Developing Agent Web Service Agreements

Shamimabi Paurobally
University of Liverpool
Department of Computer Science
Liverpool L69 7ZF, UK.
sha@csc.liv.ac.uk

Nicholas R. Jennings
University of Southampton
School of Electronics and Computer Science
Southampton SO17 1BJ, UK.
nrj@ecs.soton.ac.uk

Abstract

Web services have emerged as a new paradigm that supports loosely-coupled distributed systems in service discovery and service execution. Next generation web services will evolve from performing static invocations to engaging in flexible interactions and negotiations for dynamic resource procurement. To this end, this paper applies an agent-oriented based approach over a recent web service language, WS-Agreement, in order to facilitate conversations of sufficient expressiveness between adaptive and autonomous services. We discuss how such agent web service agreements can be implemented over IBM's Emerging Technologies Toolkit (ETTK) that itself includes an implementation of the WS-Agreement specification.

1. Introduction

Generally speaking, the aim of the web services endeavour is to obtain an environment where service customers and providers can locate each other, connect with one another dynamically, set (negotiate) the terms and conditions of service invocation automatically and then execute the necessary actions according to the prevailing contract. Given the increasing popularity of web services, next generation web services are to be embedded in goal-driven environments where dynamic resource procurement, business to business collaboration and adaptation to changes will be common. The Web Service Agreement specification (WS-Agreement) [1] starts to capture such high-level service interactions. However, the web service effort will not fulfill its full potential because current web service proposals, including the WS-Agreement specification, are limited to simple request-response exchanges in which a web service remains a self-contained application without any ability to collaborate with other web services in order to satisfy a request. In particular, such simple message types are unsuitable for coordinating transactions between multiple web services because of the explosion in the amount of communication. Moreover advanced transactional systems where participants continuously tailor their needs and offers are also beyond the scope of request-response messages because of the absence of negotiations. It is thus fundamental to develop protocols and mechanisms enabling dynamic negotiations between services rather than present static single-shot interactions.

Given this and because such issues have been extensively researched upon in the software agents community, this paper argues that work on enabling interaction between web services would benefit from the insights and techniques from the field of multi-agent systems. In more detail, we build over the WS-Agreement templates, while remaining compatible to the specification, to propose richer message types and XML-based interaction protocols. We say that our approach facilitates agent web service agreements. We also discuss the beginning of an implementation for developing agent web service agreements over IBM's ETTK toolkit [4]. The work in this paper considers more pragmatic issues than our previous work in [7]. Compared to [7], here, 1) we ground our work with a scenario in the insurance sector, 2) since WSCL (Web Service Conversation Language) has become obsolete, we adapt our approach to only be compatible with WS-Agreement rather than propose a WSCL/WS-Agreement extension as in [7] and, 3) we discuss our first attempts at implementing agent web service agreements.

This paper advances the state of the art through a framework that extends the conversational capabilities of web services by supporting non-trivial interactions in which several messages have to be exchanged before the service is completed and/or the conversation may evolve in different ways depending on the state and the needs of the participants. This increase in flexibility and expressiveness is achieved through the use of speech-acts [2] such as *inform* and *bids* (rather than just offer-accept as is the case currently). Second, as the number of services to be integrated grows and the environment becomes more dynamic, our work should help developers to understand how to write clients that flex-

ibly interact with a service and to develop automated tools to dynamically bind to a service based on the specified characteristics. We also contribute to research in agent interactions, where, to date, much of the work has yet to be put to test in open and dynamic environments. While multi-agent systems research provides intelligent techniques for brokering, negotiation and market design in grid computing, grid technologies are also useful testbeds for large scale deployment of agent systems.

The paper is structured as follows. Section 2 presents a scenario with negotiations in the car insurance sector. Section 3 critically analyses the WS-Agreement specification. Section 4 describes our layers over the WS-Agreement layer to enable agent web service agreements. Section 5 is an overview of a starting implementation of agent Web Service Agreement. Section 6 presents our conclusions.

