Automated Negotiation for Grid Notification Services

Richard Lawley[1], Keith Decker[2], Michael Luck[1], Terry Payne[1], and Luc Moreau[1]

[1] Department of Electronics and Computer Science, University of Southampton, UK
{ral01r,mml,trp,L.Moreau}@ecs.soton.ac.uk
[2] Computer and Information Sciences Department, University of Delaware, Newark, DE 19716
decker@cis.udel.edu

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.

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 which is then filtered and delivered to subscribed consumers [1–3] based on a specification of topic and delivery parameters. Notification service features include persistent, reliable delivery and prioritisation of messages. Notifications may include announcements of changes in the content of databases [4], new releases of tools or services, and the termination of workflow execution. Notification services can also be used for replicating service directory contents [5]. As such, the Grid community has recognised the essential nature of notification services such as the Grid Monitoring Architecture [6], the Grid Notification Framework [7], the logging interface of the Open Grid Services Architecture [8] and peer-to-peer high performance messaging systems like NaradaBrokering [9]. They are also core architectural elements within the MyGrid [10] project.

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 in the bioinformatics domain prefer to control the frequency at which notifications are published (such as daily digests), and discourage clients from continually polling for changes. However, clients have their own preferences about the frequency, format, quality or accuracy of the information being propagated. Similarly, many services within this 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 the type (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.

Automatically finding mutually acceptable notification parameters for a set of terms (e.g. price, bandwidth, etc) can be viewed as a search for an optimum in a multidimensional space. A *cooperative* approach requires both parties to make valuation functions and preferences available to a search component. Here, preferences and utility functions are shared, enabling the optimal values to be found. However, it is not always possible to share preferences and valuation functions freely — businesses may view preferences as private, and valuations may be linked to a locally sensed environment. In these situations, an automatic search is not possible, as an optimum cannot be calculated. These situations require *competitive* approaches using mechanisms such as *negotiation*. Various approaches exist; in this paper we present a *bilateral negotiation framework* [11] that is applicable to the context of notification service negotiation.

Our contributions are a practical implementation of an automatic negotiation engine, a study in terms of performance of bilateral negotiation and a study of negotiation in the specific context of notification services. This paper is organised as follows. Section 2 discusses negotiation in general. Section 3 describes the design of our system. In Section 4 we study the negotiation engine in general and more specifically in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2 Negotiation

Negotiation is the process by which two or more parties exchange proposals in order to reach a mutually acceptable agreement on a particular matter. Parties in negotiation exchange *proposals* [12] that are either accepted or rejected. Rejection involves either turning down the proposal and allowing another to be sent or submitting a *counter-proposal*, so that both parties converge towards an agreement. *Utility functions* (used to evaluate a proposal) and *preferences* (defining an acceptable range of values for each term) remain private and, because they are stored locally, can be linked to external conditions such as resource levels. For this reason, it is not practical to use cooperative searching in a Grid environment, where limited system resources need to be allocated. One solution is Faratin's *negotiation decision functions* [11] algorithm, which is a bilateral negotiation model that allows external resource functions to be used to evaluate proposals and generate counter-proposals.

In Faratin's algorithm, the methods for generating proposals and counter-proposals are 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 in the negotiation thread 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. To combine different tactics during counter-proposal generation, strategies are used. These modify the weightings given to each tactic, which can be changed during a negotiation, for example to honour a resource-dependent tactic at the

start of a negotiation and a time-dependent one nearer the deadline. Utility functions evaluate the utility of a single negotiation term in a proposal. These can be simple linear functions or more complex callback functions to link the utility to resource conditions. 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. Faratin's algorithm is the basis for the rest of this paper.

3 Negotiation Engine Description

Our system is intended to allow applications to be programmed without specific knowledge of negotiation protocols. The actual negotiation mechanism is based on the work of Faratin [11], which allows a host to easily supply external conditions, such as resource levels, as inputs into the negotiation process without knowing *how* to negotiate.

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. The conditions being negotiated over are called *negotiation terms*, and can be such things as cost or duration. A conversation between a requester and requestee, where proposals are exchanged, is called a *negotiation thread*. Each party has a set of *preferences*, which consist of 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 get 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 — the *negotiation thread length*. We refer to the application containing the negotiation component as a *host* and to the complete system as the *negotiation engine*.

Although we conceptually regard the negotiation engine as an entity shared by the requester and requestee, it has been implemented as negotiation components distributed between each party. This approach maintains privacy of preferences and utility functions.

A description of the negotiation process is depicted in Figure 1. Before a negotiation starts, both parties initialise their negotiation components with their preferences. Then, negotiation process proper is initiated 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 negotiation component 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 negotiation components then give their hosts a successful proposal. Note that acceptance of a proposal is not a *commitment* — it is left to the host to commit, allowing negotiations with many parties.

