Parallel Processing Letters

AUTOMATED NEGOTIATION BETWEEN PUBLISHERS AND CONSUMERS OF GRID NOTIFICATIONS

RICHARD LAWLEY, MICHAEL LUCK, KEITH DECKER,
TERRY PAYNE and LUC MOREAU
School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, UK

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

Notification services mediate between information publishers and consumers that wish to subscribe to periodic updates. In many cases, however, there is a mismatch between the dissemination of these updates and the delivery preferences of the consumer, often in terms of frequency of delivery, quality, etc. In this paper, we present an automated negotiation engine that identifies mutually acceptable terms; we study its performance, and discuss its application to a Grid notification service. We also demonstrate how the negotiation engine enables users to control the Quality of Service levels they require.

Keywords: Automated Negotiation; Grid Notification Services; Negotiation Framework

1. Introduction

Notification services play an important role within distributed systems, by acting as intermediaries responsible for the asynchronous delivery of messages between publishers and consumers. Publishers (such as information services) provide information that is then filtered and delivered to subscribed consumers [8,9,14] based on a specification of topic and delivery parameters.

While the mechanisms for asynchronous notifications are well understood, and robust implementations can be found, some issues still remain open. For example, providers hosting databases typically prefer to control the frequency at which notifications are published (through, for example, daily digests), and discourage clients from continually polling for changes [21]. However, clients have their own preferences about the frequency, format, quality or accuracy of the information being propagated.

Similarly, many services within the Bioinformatics domain are hosted by public institutions and are free to the community, but there are also paying customers expecting a certain Quality of Service from providers. The prices charged for notifications will affect attributes (e.g. quality and frequency) of messages sent. As these examples suggest, both providers and consumers have preferences about the way notifications should be published,

yet current notification service technologies provide no support for determining a set of parameters that would be acceptable to both parties.

Our approach to this problem is to use negotiation to find mutually acceptable Quality of Service values. In this paper, we present an automated negotiation framework that we are developing for integration with the myGrid notification service [17]. Our contributions are a practical implementation of an automatic negotiation engine, a study of bilateral negotiation in terms of performance, and a study of negotiation in the specific context of notification services. The paper is organised as follows. Sections 2 and 3 give some background information on notification services and negotiation in general. Section 4 describes the design of our system. The negotiation engine is examined in general in Section 5 and more specifically in Section 6, before concluding in Section 7.

2. Background on Notification Services

Over recent years, there has been a significant increase in the number of computers and other devices wishing to 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 and disconnect, reconnecting 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. Queueing products such as Microsoft Message Queue (MSMQ) [12] and IBM's MQSeries[a] are robust commercial implementations that allow reliable asynchronous communication within guaranteed delivery constraints.

A notification service 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. 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 distribution and delivery to the subscribed consumers [8,9,14]. 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 filters, from particular sources, or that they want to receive all of the notifications in one day 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 (although this information is available to the notification service). 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 come

[a] http://www.ibm.com/software/mqseries/

back up.

The notifications can, for instance, include announcements of changes in the content of databases [21], 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 [23], the Grid Notification Framework [10], the logging interface of the Open Grid Services Architecture [5] and peer-to-peer high performance messaging systems like NaradaBrokering [6]. A notification service is also a core architectural element within the myGrid project [17].

One remaining issue with notification services is the lack of ability to provide negotiable Quality of Service levels. Negotiation in this context is discussed in the rest of this paper.

3. Background on Negotiation

Interactions are a core part of any distributed system. When two parties wish to interact so as to achieve some task, they must first agree on their roles and the terms of their interaction. Negotiation is the process by which two or more parties communicate with one another to try to reach such a mutually acceptable agreement on a particular matter [13]. It can be described as a joint search over a problem space with the goal of reaching a consensus [11].

The subject of a negotiation is the *Negotiation Item*; for Notification services this could be an agreement to provide notifications for a specified period of time. The attributes of the item under negotiation are the *Negotiation Terms*, such as the notification frequency. All participants in a negotiation have *preferences*, which define their desired outcome of a negotiation, and they typically use *utility functions* to measure the value of such an outcome [16].

