A Grid Message Service

Shrideep Pallickara* Geoffrey Fox†

March 17, 2001

1 Introduction

Distributed messaging systems broadly fall into the three different categories. These include queuing systems, remote procedure call based systems and publish subscribe systems. Message queueing systems with its store-and-forward mechanisms come into play where the sender of the message expects someone to handle the message while imposing asynchronous communication and guaranteed delivery constraints. The two popular products in this area include include IBM's MQSeries [IBM00] and Microsofts MSMSQ [Hou98]. A widely used standard in messaging is the Message Passing Interface Standard (MPI). MPI is designed for high performance on both massively parallel machines and on workstation clusters. Messaging systems based on the classical remote procedure calls include CORBA [OMG00c], Java RMI [Jav99] and DCOM [EE98]. Publish subscribe systems form the third axis of messaging systems and allow for decoupled communications between clients issuing notifications and clients interested in these notifications.

The decoupling relaxes the constraint that publishers and subscribers be present at the same time, and also the constraint that they be aware of each other. The publisher also is unaware of the number of subscribers that are interested in receiving a message and requires no synchronization with subscribers. The routing of messages from the publisher to the subscriber is within the purview of the message oriented middleware (MOM) which is responsible for routing the right content to the right consumers. The publish subscribe paradigm can support both the *pull/push* paradigms. In the case of pull, the subscribers retrieve messages from the MOM by periodic polling. The push model allows for asynchronous operations where there are no periodic pollings. Industrial strength products in the publish subscribe domain include solutions like *TIB/Rendezvous* [TIB99] from TIBCO and *SmartSockets* [Cor00b] from Talarian. Variants of publish subscribe include systems based on content based publish subscribe. Content based systems allows subscribers to specify the kind of content they are interested in. These content based publish subscribe systems include *Gryphon* [BCM+99, ASS+99], *Elvin* [SA97] and *Sienna* [CRW00]. The system we are looking at MSpaces is also in the realm of content based pub/sub systems with the additional feature of location transparency for clients.

The shift towards pub/sub systems and its advantages can be gauged by the fact that message queuing products like MQSeries have increased the publish subscribe features within them. Similarly OMG introduced services that are relevant to the publish subscribe paradigm. These include the Event services [OMG00b] and the Notification service [OMG00a]. The push by Java to include publish subscribe features into its messaging middleware include efforts like JMS [HBS99] and JINI [AOS+99]. Various JMS implementations include solutions like *SonicMQ* [Cor99] from Progress, *JMQ* [iPl00] from iPlanet, *iBus* [Inc00] from Softwired and *FiranoMQ* [Cor00a] from Firano. We envision a system with thousands of server nodes providing a distributed event service in a federated fashion. The grid message service (GMS) Mspaces, provides the architecture and protocols for achieving this.

*Dept. Of Electrical Engineering & Computer Science, Syracuse University
†Computational Science a& Information Technology, Florida State University

2 System Model

The system comprises of a finite (possibly unbounded) set of *server nodes*, which are strongly connected (via some inter-connection network). Special nodes called *client nodes*, can be attached to any of the server nodes in the network. Client nodes can never be attached to each other, thus they never communicate directly with each other. Let $\mathbf{C}$ denote the set of client nodes present in the system. The nodes, servers and clients, communicate by sending events through the network. This communication is *asynchronous* i.e. there is no bound on communication delays. Also the events can be lost or delayed. Some of the server nodes have access to a *persistent store* to facilitate delivery in the presence of failures and prolonged client disconnects. The failures we are presently looking into are node failures (client and server nodes) and link failures. The server node failures have crash-failure semantics. As a result of these failures the communication network may *partition*. Similarly *virtual* partitions may stem from an inability to distinguish slow nodes or links from failed ones. Crashed nodes may rejoin the system after recovery and partitions (real and virtual) may heal after repairs.

