An SLA Re-negotiation Protocol

Michael Parkin[1] Peer Hasselmeyer[2] Bastian Koller[3] Philipp Wieder[4]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
[2] NEC Laboratories Europe, Sankt Augustin, Germany
[3] High Performance Computing Centre (HLRS), Stuttgart, Germany
[4] Dortmund University of Technology, IT & Media Center, Germany

Abstract. Service Level Agreements (SLAs) are an essential foundation for the realisation of Business Grids as they provide a mechanism for a service provider to charge a customer for meeting an agreed quality of service. However, once an SLA has been formed it may need to be re-negotiated as the requirements of the SLA participants change. This paper describes an abstract, domain-independent protocol for the re-negotiation of an agreement, including SLAs formed using the WS-Agreement standard. The protocol is based on the principles of contract law to make the new agreements formed using it legally-compliant. It allows for multi-round re-negotiation in a network environment where messages may be lost, delayed, duplicated and re-ordered.

1 Introduction

Future distributed systems will be based on a model where individually defined, autonomous business or technical functions are offered by various independent providers in the form of remote services. The service abstraction fosters the migration towards a global service marketplace where the relationships between its participants will be governed by electronic contracts in the form of Service Level Agreements (SLAs). Agreement formation will be the basis for service value exchange between service customers and service providers. As a step towards this goal, the Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) of the Open Grid Forum (OGF) has produced the Web Services Agreement [1] (WS-Agreement) standard to create bilateral agreements. However, WS-Agreement is acknowledged to be limited in scope to a basic accept/reject agreement protocol [11] and the debate within the GRAAP-WG has moved to how *re-negotiation* of an existing agreement can be carried out to allow either party in the agreement to adapt the current agreement with the explicit consent of the other party [12]. In this paper we propose such a protocol that is independent of its application domain and the implementation technology.

1.1 What is Re-negotiation?

Re-negotiation is a second or further negotiation that may change the terms of an existing agreement[5]. A successful re-negotiation invalidates an existing agreement and replaces it with a new, superseding agreement. Re-negotiation may be

[5] Source: Oxford English Dictionary, Second Edition 1989.

required by either party if their circumstances change since the initial agreement was established. For example, a user may need to change the amount of CPU time they originally agreed to purchase if they find their computational job is more complex than they originally thought and it requires more CPU time to fully complete. An example where a resource provider may want to change an existing agreement is when there is an unexpected unavailability of resources they originally promised to supply. Re-negotiation can occur before the agreed service period of an SLA commences or while the SLA is active.

This paper presents an abstract, domain-independent re-negotiation protocol that allows the re-negotiation of agreements abstracted as *contracts* (i.e. a binding exchange of promises) between a service provider and a customer. The protocol allows the provider as well as the customer to initiate a multi-round re-negotiation and both to refuse a re-negotiation of an existing agreement. Re-negotiation can happen fully automatic or with manual help. The protocol is designed to support the re-engotiation process, but it does not mandate the use of a particular process. Although the protocol deals with the interaction of exactly one provider and one customer, multiple instances of this protocol, potentially involving the same providers and customers, can and are expected to proceed in parallel. Where it is possible to map this new protocol to the existing WS-Agreement protocol it is noted in the text in order to allow existing implementations of WS-Agreement to be extended.

1.2 Document Structure

The remainder of this document is structured as follows: in Section 2 we describe our approach to designing a protocol for the re-negotiation of contracts and compare it against other approaches. Section 3 then outlines the requirements (and non-requirements) for the protocol. Section 4 describes the assumptions made and the framework within which a re-negotiation occurs. Section 5 then specifies the protocol. The paper concludes with Section 6, which provides a summary and discusses future work.

2 Approach

2.1 Network Assumptions

Work on standardising open protocols for use in service-based environments is being carried out by several standards bodies, including the Organization for the Advancement of Structured Information Standards (OASIS), the World Wide Web Consortium (W3C) and the OGF. However, we feel the approaches to specifying protocols by these standards bodies often do not take into account factors inherent in distributed computing. For example, in distributed systems where "failure happens all the time" [4], it should be assumed that messages sent between services may be lost, delayed, duplicated and/or re-ordered. Most protocol specifications often do not take these circumstances into account – our work is different in that we consider these possibilities from the very start. Assuming such a network during the design phase makes the resulting protocol more robust as it can cope with network failures or malfunctions.