To enable negotiation to be carried out automatically between two parties, the particulars of negotiation must be defined [1]. Negotiation can be described in terms of *protocols* and *strategies*: protocols define the set of rules governing a negotiation [22], 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, when to bid, and so on.

We can generally consider two approaches to negotiation: *competitive* and *cooperative*, which are used depending on whether the participants are self-interested or benevolent.

Competitive negotiation takes place between self-interested parties. In this context, the primary objective of each participant is to maximise their own utility, normally at the cost of the opponent's utility; each participant has their own strategy, which is kept private [24]. If the opponent's strategy is known, it may be possible to take advantage of this fact and predict their behaviour, giving an unfair advantage. Similarly, knowing their preferences and utility functions also enables behaviour to be predicted. One form of competitive negotiation is *bargaining*, in which participants exchange proposals until an agreement is reached [22]. When a proposal is received, it is either accepted or rejected — if rejected, a counter-proposal is issued, making some concessions since the last proposal.

Cooperation is suitable for negotiations between participants that are part of the same organisation. In cooperative negotiation, preferences and utility functions are shared between the participants. Currently, most electronic marketplaces [11] use competitive negotiation [15], but a cooperative approach (sometimes used in the areas of resource allocation [18,7] and coalitions) might benefit the society as a whole in such contexts.

Although there are notification service scenarios in which it would seem appropriate to take a cooperative approach, such as when all participants involved are part of the same organisation, in most cases participants have at least an element of self-interest. It would also be very difficult to share information on utility and preferences when these are dependent on dynamic, locally-sensed environmental factors such as available computing resources, although such possibilities are being investigated elsewhere [2]. For these reasons, we have adopted a competitive approach for the provision of negotiation abilities to a notification service. This also suits scenarios in which service providers compete for business.

There has already been much research in automated negotiation. Bartolini et al. [1] developed a framework for negotiation allowing different types of negotiations to be specified using rules. We chose not to use this, as we wanted the two parties to be able to communicate directly rather than to use a third party to carry out the negotiation. Jennings et al. [13] give some details of a framework for automated negotiation, which focuses on rules of negotiation, and allowing many different types of negotiation to be carried out within the same framework. Tu et al. [24] describe a way of dynamically adding negotiation abilities to an agent at run time using a pluggable architecture, and splitting the negotiation into components. We have split the negotiation in the same way. The RFC1782 [19] describes a simple extension to Trivial File Transfer Protocol (TFTP) to allow option negotiation prior to the file transfer. Although the idea is similar in principle, there is no economically sound model of negotiation used.

4. Negotiation Engine

4.1. Negotiation Model

Automated negotiation is not a new concept, and many frameworks have been designed to facilitate it. We have based our work on an existing model, Faratin's *Negotiation Decision Functions* [4], which is a bilateral (one-to-one) negotiation model. It was chosen because it allows most of the negotiation protocol to be handled internally, while allowing external input such as resource functions to be used to control the negotiation strategy. Applications can thus effectively negotiate with varying levels of knowledge of the protocol or the strategy.

In Faratin's model, proposals are generated using methods based on *tactics* and *strategies*. Tactics are functions that generate the value for a single negotiation term for inclusion in a proposal, and come in different flavours: *time-dependent* tactics use the amount of time remaining until the deadline to concede, whereas *resource-dependent* tactics use a resource function to determine how much of a particular resource is consumed. This resource may be the number of negotiations currently taking place, or the load on the system, and may involve callbacks to monitor external resources. Multiple tactics can be combined for each term during proposal generation, and strategies are used to control the combination of these tactics. Each tactic can be assigned a weighting, which can be changed over the course of the negotiation. For example, a resource-dependent tactic can be favoured at the start of the negotiation but, as the deadline approaches, the strategy can modify the weightings so that a time-dependent tactic exerts more influence.

Utility functions evaluate the utility of a single negotiation term in a proposal. They can range from simple linear functions to more complex functions, and can include callbacks

to allow resource conditions to influence the utility. All utility functions are monotonically increasing or decreasing, and the utility of a proposal is a weighted summation of the utility of the elements within the proposal. Proposals become acceptable when the counter-proposal generated has lower utility than the incoming proposal.

