Chained Negotiation for Distributed Notification Services

Richard Lawley Michael Luck Luc Moreau

School of Electronics and Computer Science
University of Southampton
{ral01r, mml, L.Moreau}@ecs.soton.ac.uk

Abstract

Distributed notification services allow consumers and publishers of notifications to interact with different notification services. However, such a distributed infrastructure makes it difficult to share notifications between consumers when consumers are allowed to specify Quality of Service levels. In this paper, we present a chained negotiation engine, enabling distributed notification services to support negotiation and to reuse existing subscriptions. We demonstrate the benefit to the system as a whole by reducing load on service providers and enabling content to be shared.

1 Introduction

Notification services (NSs) are messaging middleware components providing asynchronous communications between services and/or users in a distributed environment. They are responsible for the delivery of messages between publishers and subscribers; publishers (such as information services) provide information that is then filtered and delivered to subscribed consumers [7, 8, 12] based on a specification of topic and delivery parameters.

Publishers and consumers of notifications may have different and conflicting requirements in terms of delivery parameters. For example, subscribers to a service may wish to filter the notifications based on a set of criteria, or specify minimum intervals between notifications. These are essentially measures of Quality of Service (QoS), and different subscribers may demand or request different levels of QoS. As the publisher may not be able to meet these requirements for reasons of policy or system load, a mechanism is required to resolve the difference between the preferences of the consumer and the provider.

In previous work [14], we determined that a competitive approach using negotiation would enable differences between the requirements of publisher and consumer to be resolved. A bi-lateral negotiation model was presented enabling mutually acceptable values for notification parameters to be automatically determined. We demonstrated that, by using this negotiation model to manage client demands, the load on the service provider could be reduced, enabling more clients to use the same service. The requirement to negotiate over QoS levels in a distributed service-oriented computing environment has also been recognised by the Grid community [1].

NSs can be distributed to address security and scalability issues. Here, multiple instances of a NS are hosted at different locations on the grid [13]. Publishers and consumers interact with different NSs; typically, they publish messages at or consume them from the NS located at their institution. Distributed NSs ensure that messages are routed between them to propagate notifications from publishers to the relevant consumers. If multiple consumers are subscribed to the same topic, the same notifications will be sent to all of them. When multiple consumers at the same site are receiving the same notifications, it becomes sensible to reuse the subscriptions in order to reduce network traffic and potential delays. However, if each consumer is allowed to specify different QoS parameters for a subscription, it may become difficult to share notifications.

A key aspect of distributed NSs is that intermediaries can exist between publishers and consumers in the form of other NSs. In order to negotiate quality of service between a publisher and a consumer, a more complex form of negotiation is required. Since the publisher and consumer no longer communicate directly, the negotiation has to take place through intermediaries, or middlemen. These middlemen pass on proposals or suggest existing commitments that can be reused to satisfy the consumer's requirements, which can impact on the negotiation outcome for the consumer or the publisher.

To enable negotiation over QoS in the context of a notification service and provide more efficient use

of notifications, we have designed an extended negotiation model to support *chained negotiation*. In this model, publishers and subscribers do not need to know whether they are negotiating directly or through a middleman — the interactions in which they participate remain identical in both cases. Negotiations take place between negotiation components built into publishers and consumers through a number of middlemen. The publisher is a NS publishing notifications on a topic the consumer is interested in. In situations in which the consumer is connected to a different notification service, chained negotiation is required to mediate the differing requirements of the two.

In this paper, we discuss the design and preliminary evaluation of a negotiation engine implementing chained negotiation. To demonstrate its effectiveness we have designed a series of experiments which aim to show that using a chained negotiation system in conjunction with a distributed notification service enables more consumers to be serviced by a given set of notification services, and that this can be done at the same time as reducing the load on service providers. We have already shown that negotiation can reduce the load on a service provider in a directly connected scenario — we aim to show improved results using a distributed system.

2 Notification Services

Over recent years, there has been a significant increase in the number of computers and other devices that access and run remote services over a network. Traditionally, this would have been accomplished with remote-procedure calls (RPC), where a remote service is invoked by a client that issues the requests and stays connected waiting for it to complete. This has turned out to be an unsuitable model as it has become a common requirement to be able to issue requests, disconnect, and reconnect again later to receive the outcome of the request. This is needed when a permanent connection is not available, or where a job is long running or continuous and it is not practical to stay connected while waiting for results.