2. Scenario of Service Negotiations

We analyse a scenario [9] from the car insurance sector because it involves various actors and information flow between them in claims handling. Currently, the insurance market mostly relies on traditional ways of handling claims, which can be slow and costly because of the interdependency between the multiple parties involved in expediting a claim. Thus, automating the various steps in claims handling can help save costs and time, integrate chains of services 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. We present two cases to illustrate the interactions when handling claims for car accidents: 1) *Repair Claims*. Managing repair claims and involved businesses. 2) *Detecting Fraudulent Claims*. Detecting duplicate claims at different insurance companies.

2.1. Repair Claims Scenario

In figure 1, a customer is insured at a company offering insurance services. Car repair services are carried out by the damage repair company. The manager is a company that provides services for managing the businesses involved in dealing with car damage claims to insurance companies. The manager service aims to enhance the quality and efficiency in handling damage claims.

Before any repair claims are received, the repair service can negotiate a contract with the manager service and insurance companies for bidding on repair jobs from insured customers. The terms of a contract may include price of material and labor, speed and quality of repair jobs. All parties to the contract can re-negotiate their contracts if they are not satisfied with the repairs or payments. Customers buy services from insurance companies which negotiate with the manager service on the best price and quality for their insured customers. After an accident, the following interactions to handle a claim occur between the services:

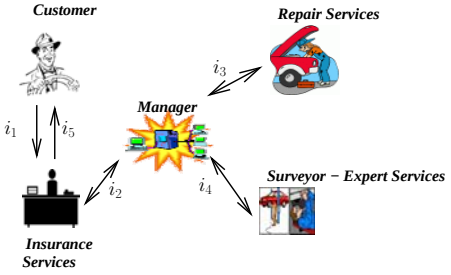

Figure 1. Interactions in repair claims

- i_1: A customer makes a claim to the insurance service.
- i_2: The insurance service requests the manager to find the most appropriate repair service based on a description of the damage.
- i_3: The manager selects a number of repair services and ask them to provide an offer. The repair companies analyse the damages and make an offer to the manager, who decides which proposal to accept or reject.
- i_4: The manager can employ the services of an expert surveyor to analyse the quality of repairs on the damages and the charged costs. If a party is not satisfied, the contract between the manager and repair services may be revised.
- i_5: The insurance company will pay the repair company and/or the victims of the accident.

On analysing the above scenario, it can be seen that negotiation arises at various points between the services – before and when any specific accident and claim is handled. During the claims process itself, there is a negotiation following a contract net protocol between the manager and repair services. The manager makes a call for proposal for a repair job, the repair services send offers to carry out the repairs and the manager selects the most appropriate offer. Thus the interactions between the various services are not always static and pre-determined.

2.2. Fraudulent Claims Scenario

The second scenario [9] we consider involves detecting fraudulent claims, where 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. 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 FraudManager. The FraudManager 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.

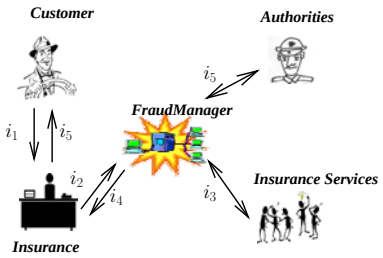

Figure 2. Detecting fraudulent claims

Before dealing with a claim, an insurance service s_1 may negotiate with the FraudManager about s_1 querying the FraudManager regarding a particular customer in return of s_1 providing information to the FraudManager about other customers. On receiving the FraudManager's report, the insurance service may choose whether to accept, reject or negotiate the terms of the insurance with the customer. The interactions when an insurance company receives a doubtful claim are shown in figure 2 and 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 FraudManager to check the claim.
- i_3: The FraudManager 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 FraudManager 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.

As for the repair claim 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 FraudManager) 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 FraudManager 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. Web Services Agreement (WS-Agreement)

The above scenarios show negotiations occuring between the various services. Thus the next generation of web services have to be capable of flexible interactions and negotiations in open environments. In this section, we analyse whether the web services agreement specification [1] can answer to this need.