2.1 The event service problem

Client nodes can issue and deliver events. Every event e is time stamped message. Any arbitrary event e contains implicit or explicit information regarding the client nodes which should deliver the event. We denote by $L_e \subseteq \mathbf{C}$ this destination list of client nodes associated with an event e. The dissemination of events can be one-to-one or one-to-many. Client nodes have intermittent connection semantics. Clients are allowed to *leave* the system for prolonged durations of time, and still expect to receive all the events that it missed, in the interim period, along with real time events on a subsequent re-*join*. The system places no restriction on the server node that a client node can attach to, at any time, during an execution trace σ of the system. We term this behavior of the client as *roam*. Clients could also initiate a roam if it suspects, irrespective of whether the suspicion is correct or not, a failure of the server node it is attached to.

For an execution σ of the system, we denote by E_σ the set of all events that were issued by the client nodes. Let $E^i_\sigma \subseteq E_\sigma$ be the set of events e^i_σ that should be relayed by the network and delivered by client node c_i in the execution σ. During an execution trace σ client node c_i can *join* and *leave* the system. Node c_i could also *recover* from *failures*. Besides this, as mentioned earlier client nodes can roam (a combination of leave from an existing location and join at another location) over the network. A combination of join-leave, join-crash, recover-leave and recover-crash constitutes an *incarnation* of c_i within execution trace σ. We refer to these different incarnations, $x \in X = 1, 2, 3...$, of c_i in execution trace σ as $c_i(x, \sigma)$.

The problem pertains to ensuring the delivery of all the events in E^i_σ during σ irrespective of node failures and location transience of the client node c_i across $c_i(x, \sigma)$. In more formal terms if node c_i has n incarnations in execution σ then

$$\sum_{x=1}^{n} c_i(x, \sigma).deliveredEvents = E^i_\sigma.$$

2.2 Assumptions

(a) Every event e is unique.

(b) The links connecting the nodes do not create events.

(c) A client node has to accept every message, events and control information routed to it.

(d) Not all events can have zero targeted clients.

(e) If a client issues an event e infinitely often, eventually the event would be disseminated within the system.

Items (d) and (e) constitute the liveness property eliminating trivial implementations in which all events are lost or all events have no targeted clients.

3 The Server Node Topology

The smallest unit of the system is a *server node* and constitutes level-0 of the system. Server nodes grouped together from a *cluster* and level-1 of the system. A single server node could also decide to be part of a traditional clusters, or along with other such server nodes form a cluster connected together by geographical proximity but not necessarily high speed links. Several such clusters grouped together as an entity comprises the level-2 of our network and is referred to as a *super-cluster*. Clusters within a super-cluster have one or more links with at least one of the other clusters within that super-cluster. This topology could be extended in a similar fashion to comprise of *super-super-clusters* (level-3) and so on. In general there would be multiple links connecting a single unit to several other units, which provides for a higher degree of fault tolerance by providing multiple *routes* to reach the same unit. We limit the number of super-clusters within a super-super-cluster, the number of clusters within a super cluster and the number of nodes within a cluster viz. the *block-limit* to 64. In a N-level system this scheme allows for $2^6_{N-1} \times 2^6_{N-2} \times \cdots 2^6_0$ i.e 2^{6*N} server nodes to be present in the system.

