# myGrid Notification Service

Ananth Krishna Victor Tan Richard Lawley Simon Miles Luc Moreau

School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ UK
{kahs, hkvt99r, ra01r, sm, L.Moreau}@ecs.soton.ac.uk

**Abstract**

The notification service is the part of myGrid that enables asynchronous delivery of messages between distributed components. It includes features such as topic-based publish-subscribe messaging, push/pull models, asynchronous delivery, persistence, transient and durable subscriptions, durable topics, negotiation of QoS, hierarchical topic structure and federation of services. Some of these features are novel in the area of messaging middleware. A cost evaluation of some of these features indicate that the overhead incurred is justified in terms of compensating benefits gained.

## 1 Introduction

myGrid is an e-Science project that aims to help biologists and bioinformaticians to perform workflow-based in silico experiments and also help them in automating the management of such workflows though personalisation, notification of change and publication of experiments [10]. The focus of myGrid is on increasingly data-intensive bioinformatics and the provision of a distributed environment that supports the in silico experimental process. This leads to the idea of a lab-book environment in which the scientist can construct in silico experiments, find and adapt others, store partials results in local data repositories and adopt their own view on public repositories.

The myGrid experimental in silico process is expressed as a workflow script, which specifies how services should be composed in order to realise the experiment desired by the scientist. Some workflows may take days, if not weeks, to complete their execution. Users therefore need to be notified when workflow execution terminates. We prefer not to assume the existence of user agents able to handle incoming notifications. Indeed, users are not logged on permanently, and we feel that always running user agents would overload the system unnecessarily. Instead, we make use of a *notification service* able to forward messages to user agents, when present, or to store messages in their absence. The use of the notification service is of course not restricted to the user agent, but may be used by any services in myGrid. In particular, the notification service is also used to propagate notifications of changes in databases or in the service directory [9].

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 can be filtered and delivered to subscribed consumers under a given topic or channel [7]. As such, the Grid community has recognised the essential nature of notification services such as the Grid Monitoring Architecture [13], the Grid Notification Framework [6], the logging interface of the Open Grid Services Architecture [4] and peer-to-peer high performance messaging systems like Narada Brokering [11]. The notification service is also a core architectural element within the myGrid project.

The purpose of this paper is to present the myGrid notification service. In Section 3, we discuss its novel features. In Section 4, we describe its architecture. In Section 5, we evaluate the performance of some its aspects. Finally, we discuss related work and conclude the paper. But beforehand, we introduce the terminology and define the key concepts of a notification service.

## 2 Terminology

Before we proceed to discuss the features of the notification service in detail, we provide a definition for the various components within the notification service.

A messaging model describes the way in which exchange of messages between two or more communicating systems is achieved, particularly with regards to how messages are defined, published, consumed, organised and so on. A communication model concerns the modes of and the protocols used for communication.

### 2.1 Notifications, topics, publishers and subscribers

A notification is the message published by any application or component or object conveying some event occurrences on some specified resources. The notification also means the process of announcing some significant events. A topic is used to identify the kind of message being sent or received and can be used to group and distribute messages that are addressed to it. A publisher (or provider) is any application or component or object that publishes notifications, while a subscriber (or consumer) is any application or component or object that consumes notifications.

### 2.2 Notification Service

The Notification Service (abbreviated as NS from this point onwards) is a service that provides mechanisms for managing the synchronous or asynchronous transmission of notifications between publishers and subscribers. In this context, bi-directional communication or two-way communication means that a consumer is able to receive incoming messages and submit requests to the NS; a publisher is able to publish messages and receive incoming requests from the NS.

### 2.3 The publish-subscribe model

A subscriber that is interested in receiving messages relating to a specific topic can register an interest in that topic with the NS. This registration of interest represents a *subscription* and will have a expiration time representing the duration of interest. Publishers will publish messages on a given topic; these messages are retained by the NS and later propagated to the appropriate subscribers who possess a non-expired subscription. The use of topics and subscriptions in this way constitutes the publish-subscribe model, and has the advantage of decoupling the provider from the consumer providing for any asynchronous messaging model.

