formation between parties with heterogeneous information needs. In current grid applications, heterogeneity and dynamic provisioning are limited, and dynamic virtual organisations are restricted to those parties with apriori agreements to common policies and practice. Cooperation, coordination and negotiation are issues that Grid users and Grid resource providers need to address in order to successfully interoperate [4]. Given the view that multi-agent systems are groups of agents that interact through cooperation, coordination and negotiation to satisfy their individual or common goals, grid and agent technology can complement each other for efficient provisioning and management of services.

The Ontogrid project [6], started in October 2004, is investigating on how to share and deploy knowledge in grid computing for the rapid prototyping and development of knowledge-intensive distributed open services. The semantic grid architecture that results from this project would stand as a sound methodology for developing grid systems that optimize cross-process, cross-company and cross-industry collaboration. A principle of OntoGrid is to adopt and influence standards in Semantic Grid and Grid Computing, in particular the Open Grid Service Architecture [6]. The Ontogrid project aims to develop open distributed knowledge-based solutions in the grid over the Open Grid Services. Thus, the project's aims include developing knowledge services that are grid aware, enabling a selection of grid services to become knowledge aware and to develop a semantic grid architecture that incorporates peer-to-peer and agent techniques. In this paper we focus on two objectives of the Ontogrid project: 1) to apply grid computing architectures in frameworks involving virtual organizations formed by autonomous entities with conflicting goals. We take these entities to be autonomous and rational (goal-oriented) agents, and 2) to demonstrate that Semantic Grid computing systems can be successfully exploited in e-business applications.

Since automated negotiation can effectively help in resolving conflicts typically over resource allocations and in setting up agreements such as for service provision between autonomous entities, we develop such negotiation mechanisms between web services. Our mechanisms address the current limitation of semantic web services to support dynamic negotiation strategies for task and resource allocation. Our aim is to successfully deploy negotiation and interaction mechanisms between grid services in the same way that autonomous agents negotiate in a collaborative or competitive distributed system. Therefore, our interest in the W3C frameworks for semantic web services workshop resides in the following topics: business processes, negotiation between semantic web services, and automating tasks based on semantic descriptions and integration in the Web Architecture.

The remainder of this paper is structured in the following way. Section 2 presents a usage scenario in the insurance sector to show negotiation between insurance services for sharing information. Section 3 critically analyses current work on grid-based models for service negotiation and service level agreement. Section 4 discusses how we develop negotiation mechanisms for dynamic coordination, task and resource allocation in the semantic grid. Section 5 concludes.

2. Insurance Usage Scenario

We analyse a scenario [12] from the car insurance sector because it involves various actors and information flow between them in claims handling. For example, services that can be invoked in processing claims may be provided by lawyers, medical doctors, bankers and car mechanics. Currently, the insurance market mostly relies on traditional ways of handling claims, which can be slow and costly because of the inter-dependency between the multiple parties involved in expedit-

ing a claim. Thus, automating the various steps in claims handling can help save costs and time, integrate chains of services, exchange substantial data via secure, reliable and robust channels and encourage interactions between insurance companies which would otherwise have not trusted each other. The various actors are considered as being part of a grid and offering grid or web services. A grid computing approach is taken because the heterogeneous parties may maintain long-term relationships, exchange large amounts of data and participate in long-lived transitions. To illustrate the interactions when handling claims for car accidents, we present the case for detecting fraudulent claims [12]. Insurance companies communicate with one other about questionable clients and debatable claims to detect duplicate claims at different companies.

Fraudulent claims may occur when the same claim is made at two or more (international) insurance companies. In current systems, these claims are often not detected because insurance companies, especially if located in different countries, do not share information with each other about their clients and reported damages. Insurance companies place high importance on preserving privacy of information about their customers for commercial reasons or because they can be liable to violating privacy laws. They can also be sued for bad faith or malicious prosecution if they pursue suspicious claims. Therefore, insurers are not allowed free access to the data of other insurance companies, but rather they can only ask specific questions about a specific customer. The actors in this scenario are the insurance services, customers, police authorities and the FraudGridManager. The FraudGridManager helps the insurance services to communicate with one other about questionable clients and questionable damage reports in a rapid and low profile way to avoid encroaching privacy laws.

