

# The Narada Event Brokering System: Overview and Extensions

Geoffrey Fox  
gcf@indiana.edu

Shrideep Pallickara  
spallick@indiana.edu

Community Grid Labs, Department Of Computer Science  
Indiana University, Bloomington IN-47405

## Abstract

*Narada is a distributed event brokering system designed to run on a large network of cooperating broker nodes. Narada supports heterogeneous client configurations that scale to arbitrary size and incorporates efficient routing algorithms to optimize disseminations to clients. Narada is failure resilient and guarantees delivery in the presence of failures. In this paper we provide an overview of the Narada event brokering system. The paper also describes the Java Message Service compliance process and support for peer-to-peer interactions in Narada. Finally, the paper outlines issues and entry points for supporting Web Services within Narada*

We believe that it is interesting to study the system and software architecture of environments, which integrate the evolving ideas of computational grids, distributed objects, web services, peer-to-peer networks and message oriented middleware. Such peer-to-peer (P2P) Grids should seamlessly integrate users to themselves and to resources, which are also linked to each other. We can abstract such environments as a distributed system of “clients” which consist either of “users” or “resources” or proxies thereto. These clients must be linked together in a flexible fault tolerant efficient high performance fashion. In this paper, we study the messaging or event system – Narada – that is appropriate to link the clients (both users and resources of course) together. For our purposes (registering, transporting and discovering information), events are just messages – typically with time stamps. The event brokering system Narada must scale over a wide variety of devices – from hand held computers at one end to high performance computers and sensors at the other extreme. We have analyzed the requirements of several Grid services that could be built with this model, including computing and education and incorporated constraints of collaboration with a shared event model. We suggest that generalizing the well-known publish-subscribe model is an attractive approach and this is the model that is used in Narada. Industrial strength products in the publish/subscribe domain include solutions like *SmartSockets* [13] from Talarian and *TIB/Rendezvous*

[14] from TIBCO. Related efforts in the research community include Gryphon [15], Elvin [16] and Sienna [17]. The push by Java to include publish subscribe features into its messaging middleware include efforts like JINI [7] and the Java Message Service (JMS) [8]. One of the goals of JMS is to offer a unified API across publish subscribe implementations. JXTA (from *juxta*-position) [19] is a set of open, generalized protocols to support peer-to-peer interactions. Narada is designed as event brokering system to support Community Grids [28] and needs to encompass both P2P and the traditional centralized middle tier style of interactions. This is needed for robustness (since JXTA provides no guarantees and interactions are not reliable), scaling (JMS does not scale) and dynamic resources (since JMS is not natural for very dynamic clients and resources). This paper describes the support for these interactions in Narada. Section 1 of this paper provides an overview of Narada. Sections 2 and 3 describe the rationale and the process of providing JMS compliance and support for JXTA interactions respectively. Section 4 outlines entry points for supporting Web Services within Narada.

## 1.0 Narada

Narada is an event brokering system designed to run on a large network of cooperating broker nodes. Communication within Narada is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. Events are central in Narada and encapsulate information at various levels as depicted in the figure 1. Where, when and how these events reveal their expressive power (at different layers) is what constitutes information flow within the system. Narada deals with the efficient management of this information flow. Clients can create and publish events, specify interests in certain types of events and receive events that conform to specified templates. Client interests are managed and used by the system to compute destinations associated with published events. Clients, once they specify their interests, can disconnect and the system guarantees the delivery of matched events during subsequent reconnects. Clients reconnecting after

prolonged disconnects, connect to the local broker instead of the remote broker that it was last attached to. This eliminates bandwidth degradations caused by heavy concentration of clients from disparate geographic locations accessing a certain known remote broker over and over again. The delivery guarantees associated with individual events and clients are met even in the presence of failures. Furthermore, brokers can fail and these failed brokers need not recover for these guarantees to be met. Details pertaining to the protocol suite in Narada can be found in [4, 5, 6].

