Modelling & Simulating Chained Negotiation to Enable Sharing of Notifications

Richard Lawley Michael Luck Luc Moreau
School of Electronics & Computer Science, University of Southampton
E-mail: {ral01r,mml,L.Moreau}@ecs.soton.ac.uk

Abstract

Notification services (NSs) are middleware components providing asynchronous message delivery between publishers and consumers. Multiple interconnected NSs form a distributed NS, with each NS routing notifications between publishers and consumers at different locations, enabling consumers to share subscriptions, reducing the number of messages sent. Consumers can specify Quality of Service (QoS) levels when subscribing to a NS, using negotiation to find QoS levels acceptable to both parties. However, if consumers specify sufficiently different QoS levels, notifications cannot be shared and new subscriptions must be made. Chained negotiation can be used to negotiate QoS levels through intermediate NSs, enabling the reuse of existing subscriptions for additional consumers. In this paper, we present a chained negotiation engine, evaluating its performance and behaviour, showing that it enables negotiation over QoS while still sharing notifications, and that it provides better results for a consumer by negotiation directly with the publisher.

1. Introduction

A *Notification service* (NS) is a messaging middleware component providing asynchronous message delivery between publishers and consumers: publishers provide information that may be filtered and then delivered to subscribed consumers based on topic specification and delivery parameters [5]. However, consumers and publishers of notifications may have different and conflicting requirements on delivery parameters. For example, subscribers may wish to filter notifications based on a given set of criteria, while publishers may wish to limit the number of messages sent for reasons of policy or system load. Such requirements are essentially Quality of Service (QoS) measures. *Negotiation* provides a means to resolve the differences between the preferences of subscribers and publishers [8].

In large-scale deployments, multiple instances of a NS are hosted at different locations [7] and, consequently, publishers and consumers may interact with different NS instances — typically their local NS. Such distribution offers better scalability and security. A common configuration pattern consists of several NSs chained between a consumer and a publisher; hence publisher and subscribers can be separated by several NSs, or *middlemen*, which propagate notifications between publishers and consumers.

Although more complex, distributed notification offers potential efficiency gains. Instead of sending the same notification messages to multiple consumers subscribed to the same topic, it is possible to propagate a single message instance between NSs to reduce network traffic, *sharing notifications*. The difficulty, however, is that if consumers can negotiate QoS parameters for a subscription, two sets of requirements may be sufficiently different that they preclude xnotification sharing, and may impose a higher load on the network of NSs.

Negotiation is typically aimed at directly connected consumers and subscribers, which is unsuitable here. In response we designed a *chained negotiation* model, where consumers and publishers no longer communicate directly; negotiation takes place through middlemen, which pass proposals between publishers and consumers, potentially modifying them to satisfy their own QoS requirements. Middlemen also record previous commitments, and attempt to identify commitments that can be reused to enable an existing subscription to be shared with a new consumer.

In this paper, we discuss the design and evaluation of ChaNE, our chained negotiation engine, justifying the need for chained negotiation and showing the performance and behaviour of ChaNE.

2. Notification Services

Notification services are message-oriented middleware services for asynchronous communications over a network, for handling remote requests or for delivering information. Queuing products such as Microsoft's MSMQ and IBM's MQSeries are robust commercial implementations allowing reliable asynchronous communication within guaranteed delivery constraints.

Notifications can include announcements of changes in the content of databases [9], releases of tools or services, and the termination of workflow execution. The Grid community has recognised the value of NSs such as the logging interface of OGSA [4], and myGrid (www.mygrid.org.uk), where a NS is a core architectural element [7].

As mentioned earlier, larger-scale deployments of a NS could see multiple interconnected instances of a NS forming a distributed NS, for scalability and security. If consumers specify different levels of QoS to a publisher, negotiation is required to find mutually acceptable QoS levels. Since direct negotiation cannot be used (as the consumer and publisher may not be directly connected), we propose *chained negotiation*, which enables negotiation via the intermediate NSs. This enables a NS to recognise that a new subscription request is similar to an existing subscription, and to propose the existing subscription conditions to the consumer with the intention of sharing the existing subscription.