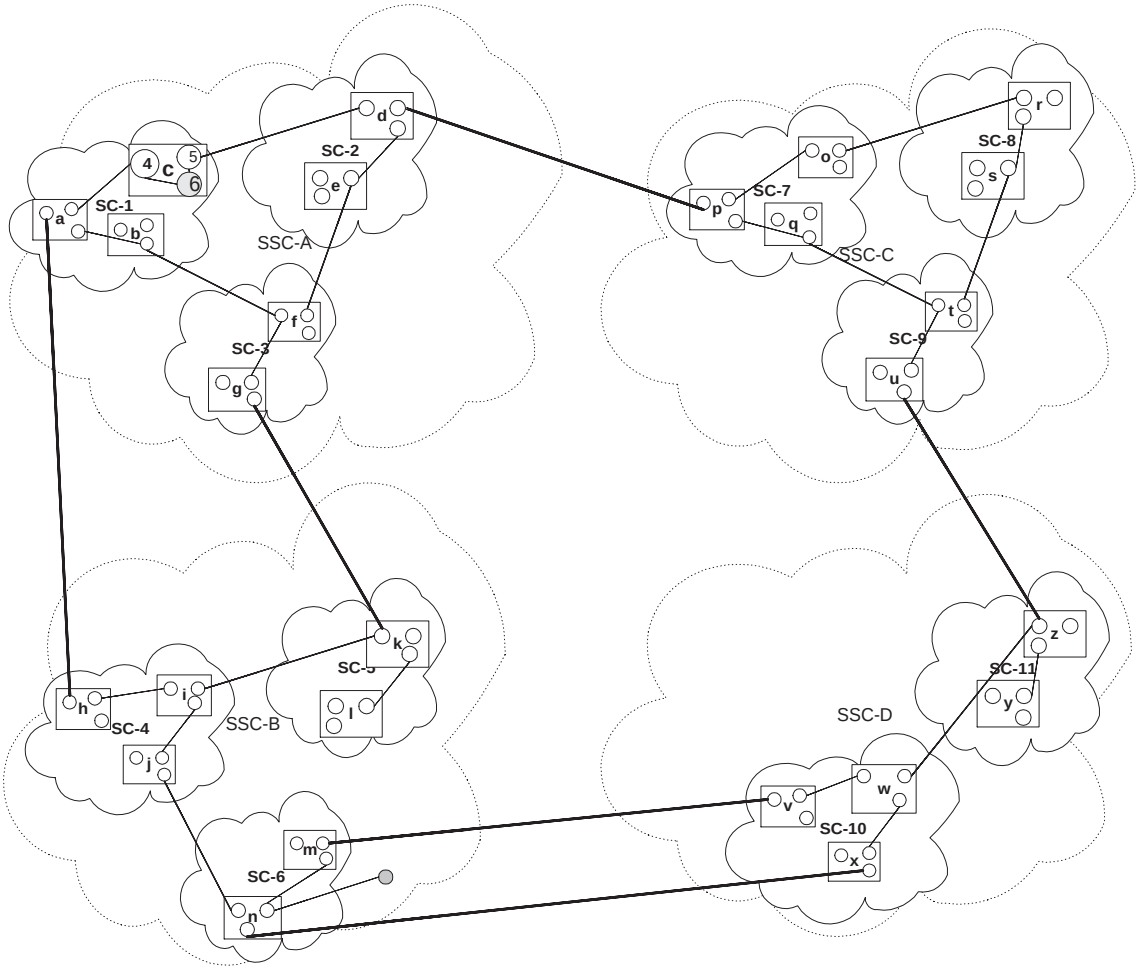

Figure 1: Connectivities between units

In general a context C^ℓ_i at level l exists within the context $C^{\ell+1}_j$ of a level $(\ell+1)$. Connections between nodes within different unit, provide gateways to the other units. We refer to such nodes as *gatekeepers*. A gateway at level ℓ exits within a higher context $C^{\ell+1}_j$ and is denoted $g^\ell_i(C^{\ell+1}_j)$, g^ℓ_i for short. Fig 1 shows a system of 78 nodes organized into a system of 4 super-super-clusters, 11 super-clusters and 26 clusters. In general if a node connects to another node, and the nodes are such that they share the same context $C^{\ell+1}_i$ but have differing contexts C^ℓ_j, C^ℓ_k, the nodes are designated gateways at $level - \ell$

i.e. $g^{\ell}(C^{\ell+1})$. Thus in Fig 1 we have 12 super-super-cluster gateways, 8 super-cluster gateways (6 each in SSC-A and SSC-C, 4 in SSC-B and 2 in SSC-D) and 4 cluster-gateways in super-cluster SC-1.