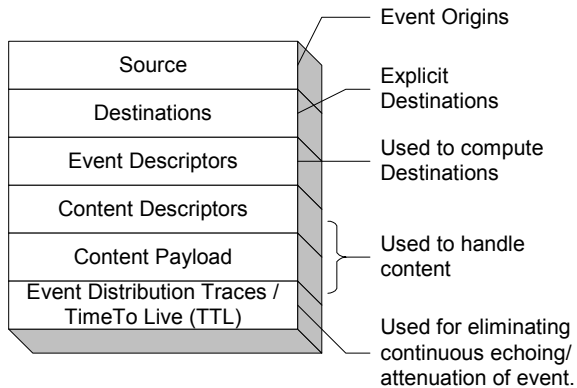


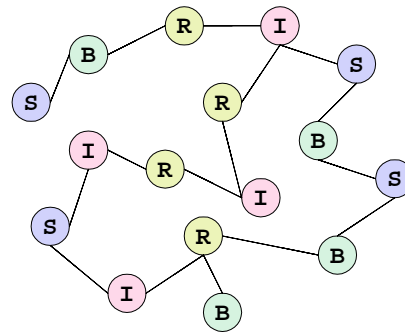
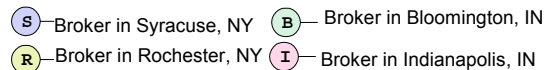
Figure 1: Event in the Narada System

### 1.1 Broker Organization & small world behavior

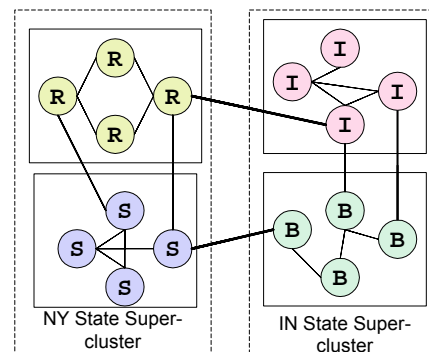
In most systems brokers can be added to the system simply by instantiating the broker process and initiating a connection to one of the brokers within the broker network. Devolving control over these modifications to the network fabric lead to scenarios where broker and connection instantiations result in the network being susceptible to network partitions, poor bandwidth utilizations and inefficient routing strategies, see figure 2.a. To circumvent this, Narada incorporates a broker organization protocol, which manages the addition of new brokers and also oversees the initiation of connections between these brokers. Uncontrolled settings, resulting in a broker network devoid of any logical structure make creation of efficient broker network maps (BNM) an arduous if not impossible task. The lack of this knowledge hampers development of efficient routing strategies, which exploit the broker topology. Such systems then resort to “flooding” the entire broker network, forcing clients to discard events they are not interested in. The node organization protocol incorporates IP discriminators, geographical location, cluster size and concurrent connection thresholds at individual brokers in its decision making process to prevent these situations.

In Narada we impose a hierarchical structure on the broker network (fig 2.b), where a broker is part of a cluster that is part of a super-cluster, which in turn is part

of a super-super-cluster and so on. Clusters comprise strongly connected brokers with multiple links to brokers in other clusters, ensuring alternate communication routes during failures. This organization scheme results in “small world networks” [1,2] where the average communication pathlengths between brokers increase logarithmically with geometric increases in network size, as opposed to exponential increases in uncontrolled settings. This distributed cluster architecture allows Narada to support large heterogeneous client configurations that scale to arbitrary size. Creation of BNMs and the detection of network partitions are easily achieved in this topology. We augment the BNM hosted at individual brokers to reflect the cost associated with traversal over connections, for e.g. intra-cluster communications are faster than inter-cluster communications. The BNM can now be used not only to compute valid paths but also for computing shortest paths. Changes to the network fabric are propagated only to those brokers that have their system view altered. Not all changes alter the BNM at a broker and those that do result in updates to the routing caches, containing shortest paths, maintained at individual brokers.