2.2 Protocol Description

Another difference to existing work is how we describe the protocol; we do not use UML sequence diagrams to describe the protocol as they often cannot capture the entire set of message exchanges possible in the network environment described above. To describe the behaviour of each re-negotiation participant we provide a finite state machine to describe the state of the contract, similar to the WS-Agreement specification. However, unlike in WS-Agreement, the state machine is not shared between the negotiation participants. Instead each participant has their own 'copy' of the state machine which they update as they send and receive messages. In addition, we also explain the possible messaging events using pre- and post-constraints that together specify the conditions which must be satisfied before and after each message is sent. The conditions explain each messaging event as an atomic action and together describe the messaging behaviour of each participant in the re-negotiation protocol.

2.3 Consistency & Brewer's Conjecture

Because messages may be lost, duplicated and/or re-ordered whilst being sent the state machines of the participants may be inconsistent at some points in time. This 'looser consistency' is a consequence of Brewer's Conjecture [3] where, in a distributed system, "there are three properties that are commonly desired: consistency, availability and partition tolerance". Brewer's conjecture is that "it is impossible to achieve all three [of these properties at once]" and that a distributed system can only demonstrate two of these three properties at the same time. Thus, in our scenario where the customer and provider are partitioned by a network and the resources (within the provider's administrative domain) should be available to everyone if they are not booked we loosen the property of consistency, i.e. we cannot ensure that both copies of the state machine (representing the contract) are in a consistent state.

Looser consistency may seem like a problem, but the protocol we present is designed to guarantee consistency will eventually be reached through the ability to re-send messages, as we will show in Section 5.3. As [8] discusses, by not attempting to have "increased fidelity" (i.e. strong consistency), we can also reduce the cost (in terms of time and resources) to implement the protocol.

The issue of consistency is *the* main difference between other agreement protocol designs and the one we advocate here: e.g. the GRAAP-WG attempts to maintain strong consistency of the contract's state using a two-phase commit (2PC) protocol. However, as a consequence of Brewer's conjecture, these protocols make the resources *unavailable* even if they have not been booked as they enter a state between 'not-booked' and 'booked' (a 'limbo' state). It is for these reasons that 2PC-style protocols have been described as "anti-availability protocols" [9]. This, as we describe in [13], is unacceptable to a resource provider and the reason why we advocate loosening the consistency requirement.

3 Protocol Requirements

This section describes the requirements for a re-negotiation protocol taken from the GRAAP-WG Wiki page [5] and associated usage scenarios [6].

3.1 Non-Requirements

Before proceeding it should be noted that many of the requirements given by the GRAAP-WG are not actual requirements for a re-negotiation protocol. In this paper a protocol is defined as *the semantics of the messages exchanged and the allowed sequences of message exchange*. This is referred to as services sharing a *schema* (the allowed messages) and a *contract*[6] which "describes [the] message sequences allowed in and out of the service" [7].

Thus, reviewing the requirements from the GRAAP-WG [5], requirements about 're-negotiable and service description terms', 'clearer information about why parties do not agree' and 'how to re-negotiate the expiry of an agreement' can all be seen as domain or agreement-specific information. Two of the three protocol usage scenarios (about reserving more resources and extending the agreement expiration time) also describe requirements for the information exchanged by the protocol. This domain and agreement-specific information has little to do with the protocol (as defined above) used for re-negotiation because, in order to keep the protocol domain-independent, the content of messages should be orthogonal to the protocol and the reasons *why* messages are exchanged (as this is dependent on the re-negotiation strategy of each participant).

The final requirement for re-negotiation discussed by the GRAAP-WG is how contracts are 'versioned', e.g. through the issuing of a new agreement identifier to the superseding contract. This topic is not covered in this paper as, again, this process is orthogonal to the protocol used to re-negotiate the contract.