4 The problem of event delivery

The problem of event delivery pertains to the efficient and reliable delivery of events to the destinations which could be internal or external to the event. In the latter case the system needs to compute the destination lists pertaining to the event. The system merely acts as a conduit to reliably deliver the events from the issuing client to the interested clients.

4.1 Gateway Propagation Protocol - GPP

Gateway Propagation Protocol (GPP) is responsible for the dissemination of connection information within relevant parts of the sub system to facilitate creation of abbreviated system interconnection graphs. Providing precise information for routing of events, and the updating of this information in response to the addition, recovery and failure of gateways is in the purview of GPP.

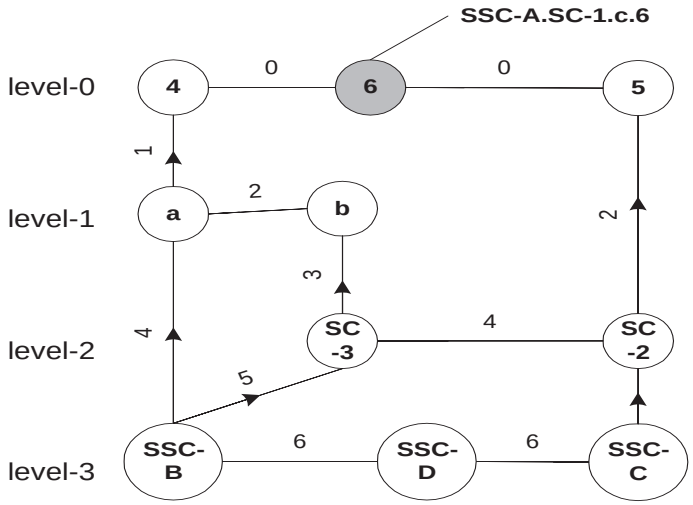

Figure 2: The connectivity graph at node 6.

Figure 2 depicts the connectivity graph that is constructed at the node SSC-A.SC-1.c.6 in Figure 1. The abbreviated system view at each node is different, this reflects the nodes view of the interconnections within the system. Along each edge is the cost associated with traversal along that link. The cost associated with communication between units at different levels increases as the levels of the units increases. One of the reasons we have this cost scheme is that the dissemination scheme employed by the system is selective about the links employed for finer grained dissemination. In general a higher level gateway is more overloaded than a lower level gateway.

4.2 Profile Propagation Protocol - PPP

To snapshot the event constraints that need to be satisfied by an event prior to dissemination within a unit and subsequent delivery at a client we use the Profile Propagation Protocol (PPP). PPP is responsible for the propagation of profile information to relevant nodes within the system to facilitate hierarchical dissemination of events. We use the organization and matching scheme based on the general matching algorithm presented in [ASS+99] of the Gryphon system to organize profiles and compute the destinations associated with the events. Constraints from multiple profiles are organized in the *profile graph*. Associated with every edge we maintain the units that are interested in its traversal. And for

each of these units we maintain the number of predicates $\delta\omega$ within that unit that are interested in the traversal of that edge.

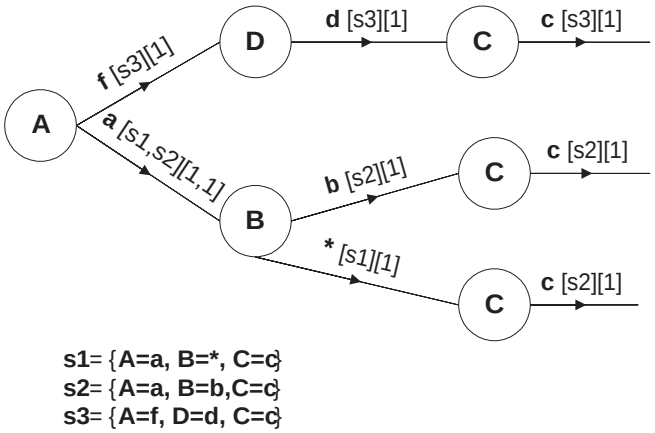

Figure 3: The complete profile graph with information along edges.