3.1. Web Services Agreement Structures

WS-Agreement specifies an XML-based language for creating contracts, agreements and guarantees from offers between a service provider and a client. In this case, 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. In more detail, an agreement is defined as being composed of:

1. *Name* identifies the agreement and is used for reference in other agreements.
2. *Context* includes parties to an agreement, reference to the service provided and to possibly other related or prior agreements.
3. *Service Description Terms* provide information to instantiate or identify a service to which the agreement pertains.
4. *Guarantee Terms* specify the service levels that the parties are agreeing to and may be used to monitor and enforce the agreement (e.g. the penalty upon failure to meet the objective or the strength of a commitment by a service provider).

An agreement template follows the above structure. 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 (of type `wsag:offer`) to the provider. The provider decides whether to accept (of type `wsag:agree`) or reject the offer, depending on its resources.

3.2. Weaknesses of WS-Agreement Contracts

There are a number of significant shortcomings in the WS-Agreement specification.

Limited Message Types The first significant weakness lies in the fact that messages in WS-Agreement are limited to two types – *offer* and *agree*, according to a template published by a service provider. The WS-Agreement specification is only used at the last stage in a transaction where the parties are closing 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 scenarios in section 2.

No Interaction Protocols Even with a more varied set of messages, WS-Agreement still suffers from the lack of an interaction protocol specified between parties. 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. For example the WS-Agreement specification is not expressive enough to specify the Contract Net protocol which is probably the most widely used interaction protocol in the multi-agent systems field and which occurs in the insurance scenario for repair claims (section 2). Even if we increase the WS-Agreement schema with various speech-acts, there is no concept of how to sequence messages to form a valid conversation.

Lack of Semantics On the whole, WS-Agreement is a complex specification, with vague and unclear semantics. Significant work is required in clarifying the interfaces before it is successful in enabling web services interactions. Furthermore, 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, for example labelling of messages that are mutually understandable to all parties. Projects such as Ontogrid [6] are investigating how to render grid services semantically aware. Finally, currently the interaction is biaised towards a service provider since the customers rely on the agreement templates published by the provider to create agreements that the latter will understand.

4. Agent Web Service Agreements

As discussed in the previous section, messages in WS-Agreement are limited to two types – offer and agree. To remedy this, we specify other types of WS-Agreement messages in terms of the information contained in the `Context`, `Service Description Terms` and `Guarantee Terms` fields in a WS-Agreement message. The additional message types (which can be regarded as speech-acts [8]) that we define in XML are: *inform* and *bid*. Other speech-acts may be similarly defined. These messages are commonly used in agent interactions [2]. First we add two types to the WS-Agreement `Context` field to express perpetrators and recipients of (1) exchanged speech-acts and (2) process executions. We define a speech-act in XML as a complex type, called `Speech-Act`, with attributes the sender, the list of recipients and any action to be executed by a particular service. For example, if the web service *s* sends a speech-act *sa(r,*α*)* to web service *r*, the details about the participants (*s* and *r*) and the action α in *sa* are added to the context field. Thus, there is one initiator (the sender *s*) of the speech-act, a list of recipients *r*, and the action α. The action α itself may be associated with those services responsible for executing it and those services receiving the result of its execution.

```
<xs:complexType> xs:Name="Speech-Act "
   <xs:attribute> Name="xs:NCName" </xs:attribute>
     <xs:sequence>
       <xs:element name="Initiator" type="xs:NCName"/>
       <xs:simpleType name="Respondents" use="optional">
         <xs:list item Type="xs:NCName"/>
       </xs:simpleType>
       <xs:element name="Process" type="wsdl:Operation"/>
     </xs:sequence>
</xs:complexType>
```