Various message-oriented middleware solutions enable asynchronous, reliable communications and are suitable for the role of handling remote requests. Queuing products such as Microsoft Message Queue (MSMQ) [10] and IBM's MQSeries are robust commercial implementations that allow reliable asynchronous communication within guaranteed delivery constraints.

A notification service (NS) is a form of message-oriented middleware utilising the *Publish-Subscribe* model, acting as an intermediary responsible for the asynchronous delivery of messages between publishers and subscribers. (A NS is also referred to as Notification broker in the recent WS-Notification specification [6].) Publishers are information sources — they *publish* information about a given *topic*. Published information is delivered to anyone that has *subscribed* to that topic. Topics can be subscribed to by many subscribers, and published to by many publishers. The notification service takes the notifications from the publisher and handles their delivery to the subscribed consumers [7, 8, 12]. As well as simply passing on the messages, notification services can also filter and collect the notifications, allowing consumers to specify that they only want notifications matching certain criteria, from particular sources, or that they want to receive all of the notifications over a period in a single digest.

It is the responsibility of the notification service to ensure that the notifications are distributed to all of the subscribers — a publisher does not need to know who has subscribed to a topic. Notification services are able to offer persistent, reliable delivery of notifications, meaning that if a subscribed consumer cannot be reached (for example if they have disconnected or the network is down), the notification service can attempt to deliver the notification when they can be reached.

The notifications can, for instance, include announcements of changes in the content of databases [15], new releases of tools or services, and the termination of workflow execution. As such, the Grid community has recognised the essential nature of notification services such as the Grid Monitoring Architecture [17], the Grid Notification Framework [9], the logging interface of the Open Grid Services Architecture [4] and peer-to-peer high performance messaging systems like NaradaBrokering [5].

A notification service is also a core architectural element within the myGrid project [13]. myGrid (www.mygrid.org.uk) is an e-Science project that aims to help biologists and bioinformaticians perform workflow-based *in silico* experiments and also help them in automating the management of such workflows through personalisation, notification of change and publication of experiments. It focusses increasingly on data-intensive bioinformatics, and the provision of a distributed environment to support the *in silico* experimental process.

A NS can be viewed as a centralized server for message delivery and data persistence. This immediately introduces a potential scaling problem as the server gets heavily loaded and becomes unable to cope. Distributed notification servers overcome this problem by enabling consumers and publishers to

be connected to different NSs. Message routing is handled by the NSs to ensure that notifications reach their destinations.

Potentially, distributed NSs enable the number of messages being transmitted to be optimised. Consider two NSs, NS1 and NS2. NS1 has one publisher, and NS2 has two subscribers, both wanting to subscribe to the same topic. The two available options are for NS2 to make a subscription for each connected subscriber and have messages transmitted twice, or for NS2 to recognise that both subscribers are subscribed to the same topic and to reuse the same subscription to NS1 to serve both of them.

To address the distributed NS issue, the myGrid notification service [13] supports *federated notification topics*. At each NS, individual topics have metadata attached marking them as part of a federated notification topic. Information about federated topics and the local member topics is stored in a topic registry. When a consumer subscribes to a federated topic, the NS contacts the topic registry to find all other NSs that have a topic in the federated topic, and subscribes to these topics. This enables a consumer to receive notifications on a topic at one NS that were published at another NS.

In a distributed NS, it is still desirable to be able to request levels of QoS. However, as different levels are requested by many different consumers at the same NS, it becomes difficult to reuse existing subscriptions. For example, if a consumer has a subscription for notifications about database changes and has specified that notifications should be sent five hours apart, a second subscriber requesting notifications every hour is not going to be able to reuse the subscription. A form of negotiation is required to allow the subscriber to request appropriate levels of QoS and the NS to respond with alternative proposals that can put existing subscriptions to better use. To this end, we proposed *chained negotiation*, which we describe in the next section.

3 Negotiation

Negotiation is the process by which two or more parties communicate in order to reach a mutually acceptable agreement on a particular matter [11]. More specifically, negotiation can be described in terms of *protocols* and *strategies* [16]: protocols define the set of rules governing a negotiation such as the types of participants and valid negotiation states; and strategies determine how a single participant behaves within that protocol, including how it generates and responds to offers and when to bid.

