A Contract Re-negotiation Protocol

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

[1]Barcelona Supercomputing Centre, Universitat Politècnica de Catalunya, Spain
[2]NEC Laboratories Europe, St. Augustin, Germany
[3]High Performance Computing Centre (HLRS), Stuttgart, Germany
[4]Technical University of Dortmund, Dortmund, Germany

Abstract

This document describes an abstract, domain-independent protocol for the re-negotiation of contracts. This protocol is based on the principles of contract law to make agreements with it legally-compliant and allows for multi-round re-negotiation in an environment where messages may be lost, delayed and re-ordered.

1 Introduction

Contract formation between service-based computational applications and their users is required to enable a vision of computing where, for example, applications can be constructed from individual pieces of software exposed as services. These agreements can promote the flexible and dynamic re-use of interoperable, encapsulated functions in a Service Oriented Architecture (SOA). The Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) of the Open Grid Forum (OGF) has produced the WS-Agreement [1] standard to create these agreements. However, it is acknowledged to be limited in scope to a basic accept/reject agreement protocol [2]. Thus, the debate within the GRAAP-WG of OGF has moved to how *re-negotiation* of an existing agreement can be carried out [3]. This allows for either party in the agreement process to adapt the current agreement with the explicit consent of the other party as their circumstances change.

1.1 What is re-negotiation?

Re-negotiation is the process of negotiating an agreement again in order to change the originally agreed terms[1]. A successful re-negotiation invalidates an existing agreement and replaces it with a new, superseding agreement. However, many aspects of what is normally thought of as 're-negotiation' are considered out-of-scope within the GRAAP-WG's discussion. For example, considering what happens when one of the parties refuses to re-negotiate a new contract, who should be allowed to initiate a re-negotiation and how to provide multi-round re-negotiation are not covered by the GRAAP-WG discussion [3].

As a consequence of these limitations, this paper presents an abstract, domain-independent re-negotiation protocol to allow the re-negotiation of contracts between a service provider and a customer. The protocol presented here meets the requirements of allowing both parties to initiate a multi-round re-negotiation and either to refuse a re-negotiation. Where it is possible to map this new protocol to the WS-Agreement protocol this is noted in the text in order that implementations of WS-Agreement may be extended to allow this behaviour.

1.2 Document Structure

The remainder of this document is structured as follows: Section 2 describes our approach to creating a protocol for the re-negotiation of contracts and compares against other approaches; Section 3 describes the requirements for the protocol; Section 4 describes the assumptions made and the framework in which a re-negotiation occurs; Section 5 describes the protocol; Section 6 provides a conclusion and discusses future work.

2 Approach

At the time of writing, work on standardising open protocols for use in SOA environments is being carried out by several standards bodies, including the OASIS the W3C and OGF. For example, the GRAAP-WG of the OGF has recently

[1]Source: Oxford American Dictionary.

produced the WS-Agreement standard which describes a standard protocol for agreement creation. However, we feel the approach to specifying protocols by these standards bodies does not take into account factors inherent in distributed computing and we have therefore taken a different approach to specifying protocols to these standards bodies. For example, we assume a networking environment between a service provider and their customer where messages may be lost, delayed, duplicated and/or re-ordered once sent. This is because in distributed systems "failure happens all the time" [4]. Assuming such a network when designing a protocol makes the resulting protocol more robust as it can cope with network failures or malfunctions and it also promotes interoperability as it de-couples the protocol implementation from reliable-messaging middleware which mandate a specific reliable messaging platform.

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 do not capture the entire set of message exchanges possible in the network environment we assume. 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. Because messages may be lost, duplicated and/or re-ordered whilst being sent these state machines may be inconsistent at some points in time. This 'loose consistency' is a consequence of Brewer's Conjecture [5] where, in a distributed system, "there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three." Therefore, in our scenario where the customer and provider are partitioned by a network and the resources, within the providers administrative domain, should be available to everyone if they are not booked, we cannot ensure that the customer's and the provider's copy of the state machine (representing the contract) are in a consistent state.

Loose consistency may seem like a problem, but the protocol, as we will show in Section 5.3, can be designed to guarantee that consistency will be reached eventually through the ability to re-send messages. As [6] discusses, by not attempting to have "increased fidelity" (i.e. strong consistency), we can also reduce the cost, both regarding time and resources required, to implement the protocol.

The issue of consistency is *the* main difference between the GRAAP-WGs approach to protocol design and the one advocated here: the GRAAP-WG attempts to maintain strong consistency of state using transactional protocols using a two-phase commit (2PC) approach. As a consequence these protocols make the resources being negotiated *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" [7]. This, as we describe in [8], is unacceptable for a resource provider and the reason why we advocate loosening the consistency requirement.