### 2.4 Push-pull models

Once a connection has been established, either between a publisher and a NS, or between a NS and a consumer, notifications can be sent. There exists two models for transferring data. In the push model, the source of notifications decides when data should be propagated to the notification sink, where as in the pull model, the notification sink polls the notification source for notifications. Push or pull models can be chosen for a given connection independently of other connections, and may possibly change. For instance, a consumer could pull notifications from a NS, whereas a provider can push them to the NS.

### 2.5 Messaging models

Messaging that requires simultaneous engagement of both publisher and subscriber of the exchange is called synchronous messaging. Asynchronous messaging refers to the situation where the publisher can initiate the communication regardless of whether the subscriber is actually available when the communication is initiated, and is the mode employed in the NS.

### 2.6 Transient and durable subscriptions

A transient subscription exists as long as the subscriber is running and will not persist or survive a subscriber crash, while a durable subscription uses persistence to provide reliability in message delivery. By sending persistent messages, it can be ensured that even when the notification service crashes, the messages will be delivered to the subscriber whenever the service is restarted.

## 3 Novel technical aspects

In this section, we discuss some of the novel features of the myGrid notification service as well as their underlying motivation.

### 3.1 Durable topics

Transient subscribers are capable of receiving messages only when they are active. Durable subscribers will store messages in a persistent store even if the subscriber is inactive. Durable topic subscribers are an extension of durable subscribers. In addition to storing messages in a

persistent store, they also account for messages published on durable topics.

Durable topics are introduced in order to solve a potential race condition that might arise in the publish-subscribe model. A publisher may publish on a given topic prior to any subscriber actually having subscribed to that topic. In such a case, messages published on that topic up to the point of time when the first subscriber subscribes to that topic is then lost. To prevent this, the concept of a durable topic is introduced. This provides the capability of retaining all messages published on that topic prior to the initial act of subscription. However the publisher can still a specify a time-to-live on the messages retained by the durable topic.

## 3.2 Leases

A lease is the term or duration of a contract granting use of resources. In the myGrid NS, leasing adds time to the notion of holding a reference to a resource, for example in a subscription to a topic by a subscriber. This enables the reference to be reclaimed safely if the lease is cancelled or expires. Expiry of a lease should be treated as cancellation of subscription and the publisher should be notified of such a change if no more subscribers exist. A lease can be defined by start and end times. The leasing mechanism should be able to handle lease checking, renewal and cancellation; this feature is yet to be implemented in the current NS.

## 3.3 Distributed notification service

A NS can be viewed as a centralized server for message delivery and data persistence. This approach works well for many scenarios though its drawback is that it tends to load the central server heavily and that the failure of the server might cause a failure of the entire service. In our approach, we emphasize the distributed service model. The idea is to eliminate a single point of failure by using multiple *hubs*. A hub is simply a notification server that is used primarily to hold registration and binding information of the notification servers connected to it. It also stores publisher and subscriber registrations, topic and message details. If one of the hubs fails, its connected notification servers will be handed over to another working hub, making it possible to recover from system failure. To do this, the notification servers store registration and binding information of all the hubs in the system, and

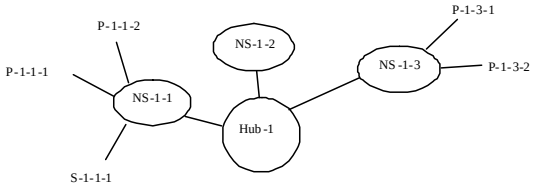


Figure 1: Simple hub model

will select a new hub (in the event of a failure) based on rules that ensure that the load is well balanced.