The negotiation model is independent of any communication mechanism, which is expected to be supplied by the calling software. We envisage that this would be implemented as a Web Services port. The model also does not make any assumptions about dynamic discovery; discovery of negotiation partners is outside of the scope of the model, since there are already many service directory applications available.

4.2. Terminology

Negotiations take place between a *requester* and a *requestee*, and we assume that there is always an item that is the subject of the negotiation. As mentioned earlier, the conditions being negotiated over are called *negotiation terms*. Some negotiation terms we have identified that will be suited to the QoS negotiation include: Message Frequency (maximum time between notifications); Message Size; Accuracy or Granularity of Notification and Cost. A conversation between a requester and requestee, in which proposals are exchanged, is called a *negotiation thread*. Preferences are represented by two values: an *ideal value* and a *reservation value*. The ideal value represents the initial negotiating value (i.e. the value the party would like to obtain in an ideal world), while the reservation value is the limiting point of concession. Values beyond the reservation value are unacceptable, and the negotiation will fail if it is not possible to find a value that satisfies all preferences. The negotiations in this system work to *deadlines* that are measured in terms of the number of messages exchanged between the two parties, which we call the *negotiation thread length*.

We refer to the whole system as the *negotiation engine*. While conceptually this could be regarded as a single entity shared between the requester and requestee, it is split into two *negotiation components* (NCs), with each party (*host*) having their own NC. This approach maintains privacy of preferences and utility functions, and allows local information to be used as factors in the negotiation.

4.3. Negotiation Process

The negotiation process is depicted in Figure 1. Before a negotiation starts, both parties initialise their NCs with their preferences. Then, negotiation process properly begins with the requester sending a proposal to the requestee. The communication mechanism is left for the host to implement.

As discussed in the previous section, when a proposal is received by party p, a counter-proposal is generated. Using p's utility functions, p's NC determines if the utility of the counter-proposal is higher than the incoming proposal. If so, the counter-proposal is sent. This cycle continues until the incoming proposal has a higher utility, at which point an acceptance message is sent. Both NCs then give their hosts a successful proposal. Note that acceptance of a proposal is not a *commitment* — this is left to the host as it enabled negotiations with many parties, with the best proposal being selected.

If the deadline passes before a successful proposal is found, a failure message is sent to the other party, and the hosts are notified. Negotiations can also be terminated by a failure message for reasons such as a system shutdown or a deal being made with another party.

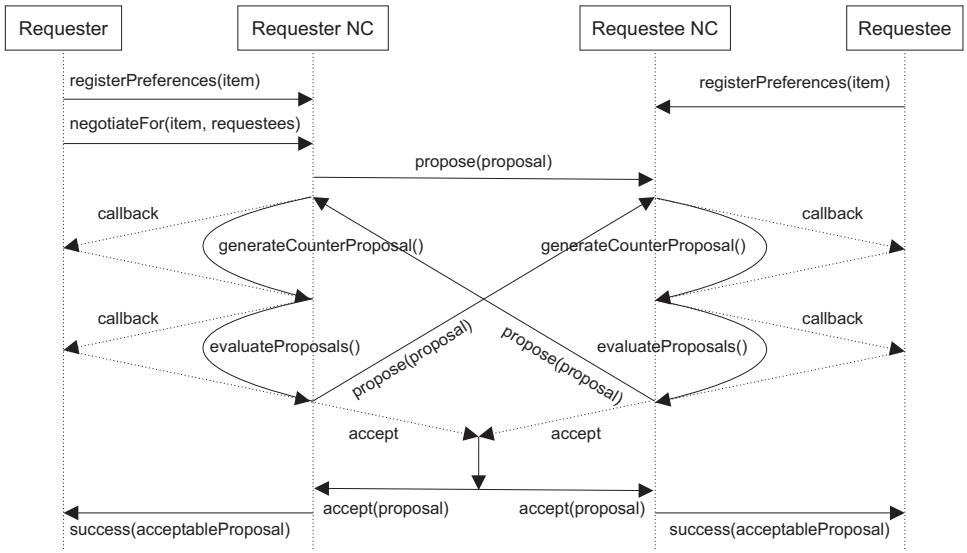

Fig. 1. Sequence diagram showing message flow during a negotiation

5. Experimental Evaluation of Negotiation Engine

In the context of a Grid notification service, the negotiation engine must scale up predictably to handle more negotiation terms, and longer negotiations, without any adverse performance. To determine that our engine was suitable for our purposes, we ran a number of simulations.