3. Negotiation

Negotiation is the process by which two or more agents communicate in order to reach a mutually acceptable agreement on a particular matter [6]. Although there is much existing work on automated negotiation (e.g., [2, 6]), none addresses our concern of chained negotiation. Previously [8], we introduced a direct negotiation engine based on a bilateral negotiation framework [3], which we use as the basis for ChaNE, our Chained Negotiation Engine.

Chained negotiation is an extended form of negotiation, in which a client initiates a negotiation with a *middleman* for a *negotiation item* (known as *agreement context* in WS-Agreement [1]), negotiating over various *issues* (terms in WS-Agreement). In a notification service, the negotiation item is a subscription to notifications on a particular topic, and the issues represent QoS levels for the subscription, such as message frequency or cost. If the middleman cannot provide the item itself, it initiates a negotiation with a supplier, or potentially another middleman, forming a chain between the client and the supplier. A fixed chain of partners is assumed in this paper, although a suitable service discovery mechanism would be used in practice.

Middlemen forward proposals between clients and suppliers so that an agreement can be reached, but they can modify the proposals sent between client and supplier. For example, when negotiating over a subscription that requires a payment, the middleman may choose to adjust the prices in the proposal so that it can make a profit in return for providing the service. Alternatively, middlemen may exist solely to benefit the community they serve, taking no profit.

When a chained negotiation terminates successfully, it can be placed into one of two classes: *matched* negotiation, which satisfies the client's request by using an existing subscription; and *unmatched* negotiations, which require a new subscription to be made upstream. We define *forwarded* negotiation as a variant of chained negotiation where no existing subscriptions are used. Forwarded negotiation represents the worst case of chained negotiation, as it provides no benefit to the client or supplier over direct negotiation.

In ChaNE, messages are exchanged sequentially between involved agents, *upstream* towards the supplier and *downstream* towards the client. Messages traverse the chain from client to supplier, unless a middleman can satisfy the request, in which case it replies to the incoming request instead of forwarding the message.

Two protocol rules ensure that negotiations terminate successfully if possible within the specified deadlines:

- *Messages must not be sent upstream unless there is time for the reply to reach the client.* This is achieved by keeping a record of the distance between each agent and the client and supplier in each negotiation, recorded in number of hops
- *Middlemen cannot initiate commitments downstream without having an upstream commitment in place to satisfy the request*. This guarantees that a client cannot make a commitment to an item that is unavailable.

Without the distance fields, a message could be sent upstream close enough to the deadline that a reply wouldn't reach the client, causing the negotiation to fail. Although the distance between client and supplier is known, a close intermediary can still satisfy the negotiation using an existing commitment.

The significant part of chained negotiation occurs when a message is received by a middleman, which generates a set of possible *actions* (each including a message to be sent), and then executes the best action, as shown in Algorithm 1.

Algorithm 1 Processing Propose and Accept messages

```
$msg_d$ =getLastDownstreamMessage()
$msg_u$ =getLastUpstreamMessage()

if remainingTime = 0 then terminateWithFailure()
if findCommit($msg_d$) or findUpstreamAccept($msg_d$) then
    actions.add(makeAction(ACCEPT, down, $msg_d$))
if $t_{rem} \geq (dist_D + 2 * dist_U)$ then
    actions.add(makeAction(PROPOSE, up, $msg_d$))
if $msg_u$ != null then
    actions.add(makeAction(PROPOSE, down, $msg_u$))
$a$ =selectBestAction(actions)
executeAction($a$)
```

Initially, the latest messages from each side are collected. The middleman then looks for a commitment that can satisfy the last message from downstream using a *proximity function*, which determines whether one proposal is satisfiable by another. A proposal p_1 is satisfiable by p_2 if each issue in p_2 is at least as good as the counterpart in p_1. When one proposal is **not** satisfiable by another, the proximity function determines how close they are. If a suitable commitment is found, or if an accept message from upstream has been received matching the downstream message, an action is generated to accept the downstream message. Otherwise, proposals from upstream are passed downstream and *vice versa*. Note that messages will only be sent upstream if there is time for a reply to reach the client. The middleman may modify a proposal at this point (to make a profit) but this is omitted from the algorithm. Finally, the best action is selected and executed.