4.1. New Speech-Act Messages in XML

The *inform* speech-act is a basic one that that can be used to define others. Here the meaning of an *inform* is that the sender *s* informs the receiver *r* that a given proposition ϕ is true. From the XML representation of *inform* given below, it can be seen that *s* and *r* are included in the `Context` field and ϕ of type `boolean` included in the `ServiceDescriptionTerm` field.

```
<wsag:Inform>
  <wsag:Name> NCName </wsag:Name>
  <wsag:Context>
     <wsag:Initiator> Sender "s" </wsag:Initiator>
     <wsag:Respondents> Receiver "r" </wsag:Respondents>
   </wsag:Context>
  <wsag:Terms>
      <wsag:ServiceDescriptionTerm wsag:Name="inform"
                wsag:ServiceName="xs:NCName">
         <xs:element name="inform" type="xs:boolean"
                value="φ"/>
      </wsag:ServiceDescriptionTerm>
  </wsag:Terms>
```

```
</wsag:Inform>
```

In the case of a *bid* message, let sender *s* make a bid to receiver *r* with attributes γ in the bid. Here *r* may be a web service acting as an auctioneer such as the manager in the repair claims scenario in figure 1. Receiver *r* may be a web service outsourcing a task and accepting bids from other web services. Such bids and auctions can allow web services to collaborate and form an agreement about task execution. As for *inform*, bids are defined in the `Context` and the `ServiceDescriptionTerms` fields. In this case, the condition for *s* to execute γ in the bid is that its bid is received by *r* and is the winning bid, requiring *received(r, bid(s,*γ*))* & *winning_bid(bid(s,*γ*))* to be true. Other message types such as accept, call for proposal, reject, cancel, propose and request can be similarly specified by encoding the semantics of the message in the `Context` and `ServiceDescriptionTerms` fields of messages of type `wsag`, and thus remaining compatible with the WS-Agreement specification.

4.2. Sequencing Messages

Interaction protocols enable sequencing of messages to form conversations and are currently absent from the WS-Agreement specification. In order to support messages sequences and conversations with web services, we introduce a layer above the WS-Agreement compatible messages layer for specifying agent interaction-like protocols, as shown in figure 3.

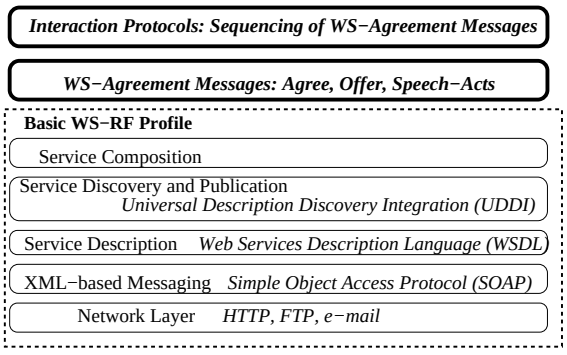

Figure 3. Protocols over WS-Agreement

The lower layers are established web services standards, and are included in the WS-Resource Framework (WS-RF) basic profile [3]. WS-RF is a family of specifications aimed at providing mechanisms for exposing and manipulating web service resources. Interaction protocols are found at the topmost layer in figure 3. To develop interaction protocols in XML over these layers, we specify in XML templates for states and transitions. As in finite state automata, transitions, here through exchanging WS-Agreement compatible 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. By way of illustration, the state *offered* triggered by sender *s* when sending a message to receiver *r* offering to do action α is expressed as follows in XML:

```
<State Name="offered" value="true" >
    <Initiator> s </Initiator>
    <Respondent> r </Respondent>
    <Process> α </Process>
</State>
```

The type `Transition` is defined in XML as having attributes name, source and target states, sender (perpetror) and recipients of the transition. A transition is of type `wsag` (WS-Agreement). Exchanged messages such as *offer*, *agree* and the new types of messages we can define as shown in section 4.1 (`bid, accept, cancel`) are also of type `wsag`. These messages are the transitions in an interaction protocol. Thus our specification of ACL-like messages and interaction protocols in XML remain compatible with WS-Agreement specification.