In the hierarchical dissemination scheme that we have, gateways $g^{\ell+1}$ compute destination lists for units u^{ℓ} that it serves as a $g^{\ell+1}$ for. A gateway $g^{\ell+1}$ should thus maintain information regarding the profile graphs at each of the units u^{ℓ}. When a profile change occurs at any level, the updates need to be routed to relevant destinations. The connecivity graph provides us with this information. From the connectivity graph in figure 2, it can be seen that node 4 is the cluster gateway thus changes in profiles at level -0 i.e. $\delta\omega^0$ at any of the node are routed to 4. $\delta\omega^1$ changes need to be routed to level two gateways with SSC-1. The way this is calculated is the following -

(a) Locate the level-(ℓ) node in the connectivity graph.

(b) The uplink from this node of the connectivity graph to any other node, indicates the presence of a level-ℓ gateway at that node.

When we send the profile graph information over to the higher level gateways g^{ℓ}, the information contained along the edges in the graph needs to be updated to reflect the nodes logical address at that level. Thus when a node propagates the clients profile to the cluster gateway, it propagates the edges created/removed with the server as the destination. In the figure 2, any $\delta\omega^0$ changes need to be routed to node **4**. Any $\delta\omega^1$ changes at node **4** need to be routed to **5**, and to a node in cluster **5**. Similarly $\delta\omega^2$ changes at node **5** needs to be routed to the level-3 gatekeeper in cluster **a** and superclusters **SC-3**, **SC-2**. When such propagations reach any unit/super-unit the process is repeated till such time that the gateway that the node seeks to reach is reached. Every profile change has a unique-id associated it, with aids in ensuring that the reference count scheme that we have does not fail due to delivery of the same profile change multiple times within the unit.

4.3 Event Routing Protocol - ERP

The Event Routing Protocol (ERP) uses the information provided by PPP to compute hierarchical destinations. Information provided by GPP, such as system inter-connections and shortest paths, are then employed to efficiently disseminate events within the units and to clients subsequently. The problem of routing events is a two pronged problem, which needs to address the basic routing scheme and the routing of real-time events. A gateway $g^{\ell}(C_i^{\ell+1})$ is responsible for the dissemination of events throughout the unit at $level - \ell$ with context $C_i^{\ell+1}$. This is a recursive process and the gateway g^{ℓ} delegates this to the lower level gateways $g^{\ell-1}, \cdots, g^1$ to aid in finer grained dissemination. A gateway g^{ℓ} is concerned with the routing information from $level - \ell$ to $level - N$. When a event has been routed to a gatekeeper g^{ℓ} the routing information associated with the event is modified to reflect the fact that the event was

received at this particular unit. In case the gateway decides to send the event over a gateway g^ℓ, it first checks $level - \ell$ routing information for the event to confirm that theevent was not routed to the unit. If it decides to route the event to that unit over g^ℓ all routing information pertaining to lower level $(\ell, \ell - 1. \cdots, 0)$ disseminations are stripped from the event routing information.

5 Mananging synchronous asynchronous streams

The real time delivery of events when clients are present in the system constitutes the synchronous streams. In our system a client is allowed to have a prolonged dis-connect or roam across to a new location in response to a failure suspicion. When the client joins the system in its new incarnation, it has missed some events during the intervening period. These events need to be recovered for the client. In addition to this, based on the clients profile any new events being issued within the system need to be routed to this client. The system allows for playback of missed events by allowing for events to be seamlessly archived for fault tolerance at different locations within the system.

Routing of these interim events are done based on the last epoch ξ associated with the client, and the replication scheme that exists in the sub system that the client was originally a part of. The replicator nodes construct recovery queues for clients based on events which were to be delivered by clients attached to nodes within units that the node is servicing. The clients then discard events which were not meant for it and for events which were meant for it, reduce the reference count associated with the event.