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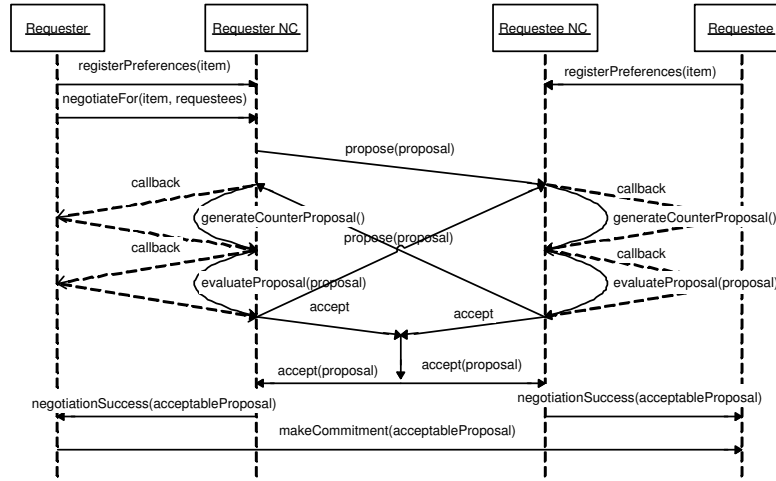
Fig. 1. Sequence diagram showing message flow during a negotiation

by means of a failure message for reasons such as a system shutting down or a deal being made with another party.

4 Experimental Evaluation of Negotiation Engine

To verify the suitability of our negotiation engine for a Grid notification service, we must check that it scales up predictably to handle more negotiation terms, and longer negotiations, without any adverse performance. To determine that this component is suitable for our purposes, we performed a number of experiments.
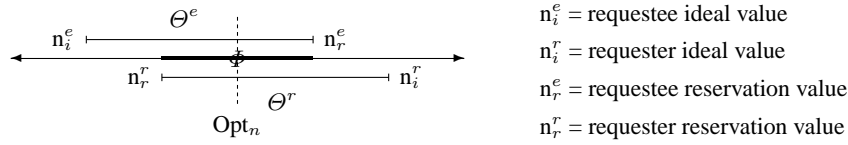
4.1 Experiment Setup

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 there is an infinitely large space of environments we generated a range of random environments using the methods from Faratin [11].

For each term, a fixed value min^r was chosen, representing the requester's reservation value. Random values within predefined ranges were assigned to the parameters Θ^r, Θ^e and Φ, where Θ^r represents the size of the acceptable region for the requester r, Θ^e represents the size of the acceptable region for requestee e, and Φ represents the degree of overlap between the acceptable ranges, with 0.99 indicating almost no overlap and 0 indicating complete overlap. These parameters are illustrated in Figure 2.

We used six tactics — three from the time-dependent and three from the resource-dependent families as in [11]. Utility functions are linear functions based on the preferences.

The experiments all had the same basic structure — each tactic was played against each of the tactics (including itself) in each of the generated environments. This allows

$$n_i^e = \text{requestee ideal value}$$
$$n_i^r = \text{requester ideal value}$$
$$n_r^e = \text{requestee reservation value}$$
$$n_r^r = \text{requester reservation value}$$

Fig. 2. Effect of environment parameters on preferences, also showing optimal value

us to build up some average values demonstrating the sort of results we should expect from a real-world implementation.

As the design of the system is independent of any communication mechanism we coupled the negotiation components together directly using method calls. Experiments measuring time only examined execution time.

4.2 Hypotheses and Results

The experiments described in this section were 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, as 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 measured *execution* time at the end of this section.

Variable Deadline Sometimes, negotiations should be completed within short deadlines, but this yields worse results than long deadlines [11]. 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 3A 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. The average optimal utilities (Opt_r and Opt_e) are plotted on the graph — these are time-independent and therefore constant. They appear to be asymptotes to the utility curve. Figure 3B 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 deadlines of negotiations. This can be used to determine what sort of values 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, 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 3C shows that the amount of time
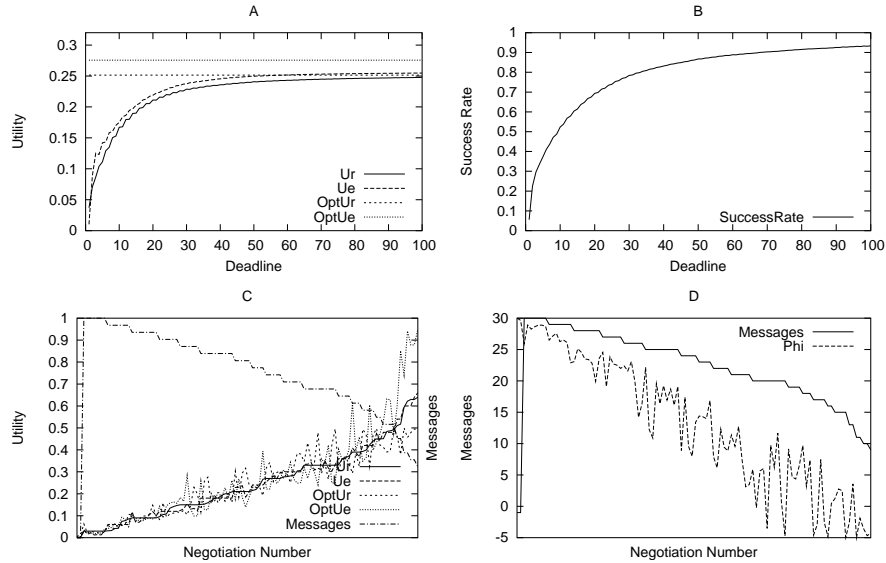
Fig. 3. A) Utilities, B) Success Rate, C) Utility vs. time used, D) Time vs. Φ