(a) Bandwidth degradation in an uncontrolled setting



(b) Brokers and connections in a controlled setting

Figure 2: Uncontrolled and controlled settings

## 1.2 Dissemination of events

Every broker serves in two capacities, the first is of course as a conduit for clients to interact with the system. The second role is that of a node within the broker network, responsible for maintaining network snapshots and making intelligent decisions aiding efficient disseminations. Every event has an implicit or explicit destination list, comprising clients, associated with it. The system as a whole is responsible for computing broker destinations (targets) and ensuring efficient delivery to these targeted brokers en route to the intended client(s). Events as they pass through the broker network are updated to snapshot its dissemination within the network. The event dissemination traces eliminate continuous echoing and in tandem with the BNM – used for computing shortest paths – at each broker, is used to deploy a near optimal routing solution. The routing is near optimal since for every event the associated targeted set of brokers are usually the only ones involved in disseminations. Furthermore, every broker, either targeted or en route to one, computes the shortest path to reach target destinations while employing only those links and brokers that have not failed or been failure-suspected.

## 1.3 Failures and Recovery

To deal with failures most systems implicitly invoke the “finite time recovery” (FTR) constraint requiring a failed broker to recover within a finite amount of time. FTR implies brokers have state and force every broker to be up and running at all times. Recovery generally involves state reconstruction from brokers, which the recovering broker had maintained active connections to prior to the failure. This scheme breaks down during multiple neighboring broker failures. Stalling operations for certain sections of the network, and denying service to clients while waiting for failed processes to recover could result in prolonged, probably interminable waits.

In Narada, stable storages existing in parts of the system are responsible for introducing state into the events. The arrival of events at clients advances the state associated with the corresponding clients. Brokers do not keep track of this state and are responsible for ensuring the most efficient routing. Since the brokers are stateless we eliminate FTR. Brokers can fail and remain failed forever. The guaranteed delivery scheme within Narada does not require every broker to have access to a stable store or DBMS. The replication scheme is flexible and easily extensible. Stable storages can be added/removed and the replication scheme can be updated. Stable stores can fail but they do need to recover within a finite amount of time. During these failures the clients that are affected are those that were being serviced by the failed storage.

## 1.4 Support for dynamic topologies

Support for local broker accesses and client roams, along with elimination of FTR provide an environment extremely conducive to dynamic topologies. Brokers and connections could be instantiated dynamically to ensure the optimal bandwidth utilizations. These brokers and connections are added to the network fabric in accordance with rules that are dictated by the agents responsible for broker organization. Brokers and connections between brokers can be dynamically instantiated based on the concentration of clients at a geographic location and also based on the content that these clients are interested in. Similarly average pathlengths for communication could be reduced by instantiating connections to optimize clustering coefficients within the broker network. Brokers can be continuously added or fail and the broker network can undulate with these additions and failures of brokers. Clients could then be induced to roam to such dynamically created brokers for optimizing bandwidth utilization.

## 1.5 Results from the prototype

Figure 3 illustrates some results [4,6] from our initial research where we studied the message delivery time as a function of load. The results are from a system comprising 22 broker processes with the “measuring” subscriber 10 hops away from publisher. The three matching values correspond to the percentages of messages that are delivered to any given subscriber. The 100% case corresponds to systems that would flood the broker network. The system performance improves significantly with increasing selectivity from subscribers. We found that the distributed network scaled well with adequate latency (2 milliseconds per broker hop) unless the system became saturated at very high publish rates. We expect the latency to decrease by a factor of about three in an “optimized production system”.