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 message type and α is the process attribute. For example, a transition named *bid* from sender *s* to receiver *r*, with source state *posted* and target state *sold* and with process α as part of the bid (*s* may bid to do α) is expressed as follows:

```
<Transition Name="Bid" >
    <Source_State> posted </Source_State>
    <Target_State> sold </Target_State>
    <Initiator> s </Initiator>
    <Respondent> r </Respondent>
    <Process> α </Process>
</Transition>
```

We may also define a complex type called `Complex_Transitions` which can be an atomic action, a WS-Agreement message as above, a sequence, alternation or iteration of `Complex_Transitions` or simply the `null` transition. Given the above extensions to WS-Agreement and the new types we defined in XML, it is now possible to express richer interac-

tions between web services. Interaction protocols such as the Contract Net protocol, auctions and bilateral negotiation protocols can now be expressed using our approach. We do not show this in this paper because it is straightforward and instead we discuss the issues in implementing these agent web services interactions.

5. Implementing Agent WS Agreements

Given the above specifications for interactions and agreements between web services leading to contracted executions, both service providers and consumers need an infrastructure to create and manage offers and agreements. In this section, we discuss the implementation of a framework supporting agent web services agreement. Such a framework should help service providers to manage agreement templates, decide whether to accept or reject incoming offers, or choose another response according to the interaction protocol, after analysing available services and capacities, and finally to execute and comply with the agreement terms. In the case of service consumers, the framework should allow them to request agreement templates, create offers from the templates, respond appropriately to received messages and dynamically monitor the state of an agreement.

5.1. Cremona and Emerging Technologies Toolkit

Cremona (Creation and Monitoring of Agreements) [5] proposes an API for implementing the WS-Agreement specification as a middleware for supporting contract agreements between web services. An implementation of Cremona is incorporated in IBM's Emerging Technologies Toolkit (ETTK) [4]. ETTK helps programmers in designing and developing potentially autonomic web services and contains implementations of various web service specifications such as WS-Agreement, WS-Resource Framework, WS-Notification and WS-Addressing [3]. In this section, we summarise the components in Cremona that are salient for developing web services negotiation.

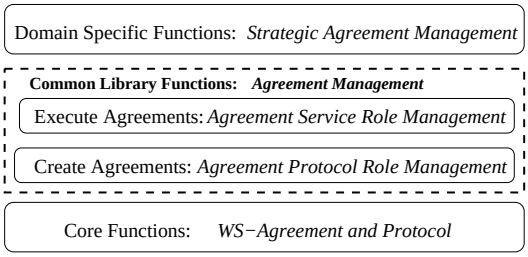

Figure 4. Layers in the Cremona Architecture

Figure 4 shows the layered architecture provided by Cremona for creating and managing agreements and their templates [5]. The lowest layer contains the core functions based on the web services standards SOAP, WSDL, UDDI and WS-Resource Framework. The second layer, Common Library Functions, implements the WS-Agreement specification. The Common Library Functions are a set of libraries, agreement management and monitoring functions that are public to providers and customers complying to the WS-Agreement specification. Different agreement management functions are provided according to the role of a service – service provider or customer.

The Agreement Protocol Management (APRM) component implements interfaces to create agreements and access the agreement state at run-time. It contains a *template set* containing valid agreement templates that customers use to submit offers. On receiving an offer, the *agreement factory* consults the provider's decision maker regarding whether the offer can be accepted. If an offer is accepted, an *agreement instance* is created and registered in the *agreement set*. Eventually, the agreement is announced to the *agreement implementer* which checks and readies the service for when the agreement is to be executed. Finally, an agreement is sent to the service customer through an `agreement instance proxy`. A *status monitor* interface allows to retrieve the status of an agreement, the service description terms and the guarantee terms, for example, whether the agreement state is fulfilled or delayed or a processing job is waiting or completed.

The Agreement Service Role Management (ASRM) component is located over the APRM (see figure 4) and facilitates provisioning and consuming services, while monitoring compliance to an agreement at run-time. The ASRM of a service provider contains an admission control component which interacts with the service implementing system to determine whether there are enough resources to satisfy the resource requirements in an agreement. If there are sufficient resources, an agreement implementation plan (for example a provisioning workflow) is generated and submitted to the provisioning system. Finally a compliance monitor checks whether the guarantees in an agreement instance are met. If guarantees are violated, the compliance manager interface is invoked and may cause a change in job scheduling or system configuration.