3.2 Re-negotiation Requirements

The remaining requirements from the GRAAP-WG Wiki fall into the category of who can initiate re-negotiation and how re-negotiation is initiated. For example, a protocol usage scenario requires that the contract can be cancelled through "asking for releasing resources which had been agreed upon". We believe that both parties in the contract should be allowed to carry out the initiation of re-negotiation and that both parties should be allowed to cancel an existing contract[7] as this is what is allowed in the 'real world', after all. We also believe that the initiation of re-negotiation should be allowed through non-binding enquiries to the other party so that an estimate as to how much it would cost to change the contract can be obtained before committing to a new contract. These requirements are met by the final protocol design.

3.3 Contract Law Requirements

When designing this protocol we also feel it is necessary to take into account the legal requirements of contract formation. This is because, as we describe in [13], such an approach benefits businesses and their customers; a re-negotiation protocol meeting the requirements of contract law means both parties can be confident

[6] Note that the service contract is not to be confused with the contract being re-negotiated.

[7] Because of space limitations the ability to cancel contracts is not included in this protocol specification though this protocol can be extended to provide this behaviour with little extra effort.

that agreements they make for providing and receiving Grid services are robust and can be taken to litigation if any dispute arises. Contracts form part of the foundations of commerce and, with the advent of business-oriented Grids such as BREIN [2], aligning standard protocols with common business practices and rules can only increase their uptake.

Thus, the requirements of contract law (as described in [13]) have been included in the protocol design. Briefly, some of these requirements are that *offer* messages are binding if accepted, all offers are acknowledged and, because of the risk of 'cheating' by the customer, we use contract law's 'mailbox rule' where a contract is formed when the accept message is sent by the offeree (the resource provider) and not when the accept is received by the offeror (the customer).

4 Protocol Design

4.1 Protocol Roles

In order to begin deriving a protocol for re-negotiating a contract, the roles that each participant plays in the protocol must be clarified. In this work we define two roles: the resource provider and the customer[8]. In our opinion defining these roles, rather than abstract 'agreement initiator/responder' roles, highlights the natural asymmetry of resource provision from resource consumption. As we will show in Section 5, by explicitly considering the requirements of the service provider, we can also remove the possibility of a denial-of-service attack on them.

4.2 Protocol Framework & Contract State Machine

Before defining the protocol we should note that the re-negotiation takes place in the context of an existing contract for the provision of services or resources. This contract has some unique identifier that is known to both parties. In an implementation of WS-Agreement, for example, this identifier is the Endpoint Reference (EPR) of a WS-Agreement.

Thus, before re-negotiation is initiated the contract at the customer and provider is in the `contracted` state[9] to reflect this existing context. When re-negotiation is initiated by one party their contract enters the `renegotiating` state, which is a sub-state of `contracted` as the original contract is still in force, irrespective of the on-going re-negotiation. After successful re-negotiation the current contract both parties' contract is in the `superseded` state as a new contract will have superseded this contract.

Figure 1 shows this behaviour as a finite state machine for the contract (as described in Section 2 each participant has a local copy of this state machine). Again, the re-negotiation takes place inside the `contracted` state to reflect an existing contract is being re-negotiated. Initially the contract enters the `contracted` state through transition 1 (how this original agreement formed is outside the scope

[8] This can be mapped onto WS-Agreement's concepts of agreement initiator and responder with the customer playing the role of agreement initiator and the resource provider that of the agreement responder.

[9] This state is equivalent to WS-Agreement's 'observed' state, but the word 'contracted' is used as it is felt that this indicates more accurately what has occurred.

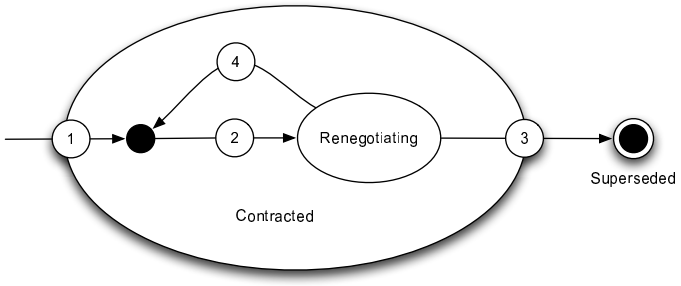

Fig. 1. Re-negotiation State Machine

of this paper, though it may be through an instance of the WS-Agreement protocol). When re-negotiation is initiated, the contract makes transition 2. If the re-negotiation is successful, transition 3 to the `superseded` state occurs. If the re-negotiation is unsuccessful, transition 4 back to the `contracted` state occurs.