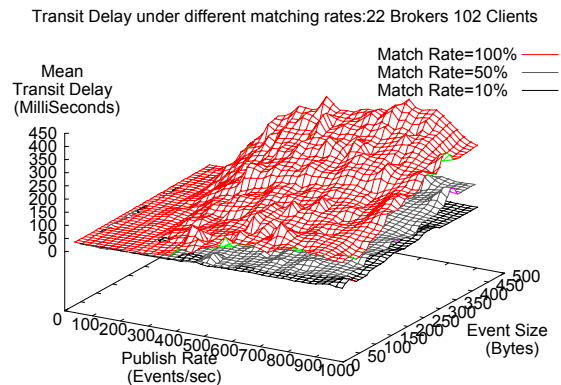


Figure 3: Event transit times in Narada

## 2.0 Providing JMS compliance in Narada

In JMS based systems, applications are developed while conforming to the specification. Various JMS implementations include solutions like SonicMQ [9], openJMS [10] and FioranoMQ [11]. The JMS provider's primary responsibility is dealing with the routing of events and support for reliably routing these events. Applications are designed expecting support for certain operations, and the application logic resides entirely in clients and some functions are built around the communication primitives that the JMS client expects the provider to provide and guarantee.

There are two objectives that we seek to achieve: *Support for JMS-clients within Narada*. Since JMS clients are vendor agnostic, this objective entails JMS providers being replaced transparently by Narada and also for Narada clients to interact with JMS clients. This support for clients conforming to a mature messaging product within the research prototype opens up Narada to a plethora of applications developed around JMS. Furthermore, Narada could then use these applications to further optimize certain most commonly used features exploited by these applications.

1. *To bring Narada functionality to JMS clients/systems developed around it*. This approach (discussed in section 2.1) transparently replaces single server JMS systems with a very large-scale distributed solution.

To complete the JMS compliance, we provided support

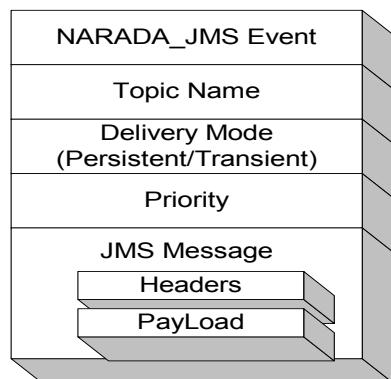


Figure 4: Narada/JMS Event

for the notion of connections, sessions, topics, topic-subscribers and topic-publishers. We also provided support for the operations that can be invoked *on* or *by* any of these entities in addition to the guarantees and the logic, associated and expected with these operations. The matching algorithm [15] used in Narada is augmented with the JMS selector mechanism implemented in openJMS [10]. Finer details pertaining to the integration of JMS within Narada can be found in [12]. JMS messages are encapsulated within the Narada/JMS Event depicted in figure 4. JMS systems where we replaced the JMS provider with Narada include the Anabas distance education conferencing system [38] and the Online Knowledge Center (OKC) [39] developed at IU Grid Labs; both systems were based on SonicMQ.

## 2.1 Distributed JMS Solution

In this approach we aim to replace existing systems built around JMS with the distributed model while entailing minimal changes to the client. In fact, the initialization changes should be identical to those that are required when a JMS provider is changed. Furthermore, the proposed solution should not mandate changes to the Narada core and the associated routing, propagation and destination calculation algorithms. Our goal is to ensure that any JMS based system can benefit from this distributed solution. Thus, applications developed would conform to the JMS specification while the scaling benefits, routing efficiencies, failure resiliency accompanying Narada's distributed solution are all automatically inherited by the integrated solution.

In distributed settings issues pertaining to broker availability and connection overloads, failure suspicions and assorted partitions exist. The network fabric itself would be very fluid with broker and connection additions and failures. Forcing clients to keep track of these changes would expose the client to keeping track of the very large and fluid set of brokers that it could connect to. To circumvent this problem we introduce the notion of *broker locators* whose primary function is the discovery of valid brokers that a client can connect to. Other features include:

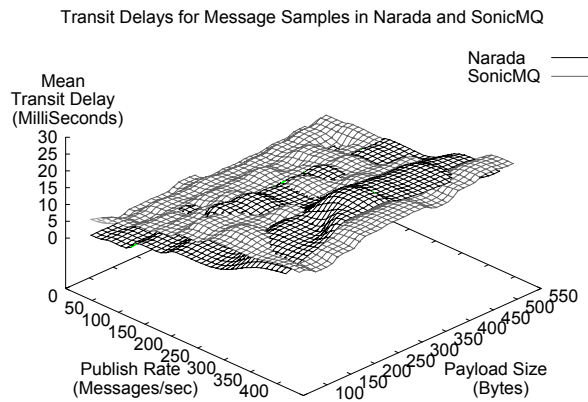
- *Load balancing* – Broker locators keep track of the number of active concurrent connections and the published limit on such active concurrent connections at a broker node. Basing the broker selection on these parameters provides for dynamic load balancing.
- *Incorporation of new brokers* – Newly added brokers are among the best available brokers to handle connection requests. Incorporation of these brokers into the routing fabric is thus very fast.
- *Availability* – The broker locator itself should not constitute a single point of failure neither should it be a bottleneck for clients trying to utilize network services. Since the Narada topology allows brokers to be part of domains there can be multiple broker locators for any given administrative domain.
- *Minimal logic* – Broker locator failures should not affect processing pertaining to any system node.

Once a broker is located, this broker is then used as the conduit through which the client inherits guarantees provided by both Narada and JMS.

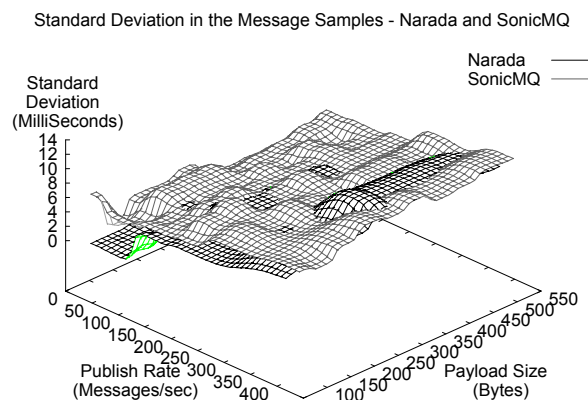
## 2.2 Performance Data for JMS Integration

To gather performance data, we run an instance of the SonicMQ (version 3.0) broker and Narada broker on the same dual CPU (Pentium-3, 1 GHz, 256MB) machine. We then setup 100 subscribers over 10 different JMS

TopicConnections on another dual CPU (Pentium-3, 866MHz, 256MB) machine. There is also a *measuring* subscriber and a publisher that are set up on a third dual CPU (Pentium 3, 866MHz, 256MB RAM) machine. The three machines have Linux (version 2.2.16) as their operating system. The runtime environment for all the processes is Java 2 JRE (Java-1.3.1, Blackdown-FCS).



**Figure 5: Transit Delays - Higher publish rates smaller payloads**



**Figure 6: Standard Deviation - Higher publish rates smaller payloads**

Figure 5 depicts the transit delays for JMS clients under Narada and SonicMQ for high publish rates and smaller payload sizes. Figure 6 depicts the standard deviation associated with message samples under the same conditions. As can be seen Narada compares very well with SonicMQ. Additional results can be found in [12].

### 3.0 Support for Peer-to-Peer interactions

The P2P style interaction model facilitates sophisticated resource sharing environments. The “disruptive” impact of peer-to-peer approaches [18, 20]

has resulted in a slew of powerful applications built around systems such as Jabber [21] and Gnutella [22]. An overview of P2P systems can be found in [24]; discussion on support for P2P interactions in brokering systems can be found in [25]. Systems tuned towards large-scale P2P systems include *Pastry* [23] from Microsoft, which provides an efficient location and routing substrate for wide-area P2P applications. JXTA [19] from Sun Microsystems is another research effort that seeks to provide such large-scale P2P infrastructures. JXTA is a set of open, generalized protocols to support peer-to-peer interactions. It is expected that existing P2P systems would either support JXTA or have bridges initiated to it from JXTA. Support for JXTA would thus enable us to leverage other existing P2P systems.