used in a negotiation ranges between 30% and 100% of the available time. There appears to be an interesting trend where the utility increased as the number of messages decreased. Our explanation for this is that the negotiations taking less time and giving greater utility have a better environment to negotiate in. The parameter of the environment that has the most significance is Φ, controlling the amount of overlap between the acceptable regions. This is plotted against the number of messages exchanged in Figure 3D, confirming this theory — negotiation finishes quicker with better negotiation environments and gives a greater overall utility.

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 set at every value 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, as 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.*
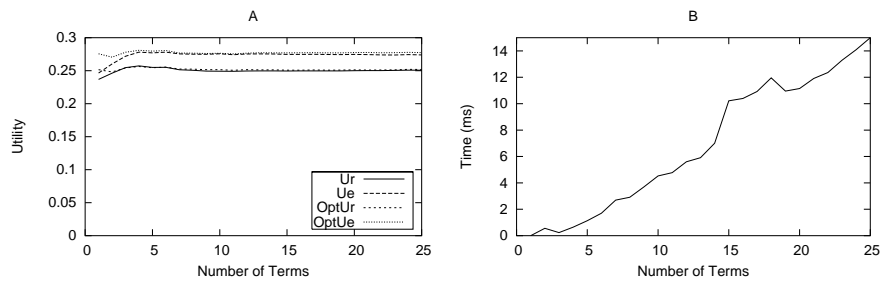
Fig. 4. Variable number of negotiation terms: A) Utilities of outcome, B) Time Taken

Figure 4A 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 4B shows that the time appears to be increasing linearly. While there are a few deviations upwards of the linear trend, these can be explained by garbage collection being triggered in the execution environment.

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: *As the number of messages exchanged between negotiations takes place, the corresponding increase in real time taken will be linear.*

As shown in Figure 5 there is a wide range of results for each number of messages exchanged. However, the average line is close to the minimum, indicating that few results are significantly higher than average. (We have explained the higher results as garbage collection in Java.) Failed negotiations are plotted as 0 messages, although they always take up to their deadline to complete, because reaching the deadline implies
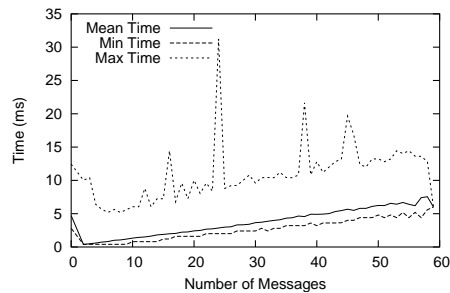
Fig. 5. Graph of time taken for negotiations
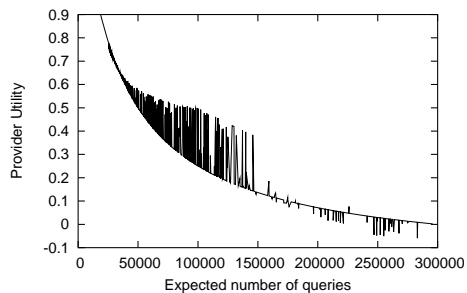
Fig. 6. Provider utility against expected number of calculations

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.

5 Evaluation for the Notification Service

We chose a specific case from the bioinformatics field as an example use of the notification service. The SWISS-PROT Protein Knowledgebase is a curated protein sequence database providing a high level of annotation, minimal redundancy and high integration with other databases [13]. Over the past 14 months it has grown by 20% with approximately 200 changes per day. As an example, we assume 1000 subscribers are interested in anything that matches 100 different sequences, and a particular similarity search takes 1 second and can be run iteratively to refine results. To provide the notifications, the provider runs the similarity search after data has been added or annotated. Although subscribers are happy to receive updates every few hours, this would place an unacceptable load on the provider. For example, daily searches iterated 5 times require five million searches per day.