4. Evaluation of Chained Negotiation

To evaluate the performance and behaviour of ChaNE, we performed a number of experiments. In these experiments, clients, suppliers and middlemen are connected directly (without any web-services-like transport layer), enabling the use of a simplified time model in which message transmission takes one unit of time. In a real-world scenario, we expect processing time to be negligible compared to transmission time, so this simplification allows us to concentrate on the behaviour of the model.

The varying factors in a negotiation are varied in every experiment, such as the preferences of each party. These are grouped into repeatable *environments*, through which each different experiment is run. In the experiments, clients and suppliers are played against each other via middleman, and averaged results are used to determine the outcome. More details of the experiment setup can be found in [8].

In these experiments, we examine the impact of chained negotiation as different factors are varied: the time available in which to negotiate; the number of middlemen in a chain; the number of issues negotiated over, and the amount of profit taken by a middleman. Optimal utilities are introduced for comparison purposes as the midpoint of the overlapping regions of client and supplier preferences.

Variable Negotiation Deadline. When an agreement must be in place by a certain time, deadlines are specified by which a negotiation must have completed. In direct negotiation, shorter deadlines lead to worse utility for both agents, and an increased chance of failure to make a deal. With chained negotiation, the time taken to send a message from client to supplier is longer, so the behaviour may change. To evaluate this, the deadline was varied between 1 and 100 messages, while negotiations were run with and without middlemen.

Figure 1A shows that direct negotiation oscillates over the optimal values, converging towards the optimal as the deadline increases. Forwarded negotiation behaves similarly, with a greater period of oscillation. Chained negotiation with a single middleman also oscillates, but converges towards a significantly higher utility, because once a *good* commitment has been found, it is reused for subsequent negotiations. When multiple middlemen are introduced to the chain, the oscillation period increases further, along with a decrease in the minimum utility seen throughout the oscillations. The oscillations in utility occur in each type of negotiation. To explain, consider direct negotiation, where messages take one period of time to be sent from client to supplier. Assuming replies are instantaneous, it is possible to predict which agent sends the last message, which will always be the reservation value (the largest concession they will make), assuming the negotiation did not complete earlier. Thus in direct negotiation, an increase to the deadline of 1 will swap the party making the final concession, which also means they receive the lowest utility, causing the oscillations in the graph. In chained negotiation, the time taken to send from client to supplier increases, thereby increasing the period of the oscillations. As longer deadlines make it more likely that a solution will be found *before* the deadline, the curves converge as the deadline increases.

During these experiments, the amount of messages actually exchanged was recorded. On average, direct and forwarded negotiation exchange the same number of proposals, linearly related to the deadline. Once allowed to establish some commitments, chained negotiation uses significantly fewer messages, as matches to existing commitments are identified quickly. Hence, over time, chained negotiations complete significantly quicker than direct and forwarded negotiation.

Variable Number of Issues. When using a single issue, it is easy to match a new negotiation to an existing commitment. As more issues are introduced, the negotiation must find a proposal where all issues are satisfied by an existing commitment to avoid having to make a new one. In this experiment, we varies the number of issues in a proposal and determined its effect on the level of re-used commitments and utility received. A single middleman was used for forwarded and chained negotiation.

Figure 1B shows that in chained negotiation, as the number of issues is increased, the number of negotiations that can be satisfied using existing commitments decreases. Since matched negotiations generally have a higher client utility than unmatched ones, client utility decreases, but only converging towards the levels of forwarded and direct negotiation. Forwarded and direct negotiation are not affected by the number of issues if concessions are made on each issue independently.

Middleman profit rate. In chained negotiation, middlemen are able to modify proposals before passing them on, enabling them to add a profit to any cost issue. However, if middlemen do take a profit, it becomes harder for the client and supplier to reach an agreement, causing negotiations to fail. To examine this effect, we varied the profit level on a single middleman running forwarded negotiations.