### 3.1 NARADA JXTA Integration

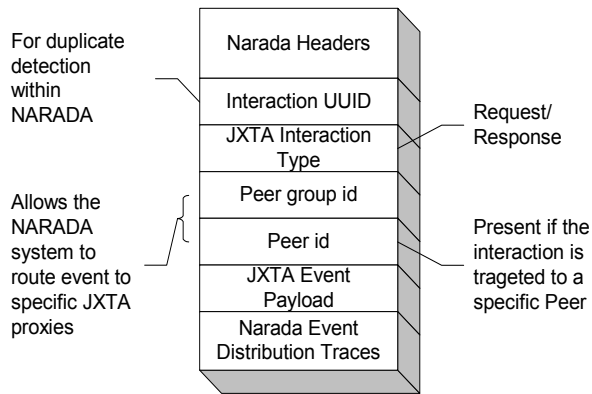
To achieve the integration we ensure the following

- Minimal or zero changes to the Narada system core and the associated protocol suites. Narada merely serves as an effective conduit for JXTA interactions
- Zero changes to the JXTA peers (not rendezvous peers) and the interactions that these peers can have.
- The model should be based on the proxy model, which essentially acts as the go-between the Narada system and JXTA. JXTA peers do not know that Narada is routing some of their interactions.

JXTA interactions that would be routed by the Narada system are fed through the Narada-JXTA proxy, which also serves as a rendezvous peer. JXTA peers can continue to interact with each other and of course some of these peers can be connected to pure JXTA rendezvous peers. These Narada-JXTA proxies, since they are configured as clients within the Narada system, they inherit all the guarantees that are provided to clients within the Narada system. Interactions pertaining to discovery/search or communications within a peer group would be serviced both by JXTA rendezvous peers and also by Narada-JXTA proxies. Interactions that peers have with the Narada-JXTA proxies are what are routed through the Narada system.

To ensure the efficient dissemination of P2P interactions, it is important for to ensure that JXTA interactions that are routed by Narada are delivered only to those Narada-JXTA proxies that should receive them. This entails that the Narada-JXTA proxy perform a sequence of operations, based on the interactions that it receives, to ensure selective delivery. The set of operations that the Narada-JXTA proxy performs comprise gleaning relevant information from JXTA interactions, constructing an event (depicted in figure 7) based on the information gleaned and finally in its role as a Narada client subscribing (if it chooses to do so) to a topic to facilitate selective delivery. By subscribing to

relevant topics, and creating events targeted to specific topics each proxy ensures that the broker network is not flooded with interactions routed by them. The events constructed by the proxies include the entire interaction as the event's payload. Upon receipt at a proxy, this payload is de-serialized and the interaction is propagated as outlined in the proxy's dual role as a rendezvous peer.



**Figure 7: Narada/JXTA Event**

As opposed to the simple forwarding of interactions, the intelligent routing in Narada in tandem with the duplicate detection scheme [26] in our solution ensures faster disseminations and improved communication latencies for peers. Furthermore, targeted peer interactions traversing along shortest paths within the broker network obviate the need for a peer to maintain dedicated connections to a lot of peers. Discovery of rendezvous peers in JXTA is a slow process. A rendezvous peer generally downloads a list of other rendezvous peers from a server, not all of which would be up and running at that time. We allow for dynamic discovery of Narada-JXTA proxies, which need not be directly aware of each other, but do end up receiving interactions sent to a peer group if they had both received peer group advertisements earlier. The scheme also allows us to connect islands of peers and rendezvous peers together, allowing for a greater and richer set of interactions for these clients. Narada, since it is JMS compliant, also opens up possibilities for JMS clients interacting with JXTA peers. Details pertaining to the integration can be found in [26].

#### 4.0 Web services

Since it is unlikely that there will be a single event service standard, using XML for events and messages will prove very important in gaining interoperability on the Grid. As long as there are enough similarities between the event systems, the XML specified messages could be automatically transformed by the use of an event system adapter that can run as Web Service and allow for seamless integration of event services as part of middle tier. We are currently in the design stages for updating

Narada protocols and event interchange to be XML based. We also intend to take a closer look at SIP [27] and its use in instant messenger standards. Web Services are being developed actively by many major companies (Ariba, IBM, Microsoft, Oracle, Sun) with the idea of componentizing Business-to-Business and Business-to-Customers applications. We suggest that a similar approach is useful in both Grid system services but also more generally to develop "Science as a Web Service" – one could term the latter e-Science.