3.1 Automated Negotiation

There are many situations where a developer would like to support negotiation over certain parameters without needing to know details of how to negotiate. Automated negotiation frameworks make this possible by defining the particulars of how to negotiate [2], in terms of protocols and strategies.

Although there is plenty of existing work on automated negotiation, we are not aware of any addressing our concern of chained negotiation. For example, in previous work [14], we introduced an automated negotiation engine based on a bilateral negotiation framework [3] intended for use in a notification service. Bartolini *et al.*[2] developed a framework for negotiation enabling different types of negotiation to be specified using rules. Jennings *et al.*[11] discuss another framework for automated negotiation focussing on rules of negotiation and allowing different types of negotiation to be carried out in the same framework.

3.2 Chained Negotiation

Direct negotiation takes place between one or more consumers and one or more suppliers. The consumers attempt to negotiate a price or set of constraints for a product or service they will obtain from the supplier. *Chained negotiation* is an extended form of negotiation where there is one or more intermediary (or *middleman*) between the consumer and the supplier. Chained negotiation enables middlemen to provide value-added services on top of an already provided service, or to make a service available more efficiently for a number of consumers. A middleman can exist for the purpose of making a profit or for increasing the social welfare of the community. In the situation where the subject of negotiation is information, a middleman is able to redistribute this to multiple consumers without each of them needing an individual agreement with the information provider.

Chained negotiation is suitable for use in a distributed notification service. Consumers subscribe to their local NS, but the provider may be connected to another NS. The distributed NS handles sending the messages between the various NSs, but has trouble when multiple consumers request different delivery constraints. For example, if one consumer requests updates to a database every day, and another consumer requests hourly updates, there is no way the existing notifications could be reused. Chained negotiation could be used for each consumer to find a set of mutually acceptable QoS values that could make use of subscriptions already in place.

In the rest of this section we give a brief overview

of our previous negotiation engine and then discuss the extensions made to support chained negotiation.

3.2.1 Bilateral Negotiation Engine

In previous work [14], we introduced a bilateral negotiation engine, in which negotiations take place for a *negotiation item*. In a notification service, this would be a subscription to notifications on a particular topic. The attributes of the item are called *negotiation terms*, which would be various QoS aspects such as message frequency, message size, granularity or cost of notifications.

The system as a whole is referred to as the *negotiation engine*. While conceptually it could be regarded as a single shared entity, it is split into a *negotiation component* (NC) at each party in the negotiation to maintain privacy of preferences and utility functions and enabling local information such as system load to be taken into consideration during the negotiation.

There are four types of messages used in the negotiation. *Proposals* are exchanged to find acceptable values. When one party receives a proposal they are prepared to accept, they respond with an *accept* message. This does not immediately commit them, but it does mean they have to commit if the other party replies with a *confirm accept* message. Negotiations can be terminated with a *terminate* message.

Due to space limitations, we omit details of how proposals are generated and evaluated — see [14] for details.

Selection of potential negotiation partners is beyond the scope of this negotiation model; we assume there is a fixed chain between a consumer and a supplier. For testing purposes we have used a fixed set of partners — these will be discovered using a suitable service discovery mechanism in future.

3.2.2 Chained Negotiation Extension

In a chained negotiation, there are three types of participants. *Consumers* attempt to obtain products or services from *suppliers*, but instead of contacting them directly, they go through a *middleman* that is likely to be local to them. Messages are exchanged sequentially between the involved parties. A typical sequence of message exchanges is shown in Figure 1, in which a distinction is made between messages travelling away (*upstream*) from the consumer (c_1) and towards the consumer (*downstream*). Messages travel from one end of the negotiation chain to the other via the middlemen (m_1 and m_2). If one of the middlemen can satisfy the request themselves, they reply to the incoming request instead of forwarding the message on.