The topmost layer, Strategic Agreement Management, defines domain specific functions such as strategic decision making, job scheduling and resource management and that are implemented by web service providers and customers.

5.2. Implementing Flexible Interactions

The above Cremona implementation in IBM's ETTK lies underneath our intended implementation for more flexible agent-oriented interactions and negotiations between web services. Service providers and customers are imple-

mented to use the Cremona interfaces and our implementation for automated negotiation, according to their roles, and to create agreements, offers and other speech-act-like message types that are compliant with WS-Agreement. Implementations of the service provider and client application are respectively based on `aprm.provider` and on `aprm.initiator` packages. A provider advertises the kinds of services it offers and exposes its interfaces to the Agreement Manager (in the APRM) for potential customers to submit offers.

In addition, our layers over WS-Agreement – various message types and message sequencing to form interaction protocols – and their semantics are encoded in the agents' knowledge bases as libraries of messages and interaction protocols. The interaction protocols are implemented as XML documents which are consulted by the customer and provider at start up when they read the configuration files. Negotiation strategies are programmed in the Strategic Agreement Management component to help an agent choose between templates and decide which offer or response to make or accept. These are system and domain specific functions.

In more detail, on the provider's side, the java package has to implement interfaces for the agreement decision maker, agreement implementer and status monitor components as classes. Each of those classes have a factory function to determine and create a corresponding instance for a particular agreement or message type. Depending on an agent's role and the interation protocol, we may implement the provider as an auctioneer or seller with its own preferences and strategies encoded in the decision maker. In the case of the client application, who is the one who initiates an interaction with an offer, its implementation is based on the classes exposed in the Cremona initiator package. The client implementation should allow connection to one or more provider factories, normally through proxy interfaces. The client application is implemented to have strategies and associated utilities, located in its own Strategic Agreement Management component, allowing it to select templates published by the provider and to fill the templates to create an offer at the factory. Similar to the provider, the client also implements the agreement implementer and status monitor. The decision maker creates an agreement decision object that may contain a decision, a context and a reply sent to the other party.

As future work, we intend to implement the Strategic Agreement Management component and the negotiation mechanisms in it for the decision making interface. We will implement brokering protocols using the semantic registry of the WS-Resource Framework to enable coalition and virtual organisation formation between web services. Resource allocation between services will be carried out through English auctions with deadlines, while the contract net protocol will enable task allocation, as occuring in the insurance scenarios in section 2. Bargaining strategies will be implemented through bilateral alternating offers protocols and used in the insurance scenarios implementation.

6. Conclusions

In this paper, we have focussed on enabling flexible interactions between web services because they are fundamental if web services are to reach their full potential in future networked environments. In such environments, there are a number of limitations on the applicability of the web service agreement (WS-Agreement) proposal. Specifically, this paper addresses the need for developers to code client and provider applications that can bind to and interact with services of a specific type. To achieve this , we have specified speech-acts as WS-Agreement schemas in XML for richer messages in conversations than just offer and agree. Consequently, protocols of realistic expressiveness (such as the Contract Net, auctioning) can be specified in our WS-Agreement extended language. Finally we provided an overview of our initial attempts at implementing agent web service agreements over IBM's ETTK toolkit.

References

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

[2] F. for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*. http://www.fipa.org, 2002.

[3] Globus Alliance and IBM Software Development and Research Labs. *WS-Resource Framework*. `http://www.globus.org/wsrf/`.

[4] IBM Software Development and Research Labs. *Emerging Technologies Toolkit: ETTK*. `http://www.alphaworks.com/tech/ettk`.

[5] H. Ludwig, A. Dan, and R. Kearney. Cremona:an architecture and library for creation and monitoring of ws-agreements. In *Proc. of 2nd International Conference on Service Oriented Computing 04, ACM*, pages 65 – 74, 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] J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, 1969.

[9] 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/`.