Figure 1C shows that as the middleman takes an increasing profit, the utility received by both client and supplier decreases. As the profit rate reaches 60%, almost no utility is received. Not shown on the graph is percentage of negotiations reaching a successful agreement — this decreases in the same way as utility, reaching 0 at 70% profit. Hence as the profit level is increased, fewer negotiations reach an agreement, and those that do, lead to lower client and supplier utility. The graph also indicates the average profit per negotiation for the middleman. Up to 25%, this increases steadily, but once above 30%, too many negotiations fail causing the average profit to fall. By combining the utility

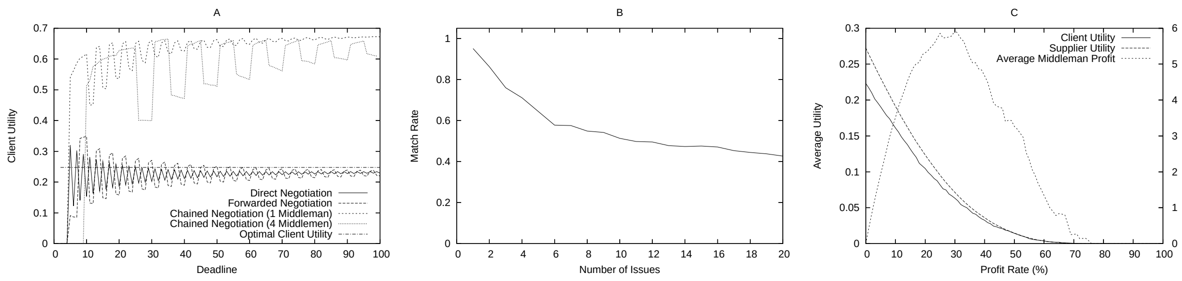

Figure 1. A) Utility with varying deadlines. B) Matched negotiations and issues. C) Utility and profit.

curves with the profit curve, we can calculate a system utility using a weighted average. This shows that as the profit is set too high, the utility to the whole system starts to decrease. Even for wholly self-interested middlemen, a profit rate above 40% is counter-productive.

Sharing of Notifications. One of the features of chained negotiation is that it can reduce the amount of redundancy required in sending notifications, thereby increasing the number of consumers a single publisher can serve. To demonstrate this, we set up a simulation in which 4 NSs were connected together in a linear chain, with the end NS connected to a publisher. A large number of consumers were spread between the NSs, and request subscriptions with different QoS levels. The publisher and each NS were restricted to making 500 downstream commitments. We then compare the number of consumers satisfied against the case of consumers subscribing directly to the publisher.

When connected directly to the publisher, only 500 consumers can be supported. However, existing subscriptions satisfy many requests when using the chain of NSs, *sharing* the notifications between consumers with similar requests. Here, 1911 consumers were satisfied before each NS reached their commitment limit, at which point the publisher had only made 31 commitments. Using different chains where multiple NSs connect to the publisher, a significantly higher number of consumers can be supported.

Summary. Compared to direct negotiation, chained negotiation can make it more difficult to make an initial agreement due to longer message transmission times, and utility that can be further from the optimal than with direct negotiation. However, once a middleman has existing commitments, subsequent negotiations can take advantage, resulting in quicker negotiations with higher utility than with direct negotiation. Below, we describe how chained negotiation can be used in a distributed notification service to enable a publisher to support more consumers.

5. Conclusions and Future Work

In this paper, we have shown that chained negotiation can enable distributed notification services to share notifications between subscribers, reducing the number of messages that must be sent. We have presented our design of a chained negotiation engine, and examined its behaviour as negotiation conditions are varied. By reusing existing subscriptions and sharing notifications, chained negotiation can lead to better utilities for consumers, and to negotiations being completed in less time. In addition, the load on a publisher can be reduced, enabling it to support more consumers. Our experiments have demonstrated that chained negotiation has a decreasing benefit as more issues are involved in a negotiation, though as we would expect a small number of issues to be used, chained negotiation should provide a benefit to a distributed notification service.

Future enhancements to this work include integrating ChaNE with the myGrid NS, showing a practical use of chained negotiation; making it WS-Agreement[1] compliant; negotiating in parallel with upstream & downstream agents; and to use multiple existing subscriptions to satisfy new requests.

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 Workshop on Agent-Oriented Software Engineering*, 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. Open Grid Services Architecture WG, 2002.

[5] S. Graham, P. Niblett, D. Chappell, S. Software, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, and B. Weihl. Publish-subscribe notification for web services. March 2004.

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

[7] A. Krishna, V. Tan, R. Lawley, S. Miles, and L. Moreau. The mygrid notification service. In *Proc UK OST e-Science second All Hands Meeting 2003*, pages 475–482, Sept. 2003.

[8] 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), 2003.

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