The basic structure of the model is publisher-service-subscriber where the service maintains persistent messages and routes messages in a smart way. However, it can be further scaled to more complex topology, forming a scalable distributed topology that supports communication to numerous servers collectively providing the notification service. Figure 1 is a simple hub model where three notification servers NS-1-i (i=1,2,3) connect to a master server called Hub-1. NS-1-1, NS-1-2 and NS-1-3 are Hub-1s clients. Hub-1 and NS-1-i are mutually visible. For simplicity, a notification server is only allowed to bind to one hub each time.

We assume that Hub-1 will store publisher and subscriber information including topics. If subscriber S-1-1-1 wishes to subscribe to topics published by publisher P-1-3-2, it first submits a request to the nearest notification server NS-1-1, who subsequently redirects the request to Hub-1 where P-1-3-2 and its topics can be found. This model thus is efficient in discovering topics because, as mentioned, all the information about publisher and topics are stored in the hubs and can be equally accessed by connected notification servers. There is no need for complex routing, thus eliminating on a substantial amount of network traffic.

Hubs like the one shown in Fig.1 can be assembled to create a more complex topology. In order to maintain data consistency, on events of registration/un-registration, disconnection, etc, all hubs will be updated and information will be replicated among them. Basically, the replication will take place if there is any change in the topology. This could be arise in any one of the following cases: a new notification server joining in, a notification server being removed, a new publisher/subscriber joining in, a publisher being removed, a subscriber being removed, a new hub joining in or a hub being removed. In addi-

tion, the replication takes place if a new topic is created, deleted or updated.

### 3.4 Negotiation over Quality of Service (QoS)

Both publisher and the consumer may have different requirements for the delivery of messages in terms of different criteria. For example, subscribers to a service may wish to filter the notifications they receive based on various factors such as the topic (event category) of the notifications, the frequency with which notifications are received or granularity of the information described by the notifications. These essentially represent different measures of QoS.

A subscriber may prefer, or demand, a particular parameter of QoS over another. Whether or how well their demands can be met by a publisher depends on the quality of service that the publisher can provide. If demands cannot be met exactly, the subscriber would then have to negotiate with the publisher to find the next best QoS that the publisher can provide. Alternatively, the publisher may be able to exceed the QoS in several ways which the subscriber may be unaware of, which could also lead to negotiation. 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. The location of such parameters is best achieved through a competitive approach such as negotiation, and we have developed a bilateral negotiation framework which has been implemented as an automatic negotiation engine [8].

### 3.5 Topic Management

As mentioned earlier, topics allow the decoupling of the publisher from the subscriber and provide an asynchronous mode of communication. The NS permits clients to create and delete topics dynamically. Hierarchical topics are supported. For example, topics MIRDataChange and TestDataChange may exist as leaf nodes under a topic called DatabaseNotifications which may in turn be a subtopic of GenericNotifications. A message published to leaf topics will automatically be applicable to super topics, i.e a message published to MIRDataChange will automatically be applicable to DatabaseNotifications and GenericNotifications.

Subscription to a topic entails subscriptions to all its sub-topics, due to hierarchical organisation of topics. However, filters can be used to subscribe to a particular subtopic only. There is an element of access control on topics, i.e topics should be visible only to certain clients and visibility should be restricted based on user profile.

## 4 Design and architecture

Figure 2 shows the architecture of NS. In order for the NS to facilitate communication and integration of services exposed by myGrid within and across domains, a subscriber-server-publisher model is adopted. The server, i.e. notification service is located between the subscriber and publisher and acts as the messaging middleware. In this model, the consumer and publisher are decoupled and the communication is asynchronous. Communication between subscriber and publisher is divided into two steps for both push and pull models. A subscriber can subscribe to a NS and the NS will retain the messages on behalf of the consumers, which can be collected later. On the other hand, a publisher will be able to publish notifications to the NS and NS decides which subscribers are qualified to consume the notifications. The NS provides functionalities like authentication, lease validation, session management, persistent store for messages, filtering, topic management, hierarchical namespace of topics, etc while driving the communication.