We see WSDL [29,30] – the Web Services Definition Language – as a well thought through proposal. It is incomplete in some ways and more research is needed to decide how best to enhance it in such areas as the integration of multiple services together to form composite systems. WSFL [31] and WSCL [32] are candidate integration standards but another possibility is to build a programming environment on top of basic XML (for data) and WSDL (for methods). Scripts in this environment could specify the integration of services. There are several interesting research projects in this area [33,34]. WSDL has (at least) two important features:

- An XML specification of properties and methods of the Web Service. This is an XML "equivalent" of IDL in CORBA or Java in RMI.
- A distinction between the abstract application interface and its realization gotten by binding to particular transport (such as HTTP) and message (such as SOAP) formats.

The result is a model for Services with ports communicating by messages with a general XML specified structure

It is very important to note that Web Services are and should be composable. However as long as composed services can be exposed through WSDL it does not matter how they were composed. Therefore all those composition mechanisms are really interchangeable by means of WSDL - this is the layer of abstraction that adds up to the robustness of Web Services technology. There are other Web Service technologies - UDDI [35] and WSIL [36] – that are approaches for registering and lookup of such services. This is a critical service but the current approach seems incomplete. The matching of the syntax of Web Service port interfaces is addressed but not their semantics. Further the field of XML meta-data registering and look up is much broader than Web Services. It seems likely that future versions of UDDI should be built on terms of a more general XML Object infrastructure for searching. Possibly developments like the Semantic Web [37] will be relevant. We expect there to be many major efforts to build Web Services throughout a broad range of academic and commercial problem domains. One will define web services in a hierarchical fashion with a set of broadly defined services. JMS could be viewed as a Web Service while JXTA

could be seen as a set of Web services. We are currently in the process of investigating strategies for providing them as Web Services within Narada.

## 5.0 Conclusion

In this paper, we presented an overview of the Narada event brokering system. We also described the process of achieving JMS compliance in Narada, our solution for distributed JMS and support for JXTA interactions. Due to these extensions there are several benefits that can be accrued by clients from these systems. It is our belief that these integrations have added considerable value to Narada and that we are well positioned to exploit the Web Services framework within Narada.