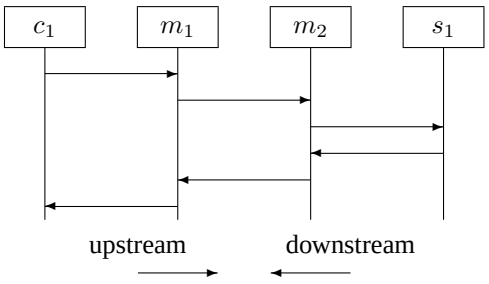

Figure 1: Message exchange in chained negotiation

One of the requirements we established when extending our bilateral negotiation model to cover chained negotiation was that the consumer and supplier should not have to know whether they are participating in a chained negotiation or not. This needed some changes to the model — mainly due to the time taken for negotiations. In the evaluation of our previous model, we used a simple interval-based time model in which the sending of one message takes one interval, rather than using the actual transmission time. If there was no successful proposal found by the time the deadline expired, the negotiation would fail. Using this system, chained negotiation could fail very easily because participants would make their final concessions as the deadline expires, but there might not be enough time to get the message to its destination.

To solve this we included an extra field in each message holding the distance (in number of hops) away from the consumer (furthest downstream) the sender of the message is. This is combined with a new rule stating that messages can only be sent upstream if there is enough time for the reply to reach the consumer: $time_{remaining} \geq dist{+}2$. This is because it is impossible to determine how far upstream a message needs to go — a proposal could be accepted by the closest middleman, or it could be sent on further. Because of this uncertainty, the rule is only for sending to the closest party.

The message types in chained negotiation are unchanged — the accept message enables all components to get to a "prepared to commit" stage without actually making the commitment. An extra rule has been introduced to the protocol stating that a negotiation component cannot confirm an accept with a downstream party without having an existing agreement or a new upstream commitment that can satisfy the downstream commitment. This prevents a consumer from thinking they have a successful subscription when it might not be possible for it to be made.

3.2.3 Proximity and Scoring Functions

A key part of chained negotiation is a *proximity* function, determining whether one proposal is *satisfiable* by another. Proposal p_1 is satisfiable by p_2 if each element of p_2 is at least as good as its counterpart in p_1 (from the point of view of the sender of p_1), and the negotiation object (the notification subject in our case) of each proposal is the same.

$$prox(p_1, p_2) = \begin{cases} -1 & \text{if Subj}(p_1) \neq \text{Subj}(p_2) \\ 0 & \text{if QoS}(p_1) = \text{QoS}(p_2) \\ < 0 & \text{if } p_1 \text{ not satisfiable by } p_2 \\ \geq 0 & \text{if } p_1 \text{ satisfiable by } p_2 \end{cases}$$

Proximity functions are used to determine appropriate existing commitments. Scoring functions are used to determine the best action to take each time a message is received by a middleman. An action could be either to offer a counter-proposal or to accept a received proposal from either direction. Actions are generated in two ways: selecting appropriate existing commitments using proximity functions and taking the last received proposals from each direction and passing them across. In this case the middleman can adjust values in the proposal, potentially to take a cut if there is a monetary aspect in the proposal. Using these two methods enables negotiation to take place through the middleman while making it possible to reuse existing commitments should one be close enough to the proposals.

The key to chained negotiation is in having good scoring functions. The framework has been developed to allow extra scoring functions to be plugged in, but in order to evaluate it we used three scoring functions. The first scoring function uses the proximity function to determine how close an action is to being acceptable. The second scoring function is one that favours actions that would lead to an acceptable state above ones that would not, and also to favour actions that are matched to an existing commitment, as this would incur less cost[1] upstream. The final scoring function we have used is one that increasingly favours actions in directions that have not been used recently. For example, if in two rounds of messages a downstream action is chosen, an upstream action will get a higher score from this function next time around. Actions are evaluated using a combination of all of the scoring functions.

If the deadline passes for a negotiation without a successful proposal being accepted, the negotiation is terminated. Negotiations can also be explicitly terminated for other reasons, such as no longer requiring the service, system shutdown etc.

[1]Cost is defined in terms of money or in having to do more work.

4 Evaluation

Before integrating our negotiation engine with a notification service, we have run a number of simulations to predict the performance of chained negotiations in the context of a notification service.

4.1 Experiment Setup

In our experimental setup, we have divided the negotiation components (NCs) into end NCs and middle NCs, representing the components and each end of a negotiation and the middlemen respectively. As the experimental system does not use any communication mechanism, the NCs are coupled directly together using method calls. The negotiation terms used in the experiments are abstract and represented by a numeric real value that could in turn represent any of the QoS terms suggested previously.