A client talks to the NS by making a Web Service call (Figure 2). The front-end is a client stub layer, making it possible for client components to publish and consume notifications. This is defined in WSDL in order to facilitate communication between a client component and the NS via SOAP calls. The NS consists of several components, most important ones being the publisher, subscriber and service browser implementations, which expose all features of the NS to a client. These components themselves delegate responsibilities to complex handlers like message filter handler, durable topic handler and QoS handler, which are responsible for message filtering, durable topic management and QoS management respectively. Apart from components shown in Figure 2, the NS also has Connection Pools to manage database connections, Resource Managers to manage system and JMS resources, Lease Manager to manage lease validations, Topic Manager to handle hierarchical namespace in topics and a Repository Handler

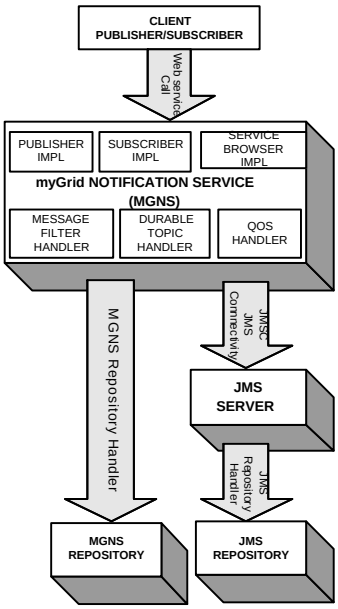


Figure 2: Architecture of notification service

to handle database read/write operations.

The NS connects to the JMS server via JMS Connectivity (JMSC). JMSC is a generic API which permits access to any JMS compliant server through Java. A generic interface was necessary to make the NS compliant with any JMS implementation because the JMS API does not precisely define the administrative API of a JMS implementation. A JMSC driver is used to implement the JMSC. The driver is a JMS server specific component (part of the administrative API). The use of JMS eliminates the need to address many development issues associated with distributed messaging, such as security, transactions, sessions, message filtering, message persistence, protocols and so on, because these issues have been handled very well by JMS servers. The JMS server is configured to work with a repository to provide message persistence, durability of consumers, etc to facilitate smooth recovery of data in case of a server failure. All JMS servers guarantee message delivery once and only once by way of session management and message acknowledgement.

Publishing a message involves a client who is registered in the NS to invoke a Web Service call on one of the publish methods exposed by the Publisher implementation. A message can only be published on a valid topic. Clients who can either be publishers or subscribers can create topics dynamically. QoS in the NS allows a topic to be defined as a durable topic. Messages published on a durable topic are saved against an anonymous durable consumer in the JMS server. The anonymous consumer registers an interest on the topic to receive all messages published on the topic. This mechanism will allow subscribers who register at any time with the NS to receive old messages published on the topic if they wish to do so. The message published can be a simple string, array of strings or a complex XML message. The NS converts the message to a JMS compliant message.

Subscribing to messages involves a client who is registered in the NS to invoke a Web Service call on the Subscriber implementation. A subscriber is allowed to have multiple subscriptions and QoS of a subscriber allows a subscription to be transient or durable and the subscription can either be pull or push. Transient, durable, push and pull subscriptions have all been defined in section 2. Pull subscriptions can also request a QoS with a pull message limit thereby bounding the number of messages for a pull operation. Durable subscriptions can receive old messages on a durable topic by requesting the relevant QoS.

Message filtering is a useful feature supported by the NS; it is based on properties set in the header of a message. Properties are set as key-value pairs and are called message filters. A publisher sets a message filter to the header of a message and publishes it on the NS. The NS internally maps this filter to a JMS filter, which publishes this message only to qualifying subscribers. It is important to note that subscribers can set different message filters for different subscriptions based on their specific area of interest. The filter, which is a set of key-value pairs, will be first converted to a JMS compliant message selector and only messages qualifying to these criteria will be served to the subscriber.