As well as providing a finite state machine to describe the protocol, we also explain the possible messaging events using a unique notation of pre- and post-constraints (or conditions). These describe each messaging event as an atomic action and explain the messaging behaviour of each re-negotiation participant.

3 Protocol Requirements

This section describes the requirements for a re-negotiation protocol from the GRAAP-WG Wiki page [9] and associated usage scenarios [10].

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 the re-negotiation protocol. In this work 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 *schema* and *contract*, the schema being the allowed messages, whilst the contract "describes message sequences allowed in and out of the service" [11][2].

Thus, reviewing the requirements from the GRAAP-WG Wiki [9], the re-negotiation requirements regarding 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 which should be contained in the messages exchanged. Two of the three protocol usage scenarios also describe requirements for the information (about reserving more resources and extending the agreement

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

expiration time) exchanged by the protocol. This information has little to do with the protocol used for re-negotiation because, in order to keep the protocol domain-independent, the content of messages should be orthogonal to the protocol itself and the reasons *why* messages are exchanged (as this is dependent on the strategy of each participant).

The final requirement for re-negotiation discussed by 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[3]. 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 in the final protocol design.

3.3 Contract Law Requirements

When designing this protocol we also feel is it necessary to take into account the legal requirements of contract formation. This is because, as we describe in [8], taking this approach benefits businesses and their customers; a re-negotiation protocol meeting the requirements of contract law means both parties can be confident 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 (e.g. BREIN [12]) aligning standard protocols with common business practices and rules, we feel, can only increase their uptake.

Thus, the requirements of contract law (as described in [8]) have been included in the protocol design. Briefly, some of these requirements are that *offer* messages are binding if accepted, that all offers are acknowledged and, because of the risk of 'cheating' by the customer, we invoke contract law's 'mailbox rule' where a contract is formed when the accept message is sent by the offeree (i.e. the resource provider) and not when the accept is received by the offeror (i.e. 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[4]. 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 EPR of a WS-Agreement.

Thus, before re-negotiation is initiated the contract is in the contracted state[5] to reflect this existing context. When re-negotiation is initiated the contract enters the re-negotiating 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

[3] 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.

[4] 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.

[5] 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.

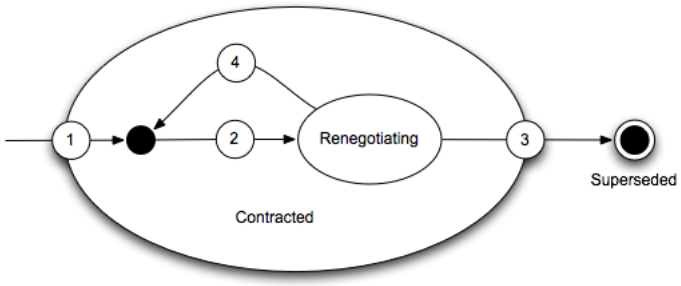

Figure 1. Re-negotiation State Machine

current contract is in the superseded state as a new contract will have superseded this contract.

This behaviour is shown in Figure 1, which shows the finite state machine for the contract. 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[6]. 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, they remain 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 happens.

5 Protocol Definition

Taking inspiration from [11], the protocol definition is split into two sections. The first defines the protocol messages and their semantics, whilst the second defines the allowed messaging behaviours of the resource provider and 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 based on the old agreement and the new requirements of the customer. This message may be sent in response to a re-negotiation quote, or it may be used by the service provider to initiate re-negotiation unilaterally.

- *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 we have discussed in a previous paper [8] 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 acknowl-

[6] How this agreement formed is outside the scope of this paper.

edge 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 described in [8].

- *RenegotiationAccept*. This message is sent from the provider to the customer to indicate that a *RenegotiationOffer* has been accepted and a new contract formed that replaces the original contract[7]. 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 be mapped onto a type of WS-Agreement *fault* message.

5.2 Protocol Behaviours

5.2.1 Safety Properties

As well as assuming that the provider and customer communicate by sending messages asynchronously, using a non-Byzantine model[8] where messages can take arbitrarily long to be delivered and be duplicated or lost but not corrupted, we also define what are called *safety properties* for the protocol [13]. 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. The safety properties for this protocol are that:

- Only an offer that has been made can be accepted.
- Only one offer can be accepted in an instance of the re-negotiation protocol.
- The acceptance of an offer revokes (i.e. makes invalid) all other outstanding offers within the instance of the re-negotiation protocol.
- The receipt of a *RenegotiationNotPossible* message by the customer means that all all outstanding offers have been invalidated.

5.2.2 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* message.
  - Post-condition: If the customer's contract is in the superseded state then it must remain in this state and no message should be sent in return. 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[9].
  - Post-condition: The customer's contract is in the re-negotiating state.