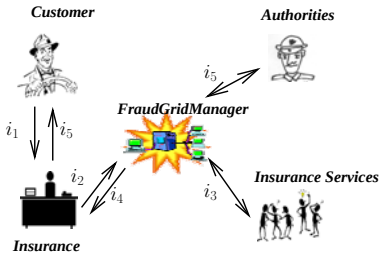

Figure 1. Detecting fraudulent claims

Before dealing with any customers or claims, an insurance service s_1 may negotiate with the FraudGridManager about s_1 querying the FraudGridManager, in the future, regarding a particular customer in return of s_1 providing information to the FraudGridManager about other customers. Two cases are considered for detecting fraud: 1) when a customer wishes to become a client with an insurance service and 2) when an insurance service receives a claim. In the first case, when a customer wants to register as a client, the insurance company if doubtful of the customer's intentions may ask the FraudGridManager to check whether the customer is blacklisted or has a questionable history with other insurance companies. On receiving the FraudGridManager's report, the insurance service may choose whether to accept, reject or negotiate the terms of the insurance with the customer. In the second case of the insurance company receiving a doubtful claim, the interactions, shown in figure 1, are as follows:

- i_1: A customer sends a claim to the insurance service.
- i_2: If the insurance service doubts its client's claim, it requests the FraudGridManager to check the claim.
- i_3: The FraudGridManager contacts other registered insurance services and performs a bilateral negotation with them to decide how much information can be exchanged and would be enough for detecting whether the claim has already been reported elsewhere.
- i_4: The FraudGridManager sends its conclusions about whether a fraud was detected to the insurance service.
- i_5: The insurance service either accepts or rejects the client's claim. The authorities may also be contacted in case frauds are detected.

In this scenario, there are negotiations beforehand between the parties to contract relevant services. On receiving a claim or a request to register as a client, the insurance services (through the FraudGridManager) have to negotiate about what information they are willing to pass to one other. The insurance services have different ways to verify client registrations and car claims. For example, in order to verify a claim to an insurance service I_1, the FraudGridManager asks the insurance service I_2 to check its records. I_2 requests the name and address of the client and his car's mileage, to which I_1 refuses and offers the car's license plate and date of accident instead. As can be seen in this scenario, information is sensitive and valuable, giving rise to negotiation between the services regarding the exchange of clients' information.

3. Frameworks Facilitating WS Negotiation

In this section we discuss existing work that propose to converge the web services and agent communities for service provision, negotiation and problem solving in open distributed systems. In this paper, we analyse three existing proposals in the web services domain that are investigating agents-like solutions for supporting flexible and interoperable web services.

3.1. WS-Agreement

WS-Agreement [1] specifies an XML-based language for creating contracts, agreements and guarantees from offers between a service provider and a client. An agreement may involve multiple services and includes fields for the parties, references to prior agreements, service definitions and guarantee terms. Here the service definition is part of the terms of the agreement and is established prior to the agreement creation. The guarantee terms specify the service levels that the parties are agreeing to and may be used to monitor and enforce the agreement. A service provider publishes an agreement template describing the service and its guarantees. Negotiation then involves a service consumer retrieving the template of agreement for a particular service from the provider and filling in the appropriate fields. The filled template is then sent as

an offer to the provider. The provider decides whether to accept the offer, depending on its resources. There are a number of significant shortcomings [7] in WS-Agreement.

Limited Message Types. The first significant weakness lies in the fact that WS-Agreement messages are limited to two types – *offer* and *agree* – and constrained according to a template a service provider publishes. The WS-Agreement specification is only used at the last stage in a transaction where the parties close their interaction with a contract specified as a WS-Agreement. The *offer* and *agree* templates are not sufficient or appropriate for modelling the negotiations described in the insurance scenario, for auctions or the contract net protocol [11]

No Interaction Protocols Even with a more varied set of messages (speech-acts), WS-Agreement still suffers from the lack of an interaction protocol specified between parties. That is, even if we increase the WS-Agreement schema with various speech-acts, there is no concept of how to sequence messages through interaction protocols to form a valid conversation. This is the second significant weakness. There is only a two step conversation, an *offer* followed by an *agree*. Without an adequate set of speech-acts and specification of how to construct interaction protocols, the usefulness of a WS-Agreement exchange is limited to cases such as buying from catalogues, with take-it or leave-it offers from the seller or buyer.

Lack of Semantics On the whole, WS-Agreement is a specification with vague and unclear semantics. The WS-Agreement specification only defines a higher-level template for agreements and offers. There is the need of a language to express the elements in the service description terms and guarantee Terms. Thus there is no indication of how to access or provision a service from an agreement.