Another useful and important feature supported in the NS is hierarchical namespace in topics by way of utilising support offered in certain JMS implementations. However, it is important to realise here that the JMS specification does not make it mandatory for all JMS servers to support hierarchical namespace in topics and the NS supports hierarchical namespace only when JMS implementations themselves support a hierarchical topic structure. In the current release, we only support a static hierarchical topic structure. Messages can be published only on the leaf node in a hierarchy because only the leaf node in a hierarchy is a JMS

topic. Advantages of having a hierarchy are that publishers can publish messages under specific topics rather than broad based generic topics and subscribers can use wildcard parameters to subscribe to more than one topic at the same time. Disadvantages in the current release are that a hierarchical depth once defined cannot be changed, as it is static.

# 5 Evaluation and results

A series of experiments were conducted in order to evaluate performance of the notification service in conjunction with all features that have been implemented in the service. Performance of a messaging middleware can be measured in several ways. In this section we concentrate on measuring performance in terms of average time taken to publish a message. Number of messages published is gradually increased and the behaviour of the service is examined based on the results obtained. Results obtained are also used to determine any degradation in performance caused by implementing features such as durable topics and message filtering. The experimental set up involved the service to be deployed as a Web Service using Axis 1.1 inside Tomcat 4.1.24. OpenJMS-0.7.4 was used as the JMS server and it was configured to work with Mysql-2.0.11 as a persistent store. Although OpenJMS offers a performance of 500 msgs/ second for non-persistent messages sent to queues, there is no mention of performance for messages published on topics. This demonstrates that the limiting performance that can be achieved with OpenJMS is 50-60 msgs/second i.e. OpenJMS takes approximately 20 milliseconds to publish a message.

The following graphs represent performance statistics of the notification service with respect to experiments carried out in that order. It can be seen from the results that the average time taken to publish a message gradually decreases with an increase in the number of messages and then remains constant.

Figure 3 shows plots of average time taken in milliseconds to publish a non-persistent message on the NS without a web service call against publish times with a web service call. It can be seen from Figure 3 that a web service call is very expensive and adds a 200-250% overhead in comparison to average publish times without a web service call. It can also be seen that average publish times without a web service call

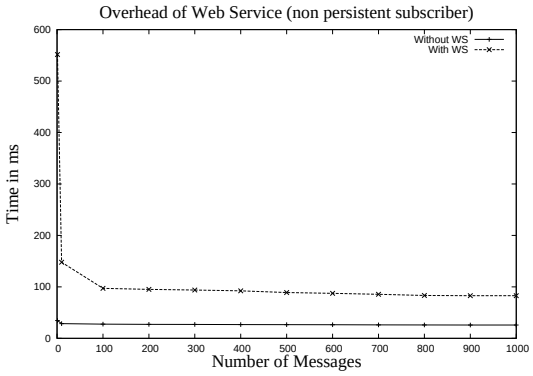


Figure 3: Publishing non-persistent message

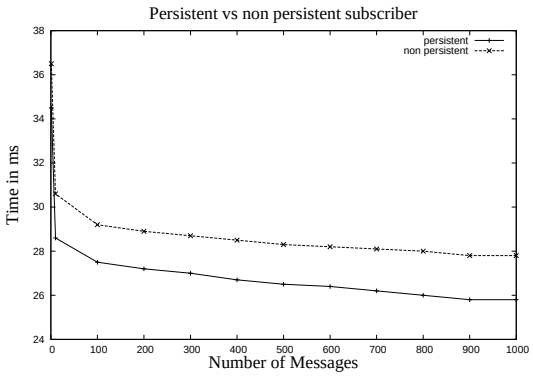


Figure 4: Publishing persistent message

is 20-25% slower in comparison to publish times offered by the underlying JMS server, OpenJMS in our case. This can easily be attributed to publisher authentication, topic authentication, lease validation, etc implemented in the NS.

Figure 4 shows plots of average time taken to publish a message when messages are marked persistent against publish times for non-persistent messages. By comparing the plots, we observce that messages marked persistent have 8-10% longer publish times than non-persistent messages.