6 An agent based approach

A client's profile comprises of a set of predicates which the client mandates that a certain event satisfy prior to the client being targetted as one of the destinations for the event. In addition to this associated with the client are a set of properties which could be used to further refine the destinations associated with an event. The refinement process is carried out by a server side agent responsible for further refining the events targetted for a client. This scheme is depcited in figure 4.

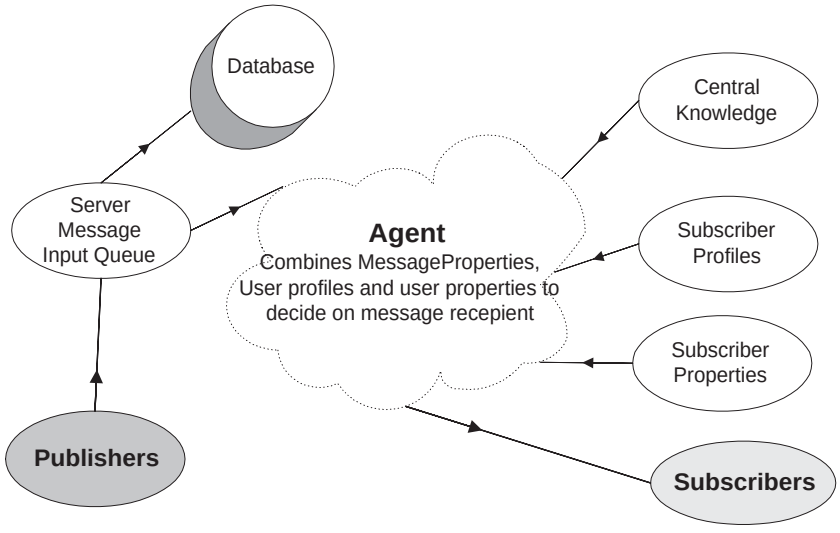

Figure 4: An agent based approach

Traditionally matching of events and calculation of destinations have been based on text properties with SQL like selections or on a static set of *tag-value* pairs contained in the client's profile. JMS employs the earlier approach while most content based pub/sub systems employ the latter approach. We seek to augment this matching process by allowing for topics to be matched to clients based on not just the profiles, but also the properties associated with the client. In addition to the matching based on

string properties or tag-value matching, the advantage of this scheme is that it allows for matching to be based on more dynamic features like the state of the system (bandwidth constraints etc.), a clients content handling capabilities and other similar constraints . This operation is performed by a server side agent which is responsible for this more powerful matching. The decision to route an event is based not only the properties contained in the event, but also on the constraints specified/detected within the user property set.

As an example an event would be routed based on not just the headers describing the event but also on the clients content handling capabilities. Thus we would use the pub/sub matching engine for routing, but we will narrow the destination lists associated with the event based on the clients content handling capabilities. Another example would be of a lecture being in progress, which reuires that interested students should have already completed a pre-requisite successfully. This information is accessed by the agent which can access a client's properties. Even though the event matches the client's profile the event will not be routed to the client if the pre-requisite constraint is not satisfied by the client. This feature could also be employed in collaborative systems, where only certain clients within a session could be allowed to deliver a certain event based onproperties accessed by the server side agent.

6.1 The execution Model - GXOS, MyXos & RDF

To deal with the large volume of objects, the objects need to be made self-defining. We make explicit all the necessary metadata to enable functions such as searching, accessing, unpacking, rendering, sharing, specifying of parameters, and streaming data in and out of them. This metadata is defined using a carefully designed XML schema GXOS and exploiting the new RDF framework. The XML meta-objects point to the location of the object they define and can initiate computations and data transfers on them. Objects can be identified by a URI and referenced with this in either RDF resource links (such as `<rdf:description about="URI" ..`) or fields in the GXOS specification. Three important URI's are the GXOS name such as gndi://gxosroot/halld/users/, and the web location of either the meta-object or object itself. All objects in GXOS must have a unique name specified in a familiar (from file systems) hierarchical syntax.