[7] 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'.

[8] Byzantine behaviour is where a participant of the protocol not only does not follow the prescribed messaging behaviour but also of fails to behave consistently when interacting with other protocol participants.

[9] 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 (either a *RenegotiationAccept* or *RenegotiationReject*) from the provider. The provider may then receive two offers from the customer in close succession. This may seem like a problem, but the safety properties of the protocol ensure that only one offer can be accepted and all other offers become invalid when it is accepted.

- Receive *RenegotiationOfferAck*.
  - Pre-condition: The customer must have sent a *RenegotiationOffer* matching the correlation iddentifier in the *RenegotiationOfferAck*.
  - Post-condition: The customer should remain in its current state.
- Receive *RenegotiationAccept*.
  - 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 re-negotiating state and there are no outstanding *RenegotiationOffer* messages it can move 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 should move to the contracted state.

5.2.3 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 sent to agree the superseded contract must be resent to indicate the state of the contract.
- 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 id matching the id of the previously accepted *RenegotiationOffer*. Otherwise a *RenegotiationOfferAck* must be sent with the correlation id 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 re-negotiating 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 correlation id of the *RenegotiationOfferAck* that was sent.
  - Post-condition: The provider's current contract is in the superseded state. The newly established contract is in the contracted state.

- Send *RenegotiationReject*.
  - Pre-condition: The provider must have sent a *RenegotiationOfferAck*. The correlation id of the *RenegotiationReject* message being sent must be identical to the correlation id of the *RenegotiationOfferAck* that was sent.
  - Post-condition: The provider's contract moves to the contracted state unless there are outstanding *RenegotiationOffer* messages, in which case it remains in the re-negotiating state.

- Send *RenegotiationNotPossible*.
  - Pre-condition: The provider's must be in the contracted 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.

5.3.1 Example 1

With both the customer and resource provider in the contracted state, the customer sends a *RenegotiationOffer* offer to the provider and moves to the re-negotiating state. This message is lost, thus the two parties have inconsistent states. Here, 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.

Following the receipt of the response from the provider the customer will be in the same state as the provider when the provider sent the message[10].

5.3.2 Example 2

The customer and provider are in the re-negotiating state and the provider sends a *RenegotiationAccept* in response to a *RenegotiationOffer*. The accept message is lost and the customer and 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.

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

The protocol is based on the assumptions of an imperfect message transmission layer and is designed to behave correctly and reliably even in the presence of network faults that lead to message loss and duplication. The protocol is independent of the contents of the contract the messages relate to. It is therefore usable in various application domains that need re-negotiation capabilities.

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). The heart of the protocol (the state machine in Figure 1) was implemented in around 120 lines of Ruby. A description of the implementation is the subject of a companion paper.

7 Acknowledgements

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.

Peer Hasselmyer's work has been supported by the NextGRID project and has been partly funded by the European Commission's IST activity of the 6^{th} Framework Programme under contract number 511563.

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, Open Grid Forum, September 2006. Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG).

[2] H. Ludwig, T. Nakata, P. Wieder, and O. Wäldrich. Reliable Orchestration of Resources using WS-Agreement. CoreGRID Technical Report Number TR-0050, October 2006.

[3] T. Nakata. Thoughts on Negotiation. OGF 19 GRAAP-WG Session Presentation, September 2006.

[4] Google, Inc. Introduction to Distributed System Design. Google Code for Educators Tutorial, October 2007.

[5] S. Gilbert and N. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *ACM SIGACT News*, 33(2):51–59, June 2002.

[6] P. Helland. Memories, Guesses, and Apologies. MSDN Blog Article, May 2007.

[7] P. Helland. SOA and Newton's Universe. MSDN Blog Article, May 2007.

[8] M. Parkin, D. Kuo, J.M. Brooke, and A. MacCulloch. Challenges in EU Grid Contracts. In P. Cunningham and M. Cunningham, editors, *Proceedings of the 4th eChallenges Conference: Exploiting the Knowledge Economy: Issues, Applications and Case Studies*, pages 67–75, October 2006.

[9] T. Nakata et. al. ReNegotiationWishlists. OGF GridForge Wiki, September 2007.

[10] T. Nakata et. al. Usage Scenarios to be Included in the Specification. OGF GridForge Wiki, September 2007.

[11] P. Helland. Data on the Outside Versus Data on the Inside. In M. Stonebreaker, G. Weikum, and D. DeWitt, editors, *In Proceedings of the Second Biennial Conference on Inovative Data Systems Research (CIDR 2005)*, pages 114–153, January 2005.

[12] Business Objective Driven Reliable and Intelligent Grids for Real Business (BREIN). EC FP6 Integrated Project. http://www.eu-brein.com/.

[13] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison Wesley Professional, 2003.