The set of varying factors in a negotiation experiment, such as preferences and deadlines, are grouped together into an *environment*. As there is an infinitely large space of possible environments, they are randomly generated in a repeatable manner, so that experiments may be re-run. In each experiment, end NCs are played against each other via a number of middlemen. Average values are then used to get an indications of real-world results. Tactics generate values for each term in a proposal. We have only used time-dependent tactics for the end NCs in this set of experiments for simplicity — for more details of tactics and the experiment setup please see [14].

In the experiments, we have compared how various factors affect three different types of negotiation: *Direct* negotiation is where the consumer and supplier are directly connected; *Chained* negotiation uses intermediaries to pass on proposals and attempts to match them to existing commitments and *forwarded* negotiation, which is a subset of chained negotiation where no existing commitments are reused. Forwarded negotiation represents the worst case that chained negotiation can take, assuming all requests are sufficiently different that commitments cannot be reused. In chained negotiation, we also examine the amount of negotiations that are *matched* — cases where existing commitments were reused instead of making new ones.

In our previous evaluation of direct negotiation we examined both consumer and supplier utility. An increase in consumer utility normally corresponds with a decrease in supplier utility. However, in chained negotiation the supplier may not be involved, as the request could be satisfied by a middleman. Hence we have concentrated on consumer utility in these experiments.

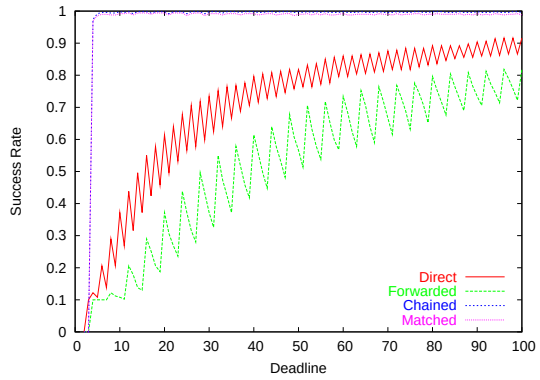

Figure 2: Effect of varying deadline on success rate

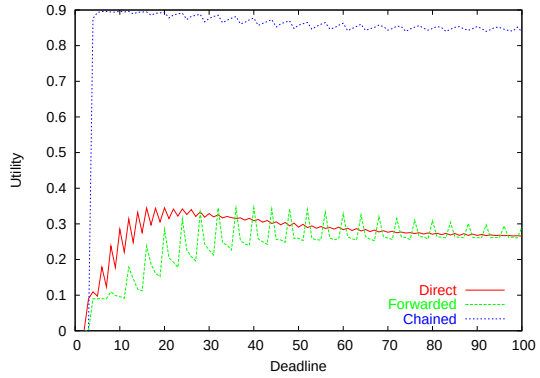

Figure 3: Effect of varying deadline on utility

4.2 Experiments and Results

4.2.1 Varying Deadline

When a negotiation must be completed by a certain time, a deadline is used to ensure that all agreements are made by this point. We previously determined that if a deadline was set too short, a negotiation was more likely to fail, and would give a lower utility to each party. We ran experiments through 500 environments using a single negotiation term to determine how the negotiation outcome would be affected as the deadline was varied upto 100 messages.

Figure 2 shows the success rates of different types of negotiation as the deadline is varied. Forwarded negotiation consistently performs worse than direct negotiation. Chained negotiation fares better, showing much better results almost immediately. This is because the set of environments the tests were run through was sufficiently large, and most of the negotiations were matched to existing commitments. The graph also shows a set of periodic spikes in the curves. These spikes occur because of the change to the protocol (described in Section 3.2.2): the consumer cannot determine the distance to its opponent, as the request could be fulfilled by any middleman inbetween. Thus it cannot determine the best time to offer its reservation value, and the negotiation is less likely to succeed. However, if the final offer is made when the remaining time allows the message to reach the supplier and return as the deadline expires, the negotiation is more likely to succeed with a better outcome for the consumer.

Figure 3 shows the consumer utility from the same experiment. This is shown to have similar results to the success rate — chained negotiation on average performs very well very quickly. Forwarded negotiation does not perform as well as direct negotiation with shorter deadlines, but approaches this level as the deadline becomes sufficiently large. The reason the utilities decrease slightly as negotiation deadlines increase is due to the concession pattern of the supplier. When there is a short deadline, concessions are made in larger steps; because the supplier can determine the last possible moment to make their final concession (to a reservation value), the consumer can get a better deal. With longer deadlines, the supplier concedes in smaller steps and a proposal that is acceptable to the consumer is often found before the deadline. earlier, resulting in the proposal being better with respect to the consumer's preferences.