It may be that one of the parties cannot or does not want to re-negotiate the contract. In this case they may indicate to the other party that they cannot re-negotiate (how they do this is described in Section 5). If the contract is in the `contracted` state, it remains in this state. However, if the contract is being re-negotiated when either party decides they cannot re-negotiate, transition 4 back to the `contracted` state occurs.

5 Protocol Specification

In Section 3 we defined a protocol to be *the semantics of the messages exchanged and the allowed sequences of message exchange*. Thus, our protocol specification is split into two sections to meet this definition. The first section defines the protocol messages and their semantics, whilst the second defines the allowed messaging behaviours of the resource provider and the customer.

5.1 Protocol Messages

The following is a list of the protocol messages derived from Section 3. Note that each message, in addition to the domain-specific information being exchanged, has three identifiers: one, the agreement identifier that provides a context for messages to be correlated under; two, a message identifier that is unique in the context of each agreement; and, three, a correlation identifier which should be set to the message identifier of the message (if any) this message has been sent in reply to (thus a correlation identifier may be *null* if it does not relate to any other messages in the re-negotiation).

- *RenegotiationQuoteRequest*. This message is only sent from the customer to the resource provider to indicate that they are interested in re-negotiating the current contract and to ask for a quote for the re-negotiated contract.
- *RenegotiationQuote* is a message only sent from the resource provider to the customer and is a non-binding estimate of a new agreement. It may be sent in

response to a *RenegotiationQuoteRequest* received from the customer, in which case the quote will be based on the new requirements of the customer, or it may be sent by the service provider unilaterally to indicate the provider wishes to re-negotiate the agreement. In the latter case the quote will be based on the provider's requirements.

- *RenegotiationOffer*. This message is sent from the customer to the provider as a binding request to form a new agreement. The customer must be careful when sending this message as, if it is accepted, a new superseding contract will be formed on the contents of this message. Sending this message can be seen as the equivalent of WS-Agreement's agreement initiator role invoking the WSAG:CreateAgreement operation.
  As discussed in [13], by specifying that only the customer makes offers to the provider we remove the possibility of denial-of-service attacks on the provider (i.e. situations where the resource provider is in a 'limbo' state, waiting on a message from the customer to confirm the booking of resources).
- *RenegotiationOfferAck*. This message is sent from the provider to the customer to acknowledge that an offer has been received and is being considered by the provider. The inclusion of this message satisfies the requirement of the EU's eCommerce directive, as we have described in [13].
- *RenegotiationAccept*. This message is sent from the provider to the customer to indicate that a *RenegotiationOffer* has been accepted and a new contract has replaced the original contract[10]. Sending this message is equivalent to WS-Agreement's agreement responder role invoking the WSAG:AcceptAgreement operation.
- *RenegotiationReject*. Only sent from the resource provider to the customer, this message indicates that a *RenegotiationOffer* was not accepted and will no longer be considered by the resource provider. Sending this message is equivalent to a WS-Agreement *fault* message being sent.
- *RenegotiationNotPossible* is a message that can be sent by either party (when allowed) to indicate that the re-negotiation of a contract is not (or is no longer) possible. This may be because, for example, the resources allocated in the current contract (i.e. the contract being re-negotiated) have expired. This may also be mapped onto a type of WS-Agreement *fault* message.

5.2 Protocol Behaviours

Safety Properties We assume that the provider and customer communicate by sending messages asynchronously using a non-Byzantine model[11]. Messages can take arbitrarily long to be delivered and may be duplicated and/or lost but not corrupted. We also define what are called *safety properties* for the protocol [10]. The safety properties for this protocol are that:

- Only an offer that has been sent by the customer can be accepted.

[10] As we described in Section 3.3, the new agreement is formed when the accept message is sent because we invoke contract law's 'mailbox rule'.