Events use the same base XML schema as the meta-objects describing the entities in the system. The uniform treatment of events and meta-objects enables us to use a simple universal persistency model gotten by a database client subscribing as a client to all collaborative applications. Integration of synchronous and asynchronous collaboration is achieved by the use of the same publish/subscribe mechanism to support both modes. Hierarchical XML based topic objects matched to XML based subscribing profiles specified in RDF (Resource Description Framework from W3C) control this. Topics and profiles are also specified in GXOS and managed in the same way as meta-objects. These ideas imply new message and event services for the Grid, which must integrate events between applications and between clients and servers. One extension of importance GMSME (GMS Micro Edition) handles messages and events on hand held and other small devices. This assumes an auxiliary (personal) server or adaptor handling the interface between GMS and GMSME and offloading computationally intense chores from the handheld device.

We are currently researching different ways of reading into memory the XML meta-objects as needed by programs running under MyXoS. SAX and DOM XML parsers are not efficient for tens of millions of XML instances at a time. Converting XML schema into Java data structures is possible (http://castor.exolab.org/sourcegen.html) but efficiency requires this be combined with *lazy* parsing so that we expand GXOS trees only as needed to refine our access. We see this as a particularly challenging and having important programming style implications as we look at models where data structures are defined in XML and not directly as C++ or Java classes.

6.2 Performance

References

[AOS+99] Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath. *The Jini Specification.* Addison-Wesley, June 1999.

[ASS+99] 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.

[BCM+99] Gurudutt Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Rob Strom, and Daniel Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, Austin, Texas, May 1999.

[Cor99] Progress Software Corp. Sonicmq :the role of java messaging and xml in enterprise application integration. Technical report, http://www.progress.com/sonicmq, October 1999.

[Cor00a] Firano Corporation. A guide to understanding the pluggable, scalable connection management (scm) architecture - white paper. Technical report, http://www.fiorano.com/ products/fmq5_scm_wp.htm, 2000.

[Cor00b] Talarian Corporation. Everything you need to know about middleware: Mission critical interprocess communication. Technical report, http://www.talarian.com/ products/ smartsockets, 2000.

[CRW00] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, Portland OR, USA, July 2000.

[EE98] Guy Eddon and Henry Eddon. Understanding the dcom wire protocol by analyzing network data packets. *Microsoft Systems Journal*, March 1998.

[HBS99] Mark Happner, Rich Burridge, and Rahul Shrama. Java message service. Technical report, Sun Microsystems, November 1999.

[Hou98] Peter Houston. Building distributed applications with message queuing middleware - white paper. Technical report, Microsoft Corporation, 1998.

[IBM00] IBM. IBM Message Queuing Series. http://www.ibm.com/software/mqseries, 2000.

[Inc00] Softwired Inc. iBus Technology. http://www.softwired-inc.com, 2000.

[iPl00] iPlanet. Java message queue documentation. Technical report, http://docs.iplanet.com/ docs/manuals/javamq.html, 2000.

[Jav99] Javasoft. Java remote method invocation - distributed computing for java (white paper). http://java.sun.com/marketing/collateral/javarmi.html, 1999.

[OMG00a] The Object Management Group OMG. Corba notification service. http://www.omg.org/ technology/documents/formal/notificationservice.htm, June 2000. Version 1.0.

[OMG00b] The Object Management Group OMG. Omg's corba event service. http://www.omg.org/ technology/documents/formal/eventservice.htm, June 2000. Version 1.0.

[OMG00c] The Object Management Group OMG. Omg's corba services. http://www.omg.org/ technology/documents/, June 2000. Version 3.0.

[SA97] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings AUUG97*, pages 243–255, Canberra, Australia, September 1997.

[TIB99] TIBCO. Tib/rendezvous white paper. http://www.rv.tibco.com/whitepaper.html, 1999.