## References

1. D.J. Watts and S.H. Strogatz. Collective Dynamics of Small-World Networks. *Nature*. 393:440. 1998.
2. R. Albert, H. Jeong and A. Barabasi. Diameter of the World Wide Web. *Nature* 401:130. 1999.
3. The Narada Event Brokering System <http://grids.ucs.indiana.edu/ptliupages/projects/narada/>
4. Geoffrey Fox and Shrideep Pallickara, An Event Service to Support Grid Computational Environments, to be published in *Concurrency and Computation: Practice and Experience*, Special Issue on Grid Computing Environments.
5. Geoffrey C. Fox and Shrideep Pallickara, An Approach to High Performance Distributed Web Brokering *ACM Ubiquity* Volume 2 Issue 38. November 2001.
6. Pallickara, S., "A Grid Event Service." PhD Syracuse University, 2001.
7. Ken Arnold, Bryan O'Sullivan, Robert Scheifler, Jim Waldo and Ann Wollrath. The Jini Specification. Addison-Wesley. June 1999.
8. Mark Happner, Rich Burrigge and Rahul Sharma. Sun Microsystems. Java Message Service Specification. 2000. <http://java.sun.com/products/jms>
9. SonicMQ JMS Server <http://www.sonicsoftware.com/>
10. The OpenJMS Project <http://openjms.exolab.org/>
11. Fiorano Corporation. A Guide to Understanding the Pluggable, Scalable Connection Management (SCM) Architecture - White Paper. [http://www.fiorano.com/products/fmq5\\_scm\\_wp.htm](http://www.fiorano.com/products/fmq5_scm_wp.htm)
12. Geoffrey C. Fox and Shrideep Pallickara JMS Compliance in the Narada Event Brokering System. To appear in the proceedings of the *International Conference on Internet Computing* (IC-02), Las Vegas, 2002 .
13. Talarian Corporation. *SmartSockets: Everything you need to know about middleware: Mission Critical Interprocess Communication*. Technical Report: URL: <http://www.talarian.com/products/smartsockets>
14. TIBCO Corporation. TIB/Rendezvous White Paper. URL: <http://www.rv.tibco.com/whitepaper.html>, June 1999.
15. Marcos Aguilera, Rob Strom, Daniel Sturman, Mark Astley and Tushar Chandra. Matching Events in a Content-based Subscription System. *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*. May 1999.
16. Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. *In Proceedings AUUG97*, September 1997.
17. Antonio Carzaniga, David S. Rosenblum and Alexander L. Wolf. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. *Proceedings of 19th ACM Symposium on Principles of Distributed Computing*, July 2000.
18. "Peer-To-Peer: Harnessing the Power of a Disruptive Technology", edited by Andy Oram, O'Reilly Press March 2001.
19. Sun Microsystems. The JXTA Project and Peer-to-Peer Technology <http://www.jxta.org>
20. openp2p P2P Web Site from O'Reilly <http://www.openp2p.com>.
21. Jabber <http://www.jabber.org>
22. Gnutella. <http://gnutella.wego.com>
23. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proceedings of Middleware* 2001.
24. Geoffrey Fox, "Peer-to-Peer Networks," *Computing in Science & Engineering*, vol. 3, no. 3, May2001.
25. Geoffrey Fox and Shrideep Pallickara, Support for Peer-to-Peer Interactions in Web Brokering Systems. To appear in *ACM Ubiquity* 2002.
26. Geoffrey Fox, Shrideep Pallickara, Xi Rao, Pei Qinglin. A Scalable Event Infrastructure for Peer-to-Peer Grids.
27. SIP Session Initiation Protocol standard <http://www.ietf.org/html.charters/sip-charter.html>
28. Geoffrey Fox, Ozgur Balsoy, Shrideep Pallickara, Ahmet Uyar, Dennis Gannon, Aleksander Slominski. Community Grids. *Proceedings of the International Conference on Computational Science* Amsterdam April 2002.
29. Web Services Description Language (WSDL) 1.1 <http://www.w3.org/TR/wsdl>.
30. Presentation on Web Services by Francesco Curbera of IBM at DoE Components Workshop July 23-25, 2001. Livermore, CA. [http://www.llnl.gov/CASC/workshops/components\\_2001/viewgraphs/FraniscoCurbera.ppt](http://www.llnl.gov/CASC/workshops/components_2001/viewgraphs/FraniscoCurbera.ppt)
31. Frank Laymann (IBM), Web services Flow Language WSFL, <http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>
32. Harumi Kuno et al (WSCL – Web services Conversational Language), "Conversations+ Interfaces == Business logic", <http://www.hpl.hp.com/techreports/2001/HPL-2001-27.html>
33. IPG Group at NASA Ames, CRADLE Workflow project.
34. Mehrotra, P and colleagues. Arcade Computational Portal. <http://www.cs.ou.edu/~ppvm>
35. Universal Description, Discovery and Integration Project UDDI, <http://www.uddi.org/>
36. Peter Brittenham, An Overview of the Web Services Inspection Language (WSIL), <http://www-106.ibm.com/developerworks/webservices/library/ws-wslover/>
37. Berners-Lee, T., Hendler, J., and Lassila, O., "The Semantic Web," *Scientific American*, May2001.
38. The Anabas Conferencing System. <http://www.anabas.com>
39. The Online Knowledge Center (OKC) Web Portal <http://judi.ucs.indiana.edu/okcportal/index.jsp>