To further investigate the effect of the middlemen in the chained negotiation, we ran the same experiment through different numbers of middlemen. We focus on forwarded negotiation, because no matter how many middlemen are present, after an initial commitment has been made, an offer similar to this would always be satisfied by the first middleman in the chain. We ran the same experiment described above and varied the number of middlemen between one and five.

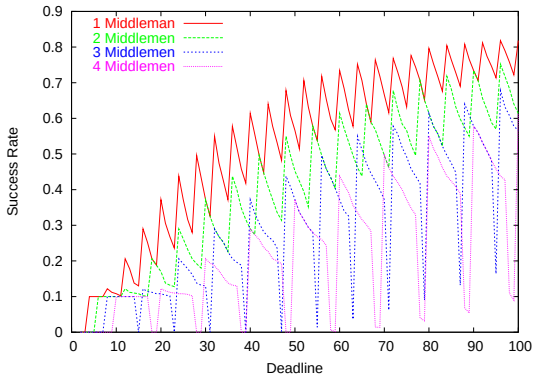

Figure 4: Effect of varying deadline with multiple middlemen

Figure 4 shows that as the number of middlemen increases, the success rate drops. It also illustrates an effect of using chained negotiation: as the number of middlemen increases, specific values for the dead-

line tend to make negotiation less successful. This typically occurs when there is only a small overlap between the ranges of values each party considers acceptable. The reason for this drop is that while the supplier can determine when it is allowed to make its final concession and offer the reservation value, the consumer cannot because it does not know how far away the supplier is, so the final concession made by the consumer does not reach the supplier before the negotiation deadline. This worst case will happen when the deadline is $2n(m+1)-1$, where m is the number of middlemen and n is an integer.

4.2.2 Varying Negotiation Terms

With a single negotiation term it is easy to match a new request to an existing commitment. With more negotiation terms, the negotiation has to get to the point where both terms are satisfiable by the commitment for it to be reused. We varied the number of terms the negotiation was run with, and used random deadlines between 30 and 60 messages.

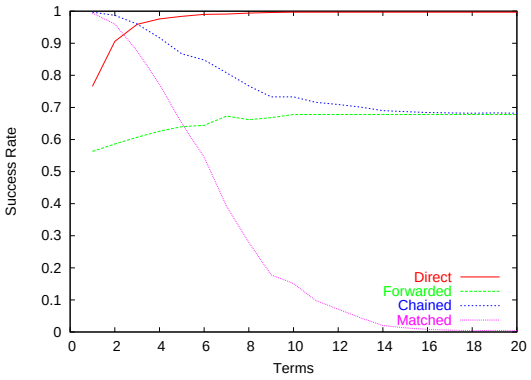

Figure 5: Effect of varying number of terms

Figure 5 shows that as the number of negotiation terms increases the number of negotiations that can be matched to existing commitments drops sharply. It also shows that forwarded negotiation does not perform as well as direct negotiation, which we explain with the same reason mentioned previously — forwarded negotiation performs worse than direct negotiation for a given deadline due to the extra time required to send messages.

We also ran the experiment through larger numbers of environments. As this produces more opportunities for commitments to be reused, the rate of matched commitments decreased at a slower rate. Overall, success rate in chained negotiation is badly affected by increasing the number of negotiation terms, but only to the level of forwarded negotiation as this is the worst case for chained negotiation.

4.2.3 Supplier capacity

One of the main objectives of chained negotiation is to reduce the amount of redundancy required in sending notifications, thereby increasing the number of consumers a single publisher can serve. To simulate this, we have set up an experiment where we have a number of NSs chained together with a single publisher. Consumers are spread evenly between the NSs, and negotiations are run between them. We measured the difference in success rate and utilities compared to all of the consumers negotiating with the publisher's NS directly.

	Success Rate	Utility
Direct	0.680	0.270
Chained	0.986	0.857

Table 1: Success rate & consumer utility with chained negotiation

Table 1 shows the success rate and consumer utility is higher when using chained negotiation than negotiating directly. Out of 493 successful negotiations, only 24 of them involved making commitments with the supplier. The rest were satisfied using existing commitments held by middlemen. Over time most matched negotiations will be satisfied by the local middleman, as this will make a corresponding commitment when a match from a more distant middleman is accepted.