3.2. Grid Resource Management

The i2CAT project [3] proposes an agent-based architecture for Grid Resource Management (GRM) based on an agent marketplace. GRM concerns management of user requests through resource and task scheduling and one of the issues GRM must address is negotiation between resource users and resource providers. In the i2CAT projects, agents representing grid users find and meet agents representing grid services, negotiate with them about a service and , a task satisfying the service is executed on the Grid node. The parties negotiate according to the contract net protocol [11], a well-known negotiation protocol in the agent community. A broker (or normally manager) agent makes a call for proposal to a set of contractors to execute a task, contractors send back proposals or bids regarding executing that task, the broker selects and accepts one or more proposals and finally accepted contractors send back the results of the task execution. In i2CAT, the negotiations are influenced by the utility assigned to the possible grid configurations, which records, amongst others, the state of the resources and expected task completion time. A broker agent check the utilities to choose seller bids. The agents are implemented using JADE over Globus Toolkit 3 as Grid middleware. The issue in the negotiation in this architecture is again that this is a one-shot negotiation where there is no flexibility for the parties to iteratively adapt and modify their bids and converge to a mutual agreement about how to manage resources. The other weakness lies in that the negotiation strategies used are quite simple and does not exploit the significant work done in both game theoretic economics and automated agent negotiation.

3.3. Negotiation for Routing in Grid

Ayienga et al. [2] propose to use agent mediated electronic commerce techniques for the provision of QoS at the network level in a grid environment. Self-interested agents negotiate in order to establish and maintain a certain level of QoS in the Grid network [2]. They address the problem faced by the Internet for supporting Grid applications requiring high bandwidth and low latency. In their architecture, agents locally computes solutions based on limited information (and thus a decentralised architecture) and use this information in a social way. Link agents are producers of resources and aim to maximise their income by selling network resources to path agents. Path agents negotiate one or more times for links with Link agents, in order to form a route with the right QoS parameters to satisfy a service request. However, none of the negotiation component has been implemented yet and it would be interesting to see how negotiation decision functions are actually specified and implemented in this dynamic environment and how the project evolves.

4. Developing Negotiating Semantic Web Services

In order to negotiate and set up agreements, web services need protocols that govern and structure interactions between them. In this section, we present a negotiation mechanism for cooperation and agreement in the Semantic Grid in the Ontogrid project. Such interaction is to enable grid proceses to better manage their activities and resources at run-time, with a reduced requirement for design time coordination and allocation regimes. On the other hand, the grid can facilitate agent communication, life-cycle management and access to resources for agents [6]. The end aim is to develop:

- Brokering algorithms for dynamic coalition formation in the insurance scenario and for the semantic registry.
- English auction with timeouts for resource allocation.
- Contract net protocol for task allocation.
- Bilateral alternating offers protocol for bargaining.
- Coordination ontology for detecting and resolving conflicting interaction in accessing resources.

In this paper we show how we adapt the Rubinstein's alternating offers protocol for web service negotiation. Rubinstein's [8] protocol of alernating offers is a well-known strategic negotiation model. In this model two parties *A1* and *A2* participate in the negotiation process and makes offers and counter-offers. Rubinstein first proposed this model of alternating offers to describe how two players could share or partition a pie of size 1. He assumed that the players cannot opt out of the negotiation. He considered situations of fixed discount costs over time and fixed discounting factors. He later extended this protocol to deal with incomplete information scenarios. In our

case also, we assume the web services have incomplete information about their opponents' preferences.

The features of negotiation normally include a communication language, an interaction protocol and the decision process used to determine responses, concessions and criteria for agreement [10]. In the following sections, we adapt each of these three components to the web services domain for the case of the Rubinstein's alternating offers protocol.

4.1. Alternating Offers Negotiation Language

The actions in this model are: *offer, counter-offer, agree, reject*. These actions are typically called speech-acts [9] and in our XML definition we type them as `wssa` (for Web Service Speech Act). Each of these actions have as parameters the sender, receiver, service description, the process to be executed and the agreement terms such as deadline of carrying out the agreement, penalty of not meeting the agreement or conditions for de-committing to the agreement. Below, we show the XML definition of a counter-offer.

```
<wssa:counter-offer>
 <wssa:Name> "xsd:NCName" </wssa:Name>
 <wssa:Parties>
    <wssa:Sender> xs:anyType </wssa:Sender>+
    <wssa:Receiver> xs: anyType </wssa:Receiver>+
 </wssa:Parties>
 <wssa:service>
    wssa:Name="xs:NCName" wssa:ServiceName="xs:NCName">
     <xsd:any> ... </xsd:any>
 </wssa:service>
 <wssa:Agreement wssa:Name="xs:NCName">
   <wssa:Deadline> xs:Time </wssa:Deadline>
   <wssa:Penalty> xsd:integer </wssa:Penalty>
   <wssa:Reward> xsd:integer </wssa:Reward>
   <wssa:Decommitment> xsd:any </wssa:Decommitment>
 </wssa:Agreement>
</wssa:counter-offer>
```

The other actions, *offer, agree, reject* can be similarly defined in XML. In the next section, we show the interaction protocol for sequencing these actions. For example, an offer can be followed by a counter-offer but not by another offer, or an agree cannot follow a rejection.