5.1. Experiment Setup

The set of varying factors, such as acceptable ranges and deadlines, are grouped into an *environment*. Running two negotiations in the same environment produces identical results, as the tactics used are deterministic. Thus, since there is an infinitely large space of environments, we can generate a range of random environments using Faratin's method [4]: we used six tactics — three time-dependent and three resource-dependent tactics; the utility functions are linear functions based on the preferences, although non-linear functions could be used with the restriction that they must be monotonically increasing or decreasing.

The experiments all have the same basic structure, with each tactic played against each of the tactics (including itself) in each of the generated environments. This allows us to build up some average values demonstrating the sort of results we should expect from a real-world implementation. The negotiation terms used are abstract and are represented by a numeric real value, that could, in turn, represent any of the negotiation terms described in Section 4.2. An experiment using concrete terms is described in Section 6.

As the design of the system is independent of any communication mechanism, the negotiation components are coupled together directly. Experiments measuring time only examine execution time.

5.2. Hypotheses and Results

The experiments described in this section are intended to determine the effects of varying both the deadline for negotiations and the number of negotiation terms. We consider the number of messages exchanged in a negotiation to be the primary component of time, since the dominant factor for message exchange in a real system is the transmission time. By contrast, transmission time in our experiments is very low, since the components are coupled by method calls, but we also measure *execution* time in Section 5.2.3.

5.2.1. Variable Deadline

Sometimes, negotiations should be completed within short deadlines, but this yields worse results than long deadlines [4]. To examine how the utility varies with the deadline, we varied the deadline between 1 and 100 messages, using a single negotiation term.

Hypothesis: *With short deadlines, the utility to both parties is poor. As deadlines increase, utility also increases, but at a decreasing rate, since a utility of 1 would indicate that no concessions were made, and is unlikely. The percentage of successful deals made increases as the deadline increases.*

Figure 2A shows that the utility (U_r and U_e) for both parties is low for short deadlines. As the deadline increases, the utility also increases. For reference we calculated an optimal utility based on symmetric utility functions at the middle of the overlapping region of the preferences. As this is independent of time, these are plotted on the graph as constants ($OptU_r$ and $OptU_e$), which appear to be asymptotes to the utility curve. Figure 2B shows that the percentage of successful negotiations has a similar curve, fitting our assumption that there is a predictable curve that can be used to determine the effect of limiting the

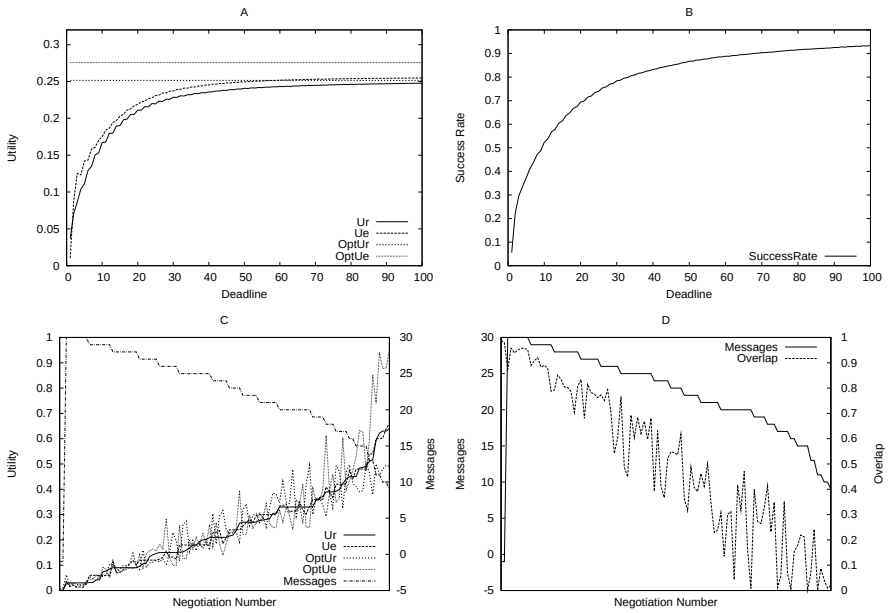

Fig. 2. A) Utilities, B) Success Rate, C) Utility vs. time used, D) Time vs. Φ

deadlines of negotiations. This can be used to determine the sorts of values that should be used to limit the length of a negotiation thread without trading-off too much utility.