Figure 5 show plots of average time taken to publish a message on a durable topic against publish times for persistent messages on a non-durable topic. By comparing the plots, we can surmise that persistent messages sent on a durable topic have 2-3% longer publish times than the ones sent on a non-durable topic.

Figure 6 show plots of average time taken to publish a message on a durable topic with a message filter specified on the message header against publish times for messages published on a durable topic without a message filter. We can conclude that published messages with a mes-

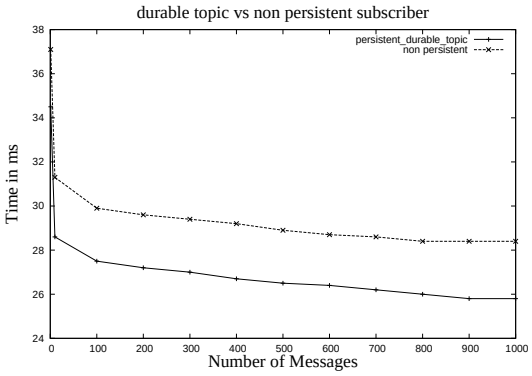


Figure 5: Publishing message on a durable topic

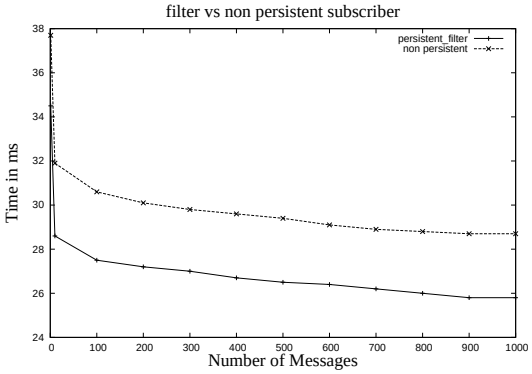


Figure 6: Publishing message with a message filter

sage filter in their header have 0.3-0.4% longer publish times than messages published without any filter.

The above results clearly show that features implemented in the notification service like JMS compliance for any JMS implementation, persistent store for messages, durable topic management and message filtering are marginally slower than their counterparts. However, Web Service calls cause a very large overhead. This can be largely attributed to expensive and complicated operations performed by the Axis engine.

# 6 Conclusion

In this paper we have discussed the myGrid notification service, which can be useful for the bioinformatics community and notifications in general. Some aspects like durable topics, negotiation of QoS, hierarchical topic structure, federation of services and Web Services compliance are new in distributed computing and messaging middleware. The results obtained show that most of the supported features in the NS justify the overhead they incur in terms of performance degradation as they offer compensating benefits.

Other examples of subscription services have been identified in the Grid community, for example allowing information to be used for monitoring or rescheduling of allocations [12]. The Grid monitoring architecture [13] is a distributed architecture allowing monitoring data to be collected by distributed components, collected in a database optimized for collection of data instead of querying data, and notifications of resource usage conditions to be sent to subscribers. The Grid Notification Framework [6] allows information about the existence of a grid entity, as well as properties about its state, to be propagated to other grid entities. This focusses on the content of the messages, rather than the notification service required. The Open Grid Services architecture [4] contains a Notification Framework [2] based upon JMS [7]. This is an interface to JMS through OGSA, allowing Grid services to use the JMS for notification services. NaradaBrokering [5] is a peer-to-peer *event brokering system* supporting asynchronous delivery of events to sensors, high end performance computers and handheld devices. It allows clients to disconnect and reconnect to a local broker instead of the one it was previously connected to, and have all waiting notifications delivered to it.

Future work in on the NS will include introducing a dynamic hierarchical topic structure and implementing the standalone negotiation engine into the notification service. We are also interested in exploring the implementation of the federation of services in a P2P manner, which will make the service more scalable and eliminate the single point of failure problem in a network of services. In our evaluations, we are also interested in measuring the round-trip time taken in the notification service as well as the performance of the service as a Web Service with evolving technologies such as JAX-RPC.