4.2. Alternating Offers Negotiation Protocol

The parties can act in the negotiation only at discrete time points in the set $T = \{0, 1, 2, \ldots\}$. At each instant t ($t \neq 0$) in the negotiation, if the negotiation has not yet terminated, the agent, whose turn it is to respond, may send a counter-offer, agree, or reject. If an offer or counter-offer made by the agent *A1* at time instant *t* is accepted by *A2* then the negotiation terminates. If *A2* rejects the offer/counter-offer the process ends in a conflict.

We express the bilateral protocol shown in figure 2 in XML so that it is in a language that web services can understand. A protocol in XML consists of a `name`, a set of `states` and `transitions`.

First we define XML templates for states and transitions. As in finite state automata, transitions, here through exchanging speech-acts or messages, lead to a change in states – from source state to target state. For example, in the source state *offered*, sending an *agree* message triggers the target state *agreed*. We define the type `state` in XML as having a name, a boolean attribute (whether the state holds or not), and optionally includes the service that triggered the state, the recipients and any action needed.

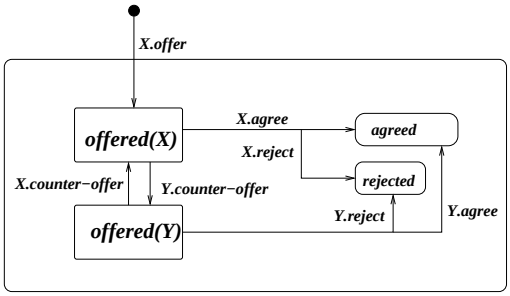

Figure 2. Alternating Offers Protocol

```
<xsd:element name = "State"/>
  <xsd:complexType>
    <xsd:attribute name="xs:Name" type="xs:boolean"/>
    <xsd:sequence>
     <xsd:element name="Initiator" type="wsag:Initiator"/>
     <xsd:element name="Respondent" type="wsag:Respondent"/>
     <xsd:element name="Process" type="wsdl:Operation"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The type `Transition` is defined in XML as having attributes name, source and target states, sender (perpetror) and recipients of the transition. A transition that initialises a conversation may not have a source state and therefore the source state is an optional field. In contrast, the target state is a compulsory field. A `Transition` also has an optional `Process` field to encode relevant information such as offering to carry out an action α, where the transition is the speech-act name and α is the process attribute.

Using the definitions of states and transitions, we show below a partial specification in XML of the salient parts of the Rubinstein's bilateral protocol shown in figure 2. It can be seen that we are now able to specify which service sent the last message and whose turn it is now to respond and what are the valid responses.

```
<Protocol name="BilateralProtocol"
  <States>
    <State Name="offered(X)" >
    <!-- X sent an offer-->
       <Initiator> X </Initiator>
       <Respondents> Y </Respondents>
       <Process> α </Process>
    </State>
    <State Name="agreed(Y)" >
    <!-- Y sent an agree-->
       <Initiator> Y </Initiator>
       <Respondents> X </Respondents>
       <Process> α </Process>
```