To determine how much of the available time is used by the successful negotiations, we examined a subset of the data using negotiations between time-dependent linear tactics. We plotted the utilities of the outcomes and the time taken for the negotiations for each environment, sorted by increasing client utility and provider utility. Figure 2C shows that the amount of time used in a negotiation ranges between 30% and 100% of the available time. There appears to be an interesting trend where the utility increases as the number of messages decreases. The explanation is that the negotiations taking less time and giving greater utility have a better environment in which to negotiate. The parameter of the environment with the most significance is the amount of overlap between the acceptable regions. This is plotted against the number of messages exchanged in Figure 2D, confirming the theory that when the preferences have greater overlapping regions, negotiations finish faster and result in a greater overall utility.

5.2.2. Multiple Negotiation Terms

If this negotiation engine were to be deployed in the notification service, it would not be negotiating over a single negotiation term. There would be many terms, so we must ensure that the system scales up as the number of terms increases. In consequence, the negotiations were evaluated with the number of negotiation terms varying between 1 and 25. The experiments used deadlines of between 30 and 60 messages.

Hypothesis: *As the number of negotiation terms increases, the number of messages exchanged during a negotiation remains constant, since each negotiation term is independent, and the component concedes on all terms at the same time. Thus the length of the negotiation is constrained by the most limiting negotiation term. The utility of the outcome remains constant since the utility is limited by the most constraining negotiation term. As the number of terms increases, the time taken to perform the negotiations increases linearly, assuming that all the terms are evaluated using the same linear utility functions and tactics.*

Figure 3A shows that the average utility achieved with a varying number of negotiation terms remains fairly constant, and does not begin to drop with respect to the number of terms. Similarly, Figure 3B shows that the time increases linearly.

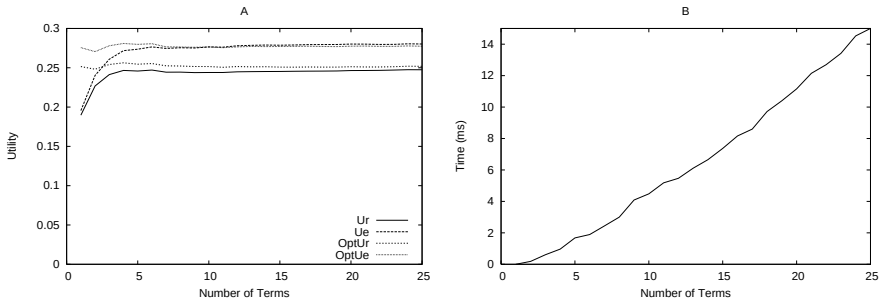

Fig. 3. Variable number of terms: A) Utilities of outcome, B) Time Taken

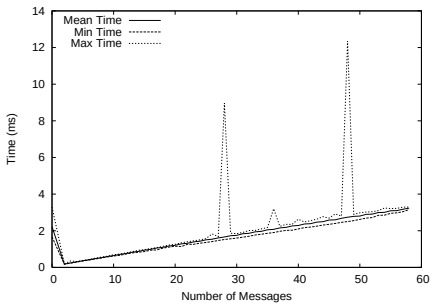

Fig. 4. Graph of time taken for negotiations

5.2.3. Execution Time

To confirm that the time taken for negotiation increases linearly with the number of messages exchanged, we measured the execution time for the negotiation to complete, averaged over 100 times to reduce inaccuracies. For a given number of messages exchanged, we recorded the average, minimum and maximum times taken for negotiations exchanging that quantity of messages. The deadlines in this experiment were between 30 and 60 messages.