Our preliminary investigation indicates that while chained negotiation may make initial agreements harder to reach, especially with multiple middlemen, this is outweighed by the benefit that emerges when enough subsequent requests are satisfied using existing commitments. In the context of a notification service, this allows a publisher at one NS to send notifications to more consumers at other NSs, while allowing consumers to specify aspects of QoS.

5 Conclusion and Future Work

In this paper, we have shown a need for chained negotiation to support QoS negotiation in a distributed notification service. We have presented our design of a chained negotiation engine, and shown that in some circumstances, better results in terms of negotiation success rates can be achieved by reusing existing commitments. This also reduces the load on the upstream components, enabling more consumers to be serviced. We have determined that chained negotiation will have a reduced benefit as negotiation involves more terms. However, we expect a small number of QoS terms to be used at one time, hence our system should provide some benefit.

The next stage of this work is to integrate the chained negotiation engine with the myGrid notification service. The myGrid NS does not currently support negotiation over QoS attributes, so this will also be added at the same time. Adding QoS negotiation abilities will enable consumers to request levels of QoS from the NS, with the NS retaining some control over what levels of QoS are realistic. Adding the chained negotiation will enable the NS to support many more consumers in the system as a whole without increasing the number of messages required significantly.

We have also identified a number of changes to the chained negotiation model which may provide better results, such as altering the middleman so it can negotiate with both sides of a negotiation simultaneously instead of sending a message in one direction and waiting for a reply.

Acknowledgements

This research is funded in part by EPSRC myGrid project (ref. GR/R67743/01).

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (ws-agreement), Feburary 2004.

[2] C. Bartolini, C. Preist, and N. R. Jennings. Architecting for reuse: A software framework for automated negotiation. In *3rd International Workshop on Agent-Oriented Software Engineering*, pages 87–98, Bologna, Italy, 2002.

[3] P. Faratin, C. Sierra, and N. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3–4):159–182, 1998.

[4] I. Foster, D. Gannon, and J. Nick. Open grid services architecure: A roadmap. Technical report, Open Grid Services Architecture WG, 2002. http://www.ggf.org/ogsa-wg/.

[5] G. Fox and S. Pallickara. The narada event brokering system: Overview and extensions. In H.R. Arabnia, editor, *2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, volume 1, pages 353–359, Las Vegas, Nevada, 2002. CSREA Press.

[6] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, S. Tuecke, W. Vambenepe, and B. Weihl. Web services notification (ws-notification), January 2004.

[7] Object Management Group. Event service specification. www.omg.org, mar 2001.

[8] Object Management Group. Notification service specification. www.omg.org, aug 2002.

[9] S. Gullapalli, K. Czajkowski, and C. Kesselman. Grid notification framework. Technical Report GWD-GIS-019-01, Global Grid Forum, jul 2001.

[10] P. Houston. Building distributed applications with message queuing middleware – white paper. Technical report, Microsoft Corporation, 1998.

[11] N. Jennings, S. Parsons, C. Sierra, and P. Faratin. Automated negotiation. In *5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems*, pages 23–30, Manchester, UK, 2000.

[12] Java Message Service API. http://java.sun.com/products/jms/, 1999.

[13] A. Krishna, V. Tan, R. Lawley, S. Miles, and L. Moreau. The mygrid notification service. In *Proceedings of the UK OST e-Science second All Hands Meeting 2003 (AHM'03)*, pages 475–482, Nottingham, UK, September 2003.

[14] R. Lawley, M. Luck, K. Decker, T. Payne, and L. Moreau. Automated negotiation between publishers and consumers of grid notifications. *Parallel Processing Letters*, 13(4):537–548, December 2003.

[15] T. Oinn. Change events and propagation in mygrid. Technical report, European Bioinformatics Institute, 2002. http://www.ebi.ac.uk/ tmo/change_notification.pdf.

[16] C. Sierra, N. Jennings, P. Noriega, and S. Parsons. A framework for argumentation-based negotiation. In *Intelligent Agents IV: 4th International Workshop on Agent Theories Architectures and Languages*, volume 1365 of *LNAI*, pages 177–192. Springer, 1997.

[17] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. Technical report, GGF Performance WG, 2002. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/.