[11] Byzantine behaviour is where a protocol participant not only does not follow the prescribed messaging behaviour but also my fail to behave consistently.

- Only one offer can be accepted in an instance of the re-negotiation protocol.
- An offer which has been rejected by the provider cannot be accepted later.
- The acceptance of an offer by the provider renders all other outstanding offers within the instance of the re-negotiation protocol invalid (i.e. they are *revoked*).
- A *RenegotiationNotPossible* message sent from the provider to the customer means that all outstanding offers have been revoked.

Safety properties are protocol behaviours that cannot be broken. If the safety properties are broken then one of the protocol participants has exhibited a fault of some kind.

Customer Behaviour

- Send *RenegotiationQuoteRequest*.
  - Pre-condition: The customer's contract must not be in the `superseded` state.
  - Post-condition: The customer's contract remains in its current state.
- Receive *RenegotiationQuote*.
  - Pre-condition: There is no pre-condition to this event occurring as it can take place at any time, including after the customer's contract has entered the `superseded` state (as it may be a message delayed from earlier in the re-negotiation). A provider can initiate re-negotiation by sending this message without prior receipt of a *RenegotiationQuoteRequest*.
  - Post-condition: If the customer's contract is in the `superseded` state then it must remain in this state and no messages are being sent in response. If the customer is in any other state they may choose to ignore this message and remain in their current state or they may choose to reply with a *RenegotiationQuoteRequest* or *RenegotiationOffer*.
- Send *RenegotiationOffer*.
  - Pre-condition: The customer's contract must not be in the `superseded` state[12].
  - Post-condition: The customer's contract is in the `renegotiating` state.
- Receive *RenegotiationOfferAck*.
  - Pre-condition: The customer must have sent a *RenegotiationOffer* matching the correlation identifier in the *RenegotiationOfferAck*.
  - Post-condition: The customer should remain in its current state.
- Receive *RenegotiationAccept*.

[12] Note that within the protocol, there is the capability for a customer to make multiple offers to the provider, i.e. to send more than one. This situation can come about if, for example, the customer sends an offer and then another offer before it receives a response from the provider. The provider may then receive two offers from the customer in close succession. This is not a problem because the safety properties of the protocol ensure that only one offer can be accepted and all other offers become invalid when it is accepted.

  - Pre-condition: The customer must have sent a *RenegotiationOffer* with a message identifier identical to the correlation identifier in the *RenegotiationAccept* message. This message may be received at any time after the *RenegotiationOffer* was sent, including after the contract has entered the `superseded` state as duplicates of messages may be received.
  - Post-condition: The customer's current contract is in the `superseded` state. No further messages should be sent in this instance of the re-negotiation protocol.
- Receive *RenegotiationReject*.
  - Pre-condition: The customer must have sent a *RenegotiationOffer* with a message identifier the same as the correlation identifier in the *RenegotiationReject* message.
  - Post-condition: The customer's contract remains in the current state or, if it is in the `renegotiating` state and there are no outstanding *RenegotiationOffer* messages it moves to the `contracted` state.
- Send *RenegotiationNotPossible*.
  - Pre-condition: The customer's contract must be in the `contracted` state.
  - Post-condition: The customer's contract must be in the `contracted` state.
- Receive *RenegotiationNotPossible*.
  - Pre-condition: This event may take place at any time, including after the customer's contract has entered the `superseded` state as this may be a message delayed or duplicated from earlier in the re-negotiation.
  - Post-condition: If the customer's contract is in the `superseded` state it should remain in this state and no messages sent in response. Otherwise, the customer's contract moves to the `contracted` state.

Resource Provider Behaviour

- Receive *RenegotiationQuoteRequest*.
  - Pre-condition: There is no pre-condition to this event occurring as it can take place at any time, including after the contract has entered the `superseded` state as this may be a message from the customer delayed from earlier in the re-negotiation.
  - Post-condition: The provider's contract remains in its current state. If the provider's contract is in the `superseded` state then the *RenegotiationAccept* message originally sent to agree the superseded contract must be resent to indicate the state of the contract. In the otehr states, the provider may consider sending a *RenegotiationQuote* message.
- Send *RenegotiationQuote*.
  - Pre-condition: The provider's contract must not be in the `superseded` state.
  - Post-condition: The provider's contract remains in its current state.
- Receive *RenegotiationOffer*.
  - Pre-condition: There is no pre-condition to this event occurring as it can take place at any time, including after the contract has entered the `superseded` state as this may be a message from the customer delayed from earlier in the re-negotiation.