Hypothesis: *An increase in the number of messages exchanged in a negotiation results in a corresponding linear increase in the amount of time taken.*

Figure 4 shows that the mean time taken for negotiations varies linearly as the number of messages exchanged increases. The maximum time has a couple of fluctuations, which have been explained as garbage collection. These are rare enough not to affect the mean execution time. Failed negotiations are plotted as 0 messages, although they always take up to their deadline to complete, because reaching the deadline implies failure. Since the time taken for failed negotiations is approximately the same as with high numbers of messages, we conclude that the time taken is linearly related to the number of messages exchanged.

6. Evaluation for the Notification Service

The previous experiments were not aimed specifically at any particular scenario. However, we are intending that this negotiation engine will be applicable for use in the myGrid notification service. myGrid[a] is a pilot project funded by the UK e-Science programme to develop Grid middleware in a biological services context.

To illustrate the functionality of Grid-based bioinformatics, myGrid has adopted an application that helps scientists study Graves' Disease, a hormonal disorder caused by overstimulation of the thyrotrophin receptor by thyroid-stimulating antibodies secreted by lymphocytes of the immune system [20]. The Graves' Disease application follows an *in-silico* experimental process typical of bioinformatics. In this process, the scientist attempts to discover information about candidate genes, makes an educated guess of the gene involved in the disease and then designs an experiment to be realised in the laboratory in order to validate the guess. This *in-silico* experiment operates over the Grid, where resources are

[a] http://www.mygrid.org.uk/

	CPU Time (sec)
With Negotiation	123,507
Without Negotiation	1,500,500

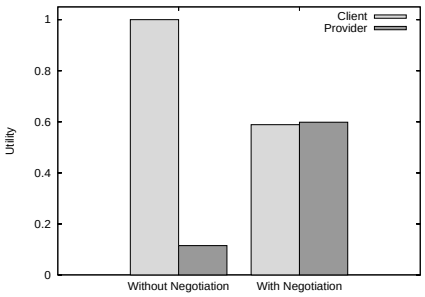

Fig. 5. CPU Time and Utility with/without negotiation

geographically distributed and managed by multiple institutions. It is a *data-intensive* Grid in which the complexity is in the data itself, the number of repositories and tools that need to be involved in the computations, and the heterogeneity of the data, operations and tools. Users would like the Graves' disease experiment to be run repeatedly as new data is added to the knowledge base, and be notified of any changes in the results.

Our specific case experiment is a simplified version of the above scenario – a notification service providing notifications from the SWISS-PROT database. SWISS-PROT is a curated protein sequence database providing a high level of annotation, minimal redundancy and high integration with other databases [3]. It can be queried for sequences and annotations that are related to specified sequences. It is continually expanding: in the eight months between the last two releases, the database grew by 10%, with an average of 230 changes per day[a].

For our example scenario, we assume 1000 subscribers are interested in anything that matches 100 different sequences, and a particular similarity search can be run with different precision. For simplicity, we represent precision by a number between 1 and 5, and we assume that a search with precision 2 takes twice as long as a search with precision 1. A particular consumer would like their search run as accurately as possible, every 8 hours. If we assume that a search with precision 1 takes 1 second, some 416 hours of CPU time are required every day (1000*100*(24/8)*5 seconds).

Negotiation is introduced into this experiment using two negotiation terms. Frequency represents the maximum number of hours between notifications; for the provider, this is between 12 and 72 hours, whereas for the consumer it is between 8 and 48 hours. The second term is the number of iterations of the search. The provider prefers this to be between 1 and 5 and the client between 5 and 1. The preferences for the provider are kept constant, and a random variation is introduced into the client preferences to simulate different clients. Negotiation deadlines are between 30 and 60 messages.

We ran negotiations using the terms described above, and calculated the average outcome. The average frequency was 46.2 hours, and the average number of iterations was 2.38. These figures lead to 123,507 seconds of CPU time, a reduction of 91% on simply allowing clients to request arbitrary Quality of Service levels. Figure 5A shows the difference in the amount of CPU time required when negotiation is introduced. Figure 5B shows the corresponding differences in utility seen by both sides using the preferences above.