# 7 Acknowledgment

This research is funded in part by EPSRC myGrid project (reference GR/R67743/01). The authors would like to acknowledge the myGrid team (past and present): Matthew Addis, Nedim Alpdemir, Rich Cawley, Neil Davis, Keith Decker, David De Roure, Vijay Dialani, Alvaro Fernandes, Justin Ferris, Robert Gaizauskas, Kevin Glover, Carole Goble, Chris Greenhalgh, Mark Green-

wood, Yikun Guo, Peter Li, Xiaojian Liu, Phil Lord, Michael Luck, Darren Marvin, Karon Mee, Arijit Mukherjee, Tom Oinn, Juri Papay, Savas Parastiditis, Norman Paton, Terry Payne, Steve Pettifer, Milena Radenkovic, Peter Rice, Angus Roberts, Alan Robinson, Tom Rodden, Martin Senger, Nick Sharman, Robert Stevens, Paul Watson, Anil Wipat, and Chris Wroe. We also thank our industrial parners: IBM, Sun Microsystems, GlaxoSmithKline, AstraZeneca, Merck KgaA, geneticXchange, Epistemics Ltd, and Network Inference.

## References

[1] Marcos Aguilera, Rob Strom, Daniel Sturman, Mark Astley, and Tushar Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, May 1999.

[2] JaiPaul Antony. Jms notification framework. Technical report, 2003.

[3] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Symposium on Principles of Distributed Computing*, pages 219–227, 2000.

[4] Ian Foster, Dennis Gannon, and Jeffrey Nick. Open grid services architecure: A roadmap. Technical report, Open Grid Services Architecture Working Group, 2002. http://www.ggf.org/ogsa-wg/.

[5] Geoffrey Fox and Shrideep 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. Gullapalli, K. Czajkowski, and C. Kesselman. Grid notification framework. Technical Report GWD-GIS-019-01, Global Grid Forum, July 2001.

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

[8] Richard Lawley, Keith Decker, Mike Luck, Terry Payne, and Luc Moreau. Automated negotiation for grid notification services. In *Ninth International Europar Conference (EURO-PAR'03)*, Lecture Notes in Computer Science, Klagenfurt, Austria, August 2003. Springer-Verlag.

[9] Simon Miles, Juri Papay, Vijay Dialani, Michael Luck, Keith Decker, Terry Payne, and Luc Moreau. Personalised grid service discovery. *IEE Proc.-Software*, 2003.

[10] Luc Moreau, Simon Miles, Carole Goble, Mark Greenwood, Vijay Dialani, Matthew Addis, Nedim Alpdemir, Rich Cawley, David De Roure, Justin Ferris, Rob Gaizauskas, Kevin Glover, Chris Greenhalgh, Peter Li, Xiaojian Liu, Phillip Lord, Michael Luck, Darren Marvin, Tom Oinn, Norman Paton, Stephen Pettifer, Milena V Radenkovic, Angus Roberts, Alan Robinson, Tom Rodden, Martin Senger, Nick Sharman, Robert Stevens, Brian Warboys, Anil Wipat, and Chris Wroe. On the Use of Agents in a BioInformatics Grid. In Sangsan Lee, Satoshi Sekguchi, Satoshi Matsuoka, and Mitsuhisa Sato, editors, *Proceedings of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing*, pages 653–661, Tokyo, Japan, May 2003.

[11] Shrideep Pallickara and Geoffrey Fox. Naradabrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids. In *ACM/IFIP/USENIX International Middleware Conference (Middleware-2003)*, Rio de Janeiro, Brazil, 2003.

[12] Schwiegelshohn and Yahyapour. Attributes for communication between scheduling instances. Technical report, GGF, Scheduling Attributes Working Group, 2001. http://ds.e-technik.uni-dortmund.de/ yahya/ggf-sched/WG/sa-wg.html.

[13] 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. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/.