  - Post-condition: If the provider's contract is in the `superseded` state a *RenegotiationAccept* message must be sent with the correlation identifier matching the identifier of the previously accepted *RenegotiationOffer*. Otherwise a *RenegotiationOfferAck* must be sent with the correlation identifier matching the id of the *RenegotiationOffer* message received. If a duplicate *RenegotiationOffer* is received the same *RenegotiationOfferAck* message must be resent. If the duplicate offer had been rejected previously, the same *RenegotiationReject* message must be resent as well.
- Send *RenegotiationOfferAck*.
  - Pre-condition: The provider must have received a *RenegotiationOffer*.
  - Post-condition: The provider's contract is in the `renegotiating` state.
- Send *RenegotiationAccept*.
  - Pre-condition: The provider must have sent a *RenegotiationOfferAck*. The correlation id of the *RenegotiationAccept* message being sent must be identical to the message id of the *RenegotiationOffer* that is being accepted.
  - Post-condition: The provider's current contract is in the `superseded` state. All outstanding offers made by the customer are marked as revoked. The newly established contract is in the `contracted` state.
- Send *RenegotiationReject*.
  - Pre-condition: The provider must have sent a *RenegotiationOfferAck*. The correlation identifier of the *RenegotiationReject* message being sent must be identical to the message identifier of the *RenegotiationOffer* that is being rejected.
  - Post-condition: The provider's contract moves to the `contracted` state unless there are outstanding *RenegotiationOffer* messages, in which case it remains in the `renegotiating` state.
- Send *RenegotiationNotPossible*.
  - Pre-condition: The provider's contract must be in the `contracted` or `renegotiating` state.
  - Post-condition: The provider's contract is in the `contracted` state.
- Receive *RenegotiationNotPossible*.
  - Pre-condition: This event may take place at any time, including after the provider's contract has entered the `superseded` state as this may be a message (possibly a duplicate) delayed from earlier in the re-negotiation.
  - Post-condition: If the provider's contract is in the `superseded` state it must remain in this state and the original *RenegotiationAccept* message must be resent. Otherwise, the provider sends out *RenegotiationReject* messages to all outstanding offers and moves to the `contracted` state.

5.3 Handling Inconsistencies

As described in Section 2, this protocol relaxes the requirement for consistency of the contract state across the customer and resource provider. In doing so we gain the availability of resources for booking as they are never in a state between 'not-booked' and 'booked', unlike in a protocol based on a transactional approach in which resources need to be reserved until a transaction completes or aborts. This section describes how inconsistencies between the customer and provider are handled through the protocol, illustrated through two example message exchanges.

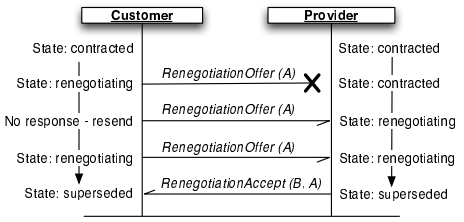

Fig. 2. Lost, duplicated RenegotiationOffer

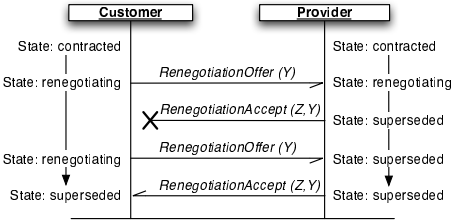

Fig. 3. Lost RenegotiationAccept

Example 1 With both the customer and the resource provider in the `contracted` state, the customer sends a *RenegotiationOffer* offer with identifier A to the provider and moves to the `renegotiating` state. Unfortunately, this message is lost on the network, thus the two parties have inconsistent states. This scenario is shown in Figure 2. Upon not receiving a reply to their *RenegotiationOffer*, the customer may keep resending the same message until they receive a response from the resource provider. Resending a *RenegotiationOffer* is not a problem as it can only be accepted or rejected once. Thus, the customer can be confident that they will not be contracted multiple times. In Figure 2, the provider returns a *RenegotiationAccept* with an identifier B and a correlation identifier A to indicate which offer is being accepted.