[a] http://www.expasy.org/sprot/relnotes/relstat.html

Introducing negotiation has significantly lowered the amount of work the provider must do in this case, resulting in a significant increase in provider utility. However, this comes at the expense of the client utility; although there was a very large gap between the provider and client utility previously, the gap has now been reduced.

Introducing negotiation enables a provider to balance the utility of its clients with its workload. Although decreasing the client utility, lowering the amount of work required enables it to serve more clients while still satisfying them.

7. Conclusion and Future Work

In this paper we have presented our design for a negotiation engine for inclusion in a notification service. We have discussed our reasons for choosing a competitive negotiation method and shown how our negotiation engine works. We have also shown that the performance of the system is predictable and does not have any adverse effects when used with many negotiation terms.

The motivation behind the next stage of our work is peer-to-peer (P2P) notification services. In these systems, there are many interconnected notification services, and consumers only need to communicate with one. The notification services handle routing and delivery of messages between each other so that notifications are always sent to a consumer, wherever they are.

To handle negotiations in these interactions, we need to support chained negotiation, where intermediaries lie between the consumer and the provider. Initially, the intermediaries will pass proposals through, potentially adjusting the values to take a profit. Eventually, we intend that each stage of the chain will be a separate negotiation, as requirements could be fulfilled at any point in the chain. We will thus expand our negotiation engine to handle chained negotiation. Finally, we will deploy this work in the myGrid notification service to use negotiation in a real environment.

Acknowledgements

This research is funded in part by EPSRC myGrid project (ref. GR/R67743/01). Keith Decker from the University of Delaware was on sabbatical stay at the University of Southampton when this work was carried out.

References

[1] 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.

[2] C. Boutilier, R. Das, J. O. Kephart, G. Tesauro, and W. E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 89–97, 2003.

[3] EBI. Swiss-prot website. http://www.ebi.ac.uk/swissprot/, 2003.

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

[5] I. Foster, D. Gannon, and J. Nick. Open grid services architecture: A roadmap. Technical report, Open Grid Services Architecture Working Group, 2002.

[6] 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.

[7] M. Frank, A. Bugacov, J. Chen, G. Dakin, P. Szekely, and B. Neches. The marbles manifesto: A definition and comparison of cooperative negotiation schemes for distributed resource allocation. In *Proceedings of the 2001 AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pages 36–45, North Falmouth, Massachusetts, 2001.

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

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

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

[11] R. H. Guttman and P. Maes. Cooperative vs. competitive multi-agent negotiations in retail electronic commerce. In *Second International Workshop on Cooperative Information Agents (CIA '98)*, volume 1435 of *LNCS*, pages 135–147, Paris, France, 1998.

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

[13] N. R. 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.

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

[15] S. Klaue, K. Kurbel, and I. Loutchko. Automated negotiation on agent-based e-marketplaces: An overview. In *Proceedings of 14th Bled Electronic Commerce Conference*, pages 508–519, Bled, Slovenia, 2001.

[16] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. *Artificial Intelligence*, 75(2):297–345, 1995.

[17] 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)*, Nottingham, UK, 2003.

[18] R. Mailler, V. Lesser, and B. Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, volume AAMAS03, pages 576–583, Melbourne, AUS, 2003. ACM Press.

[19] G. Malkin and A. Harkin. TFTP option extension (rfc 1782). Technical report, Network Working Group, 1995.

[20] L. Moreau, M. Luck, S. Miles, J. Papay, K. Decker, and T. Payne. *Methodologies and Software Engineering for Agent Systems*, chapter Agents and the Grid: Service Discovery. Kluwer, 2004.

[21] T. Oinn. Change events and propagation in myGrid. Technical report, European Bioinformatics Institute, 2002.

[22] C. Sierra, N. R. 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*, pages 177–192. Springer, 1997.

[23] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. Technical report, GGF Performance Working Group, 2002.

[24] M. T. Tu, F. Griffel, M. Merz, and W. Lamersdorf. A plug-in architecture providing dynamic negotiation capabilities for mobile agents. In K. Rothermel and F. Hohl, editors, *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *LNCS*, pages 222–236, Stuttgart, Germany, 1998. Springer-Verlag.