```
    </State>
  </States>
  <Transitions>
   <Transition Name = Y.counter-offer>
    <!-- Y sends a counter-offer from an offered state-->
    <!-- for Y to do α-->
     <SourceInteraction href="offered(X)"/>
     <DestinationInteraction href="offered(Y)"/>
      <Trigger trigger-process "Y.offer">
      <!-- offer for Y itself to perform α-->
        <Sender> Y </Sender>
        <Recipients> X </Recipients>
        <Action> α </Action>
      </Trigger>
    </Transition>
    <Transition Name = X.reject>
     <!-- X sends a rejection from an offered state-->
      <SourceInteraction href="offered(Y)"/>
      <DestinationInteraction href="rejected(X)"/>
       <Trigger trigger-process "X.reject">
       <!-- rejection from X for Y itself to perform α-->
         <Sender> Y </Sender>
         <Recipients> X </Recipients>
         <Action> α </Action>
       </Trigger>
    </Transition>
  </Transitions>
</Protocol>
```

4.3. Alternating Offers Negotiation Strategies

There are a number of attributes that negotiation strategies ideally seek to satisfy so that the parties interact productively and fairly. These attributes include efficiency (no wastage of resource), stability (no incentive to deviate from agreed strategies), simplicity (low computational and bandwidth costs), distribution (no central decision maker) and symmetry (no arbitrary biais against any web service). Here we show a strategy suitable for use in negotiation between web services.

Let the set of all possible agreements be denoted by *S*. The outcome of the negotiation is an agreement *s* reached at time *t* and is denoted by the ordered pair *(s, t)*. A disagreement denotes a rejection or a negotiation that continues forever without reaching an agreement. Each web service would prefer to agree on an outcome that is most favourable to it. The utility functions guide the web services in the process of distinguishing favourable outcomes from unfavourable ones. A web service's negotiation strategy defines what the web service should do on receiving an offer. Let *S* denote the set of all possible agreements that can be reached at time $t = \{0, 1, 2, \ldots\}$ and the offers that a web service can receive denoted as $\{s0, s1, s2, \ldots\}$. A web service's strategy can be defined as $f : \{s0, s1, s2, \ldots\} \rightarrow S \cup \{reject\}$ where *reject* denotes rejecting an offer or counter-offer.

The utility function assigns a numerical value to the actions of the web services. One of the main problems is to determine a strategy that would yield maximum utility for the web services. Formally the utility function can be defined as $U : \{S \cup \{reject\} \times T\} \rightarrow R$.

For every service $i\,\{i = 1, 2\}$ and time period *t*, *Possible(i,t)* is the set of all agreements that web service *i* prefers to rejection. If this set is non-empty then the agreement *s (i , t)* that belongs to *Possible (i ,t)* is called the *Best Possible Agreement* if it satisfies $U^i(s(i,t)) = max_{s \in Possible(i,t)} U^i(s)$.

Analogously the Worst Possible Agreement is defined as *s(i, t)* if it satisfies $U^i(s(i,t)) = min_{s \in Possible(i,t)} U^i(s)$. If *S1* is the strategy adopted by web service *A1* when the strategy adopted by web service *A2*, then the pair *(S1, S2)* forms an *equilibrium solution* if neither can deviate from their strategy without losing utility. It is one of the main objectives of game theoretic problems to determine equilibrium solutions.

5. Conclusions

The aim of our work in the context of the Ontogrid project is to develop semantic web services that can cooperate, negotiate, reach agreements and self-organise in order to effectively and correctly solve overall problems [6]. Having analysed existing proposals for enabling negotiation between web services, we have in particular found the OGSI WS-Agreement model to be limited and not expressive enough for profitable semantic web service negotiation. Thus, in this paper we show our specification of negotiation actions and protocols to enable interaction between web services and we give an overview of our model for strategic decision making. Future work includes refining and implementing our specifications, for the deployment of semantic web services with negotiation capabilities.

References

[1] A. Andrieux, K. Czajkowski, A. Dan, and et al. *Web Services Agreement Specification (WS-Agreement)*. World-Wide-Web Consortium (W3C), 2004.

[2] E. Ayienga, B. Manderick, O. Okello, and A. Nowe. Multi-agent systems for efficient quality of service routing in grids. *ERCIM News*, 59:39–42, 2004.

[3] I. Chao, R. Sanguesa, and O. Oardaiz. Grid resource management using software agents. *ERCIM News*, 59:39–42, 2004.

[4] European Research Consortium for Informatics and Mathematics. Grids: The next generation. *ERCIM News*, 59:39–42, 2004.

[5] I. Foster and C. Kesselman. *The Grid2, Blueprint for a new computing infrastructure*. Elsevier, 2004.

[6] Ontogrid Project. *Paving the way for Knowledgeable Grid Services and Systems*. `http://ontogrid.net/`.

[7] S. Paurobally and N. Jennings. Protocol engineering for web service conversations. *Engineering Applications of Artificial Intelligence, Special Issue on Agent-oriented Software Development*, 18(2):237–254, 2005.

[8] A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982.

[9] J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, 1969.

[10] M. Singh and M. Huhns. *Service-Oriented Computing*. Wiley, 2005.

[11] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1981.

[12] J. Smulders, C. Van Aart, P. Van Hapert, V. Fintelman, and P. Storms. *Ontogrid, Deliverable 9.1, Businesss Cases and User Requirement Analysis*. `http://ontogrid.net/`.