Following the receipt of the *RenegotiationAccept* the customer will be in the same state as the provider when the provider sent the messages[13].

Example 2 Figure 3 shows the customer and the provider in the `renegotiating` state after the customer sends *RenegotiationOffer* with identifier Y. The provider sends a *RenegotiationAccept* with identifier Z and correlation identifier Y in response. The accept message is lost and the customer and the resource provider are in inconsistent states. As in Example 1, the customer may keep resending the *RenegotiationOffer* until it receives a response.

6 Summary, Conclusions & Future Work

This paper has described an application-level protocol between a service customer and a provider that meets the requirements of the OGF's GRAAP-WG for renegotiating existing agreements, which may be SLAs. The protocol is based on the principles of contract law and assumes an imperfect message transmission layer so it is designed to behave correctly and reliably even in the presence of network faults that lead to message loss, delay and duplication. The protocol is independent of the contents of the contract the messages relate to and is therefore usable in various application domains that need re-negotiation capabilities, such as when SLAs for Grid services need to have their original terms changed.

[13] Note that we do not say that the customer and resource provider *will* be in the same state as the provider may have changed state in the time between sending the reply and the customer receiving it.

Whilst discussing this protocol with our peers they have commented that this protocol is "too complicated to implement". To show that it is not we have implemented this protocol and fully tested it to ensure it meets the specification. The heart of the protocol (the state machine in Figure 1) was implemented in around 90 lines of Ruby and a description of this implementation will be the subject of a future paper.

7 Acknowledgements

The authors would like to express their thanks to members of OGF's GRAAP-WG who provided feedback on an earlier draft of this paper and in particular Karl Czajkowski of UNIVA for his comments. Michael Parkin is pleased to acknowledge that this work was carried out as part of an industrial fellowship for the CoreGRID IST project N°004265, funded by the European Commission and partly sponsored by ATOS Origin Research and Innovation Spain. Bastian Koller's work is part of the BREIN project [2], partly funded by the European Commission's IST activity of the 6^{th} Framework Programme, contract number 034556. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

1. A. Andrieux *et. al.* Web Services Agreement Specification (WS-Agreement). Proposed Recommendation, Grid Resource Allocation Agreement Protocol Working Group, Open Grid Forum. September 2006.
2. Business Objective Driven Reliable and Intelligent Grids for Real Business (BREIN). EC FP6 Integrated Project. http://www.gridsforbusiness.eu/.
3. S. Gilbert, N. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. ACM SIGACT News, 33(2):51–59. June 2002.
4. Google, Inc. Introduction to Distributed System Design. Google Code for Educators Tutorial. October 2007.
5. T. Nakata *et. al.* ReNegotiationWishlists, Open Grid Forum GridForge Wiki. September 2007.
6. T. Nakata *et. al.* Usage Scenarios to be Included in the Specification. Open Grid Forum GridForge Wiki. September 2007.
7. P. Helland. Data on the Outside Versus Data on the Inside. Proceedings of the Second Biennial Conference on Innovative Data Systems Research, pages 114–153. January 2005.
8. P. Helland. Memories, Guesses, and Apologies. MSDN Blog Article. May 2007.
9. P. Helland. SOA and Newton's Universe. MSDN Blog Article. May 2007.
10. L. Lamport. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison Wesley Professional, 2003.
11. H. Ludwig, T. Nakata, P. Wieder, O. Wäldrich. Reliable Orchestration of Resources using WS-Agreement. CoreGRID Technical Report TR-0050. October 2006.
12. T. Nakata. Thoughts on Negotiation. Grid Resource Allocation Agreement Protocol Working Group Presentation, Open Grid Forum 19. September 2006.
13. M. Parkin, D. Kuo, J. Brooke, A. MacCulloch. Challenges in EU Grid Contracts. Proceedings of the 4th eChallenges Conference, pages 67–75. October 2006.