   We use two negotiation terms for this experiment. Frequency represents the maximum number of hours between notifications: for the provider, this is between 24 and 168 hours, whereas for the consumer it is between 5 and 120 hours. The second term is the number of iterations of the search. The provider prefers this to be between 1 and 3, the client between 1 and 5. 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. After running the negotiations, the average value for the frequency was 67.6 hours and the number of iterations was 2.31 iterations. This works out as 651,000 searches per day, a reduction of 87%. The average utilities over the experiments were 0.39 for the consumer and 0.30 for the provider. Figure 6 shows that the provider utility decreases as the number of computations increases. The curve would have been smoother if we had weighted the utility of each term differently, as they influence the number of calculations in different ways.

   While these figures are based on a hypothetical situation, it demonstrates that the consumer's requirements can be satisfied while reducing the number of searches the

provider has to carry out to do so, indicating that a better Quality of Service can be achieved by using negotiation to establish QoS terms.

6   Related Work

There has been much research in automated negotiation. Bartolini et al. [14] produced 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. [15] give some details on 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. The RFC1782 [16] 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.

A number of examples of subscription services have been identified in the Grid community, allowing the information to be used for monitoring or rescheduling of allocations [17]. The Grid monitoring architecture [6] is a distributed architecture allowing monitoring data to be collected by distributed components. The Grid Notification Framework [7] allows information about the existence of a grid entity, as well as properties about its state, to be propagated to other grid entities. NaradaBrokering [9] is a peer-to-peer *event brokering system* supporting asynchronous delivery of events to sensors, high end performance computers and handheld devices. The Open Grid Services architecture [8] has also identified a logging service as an essential component: it also relies on producer and consumer interfaces. None of these notification services support negotiation and might benefit from our work.

7   Conclusion and Future Work

This paper has presented our design for a negotiation engine for inclusion in a Notification Service. We have presented 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.

Further development of this work is ongoing, and we have identified several areas we would like to proceed with. In our current system all negotiation terms are independent and negotiations concede on all of them. We would like to investigate introducing dependencies between negotiation terms and the possibilities of trading off one term against another. Negotiations currently take place between a requester and a notification service. We envisage a system where negotiations are chained between consumer, notification service and provider, and will examine negotiation in this situation.

Finally, it is worth noting that this negotiation component will be deployed in the Notification Service in a real Grid environment using MyGrid as a testbed.

Acknowledgements

References

1. Java Message Service API. http://java.sun.com/products/jms/ (1999)
2. Object Management Group: Event service specification. www.omg.org (2001)
3. Object Management Group: Notification service specification. www.omg.org (2002)
4. Oinn, T.: Change events and propagation in mygrid. Technical report, European Bioinformatics Institute (2002)
5. UDDI version 3 features list. www.uddi.org (2001)
6. Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., Wolski, R.: A grid monitoring architecture. Technical report, GGF Performance Working Group (2002)
7. Gullapalli, S., Czajkowski, K., Kesselman, C.: Grid notification framework. Technical Report GWD-GIS-019-01, Global Grid Forum (2001)
8. Foster, I., Gannon, D., Nick, J.: Open grid services architecure: A roadmap. Technical report, Open Grid Services Architecture Working Group (2002)
9. Fox, G., Pallickara, S.: The narada event brokering system: Overview and extensions. Technical report, Community Grid labs, Indiana University (2001)
10. myGrid Project: mygrid website. http://www.mygrid.org.uk/ (2003)
11. Faratin, P., Sierra, C., Jennings, N.R.: Negotiation decision functions for autonomous agents. International Journal of Robotics and Autonomous Systems **24** (1998) 159–182
12. Sierra, C., Jennings, N.R., Noriega, P., Parsons, S.: A framework for argumentation-based negotiation. In: Intelligent Agents IV: 4th International Workshop on Agent Theories Architectures and Languages, (Springer)
13. EBI: Swiss-prot website. http://www.ebi.ac.uk/swissprot/ (2003)
14. Bartolini, C., Preist, C., Jennings, N.R.: Architecting for reuse: A software framework for automated negotiation. In: 3rd International Workshop on Agent-Oriented Software Engineering, Bologna, Italy (2002) 87–98
15. Jennings, N.R., Parsons, S., Sierra, C., Faratin, P.: Automated negotiation. In: 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems, Manchester, UK (2000) 23–30
16. Malkin, G., Harkin, A.: TFTP option extension (rfc 1782). Technical report, Network Working Group (1995)
17. Schwiegelshohn, Yahyapour: Attributes for communication between scheduling instances. Technical report, GGF, Scheduling Attributes Working Group (2001)