Deploying the NaradaBrokering Substrate in Aiding Efficient Web & Grid Service Interactions

Geoffrey Fox and Shrideep Pallickara

Abstract— **NaradaBrokering has been developed as the messaging infrastructure for collaboration, peer-to-peer and Grid applications. The value of NaradaBrokering in the context of Grid and Web services has been clear for some time. NaradaBrokering – combined with further extensions to, and testing of, its existing capabilities – can also take advantage of the maturing of Web Service standards and specifications to build very powerful general mechanisms to deploy and integrate it with general Web services. This paper describes a framework to integrate the NaradaBrokering substrate with Web Services.**

Index Terms— **distributed middleware, grid services, robust delivery, service oriented architectures and Web Services**

I. INTRODUCTION

NaradaBrokering has been developed as the messaging infrastructure for collaboration, peer-to-peer and Grid applications. It has undergone extensive functional testing in collaborative sessions and extensive performance measurements have been made in a variety of configurations including cross-continental applications. The value of NaradaBrokering in the context of Grid and Web Services has been clear for some time. NaradaBrokering provides a messaging abstraction that allows it to provide message-related capabilities in a transparent fashion. These capabilities include message-based security and associated encryption, time and causal ordering, compression, virtualization of transport protocol and addressing, and fault tolerance related functionalities. NaradaBrokering – combined with further extensions to, and testing of, its existing capabilities – can also take advantage of the maturing of Web Service specifications to build very powerful general mechanisms to deploy and integrate it with general Web Services.

In particular we exploit WS-Addressing and the SOAP processing stack to interface NaradaBrokering with Web Services. The strategy described in this paper will allow new,

Manuscript received April 8, 2004.

Geoffrey Fox is with the Department of Computer Science, School of Informatics and the Community Grids Lab all at Indiana University (phone 812-856-7977, fax: 812-856-7972, e-mail: gcf@indiana.edu).

Shrideep Pallickara is with the Community Grids Lab at Indiana University (phone 812-856-1311, fax: 812-856-7972, e-mail: spallick@indiana.edu).

and existing, applications built around the Web Services Framework to leverage capabilities offered by the NaradaBrokering substrate. This will entail no changes (code rewrites, recompilation etc) to the service implementations while at the same time allowing these services to leverage capabilities and functionalities provided by the NaradaBrokering substrate.

This paper is organized as follows. In section 2 we provide an overview of the NaradaBrokering substrate. In section 3 we introduce the concept of the substrate managing services. In sections 4 and 5 we include details of the ongoing effort to incorporate SOAP support into NaradaBrokering. In section 5 we provide an overview of related work in the area of distributed publish/subscribe and peer-to-peer systems. Finally, in section 6 we outline our conclusions.

II. NARADABROKERING SUBSTRATE

NaradaBrokering [1-8] is a distributed messaging infrastructure and provides two closely related capabilities. First, it provides a message oriented middleware (MoM) which facilitates communications between entities (which includes clients, resources, services and proxies thereto) through the exchange of messages. Second, it provides a notification framework by efficiently routing messages from the originators to only the registered consumers of the message in question. The smallest unit of this *substrate* should be able to intelligently process and route messages, while working with multiple underlying communication protocols. We refer to this unit as a *broker*, where we avoid the use of the term *server*s to distinguish it clearly from the application servers.

Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized messages, which we call *events*. Events can encapsulate information pertaining to transactions, data interchange, method invocations, system conditions and finally the search, discovery and subsequent sharing of resources. NaradaBrokering places no constraints either on the size, rate and scope of the interactions encapsulated within these events, or on the number of entities present in the system. Events encapsulate expressive power at multiple levels. Where, when and how these events reveal their expressive power is what constitutes information flow. NaradaBrokering manages this information flow.

A. Broker Organization

Uncontrolled broker and connection additions result in a broker network susceptible to network-partitions and devoid of any logical structure thus making the creation of efficient broker network maps (BNM) an arduous if not impossible task. The lack of this knowledge hampers the development of efficient routing strategies, which exploit the broker topology. Such systems then resort to *flooding* the entire broker network, forcing entities to discard events that they are not interested in.

In NaradaBrokering we impose a hierarchical structure on the broker network [2,3], 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. Such a scheme ensures that 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-based architecture allows NaradaBrokering to support large heterogeneous client configurations.

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, 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 broker network view altered.

B. Dissemination of events

An event comprises of headers, content descriptors and the payload encapsulating the content. An event's headers provide information pertaining to the type, unique identification, timestamps, dissemination traces and other quality of service (QoS) related information pertaining to the event. The content descriptors and the values these content descriptors take collectively comprise the event's *content synopsis*. Entities within the system can register their interests by specifying constraints on the event's synopsis. The destinations associated with an event are computed based on the registered interests and the event's synopsis. In NaradaBrokering this synopsis could be based on tag-value pairs, Integers and Strings. Entities can also specify SQL queries on properties contained in a specialized message. The synopses could also be XML documents, in which case XPath constraints can be specified. More recently support for regular expression queries on an event's content synopsis is also available.

Every event has an implicit or explicit destination list, comprising entities, associated with it. The brokering system as a whole is responsible for computing broker destinations (targets) and ensuring efficient delivery to these targeted brokers en route to the intended entity(s). Events as they pass through the broker network are updated to snapshot its dissemination within the broker network which eliminates continuous echoing due to loops in connectivity between the nodes. This prevents a node from routing the same event twice. The BNM at individual brokers is used to compute best broker hops to reach target brokers. The routing is very efficient [4] since for every event, the associated targeted 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 eschewing links and brokers that have failed or have been failure-suspected.

1) Profiling the matching engines

The matching engine is responsible for computing destinations associated with an event based on the profiles available at a node. Depending on the type of applications, standards, events and subscriptions that need to be supported there would be multiple matching engines residing within every processing broker node. We now provide some results pertaining to the matching engines residing in brokers within NaradaBrokering. These results are for stand-alone processes, where we computed the matching times as a function of the number of subscriptions maintained. In each case, an event is matched to retrieve every matching subscription. For every matching engine, the number of subscriptions is varied from 10,000 to 100,000. The results were measured on a machine (1GHz,256MB RAM) running the process in a Java-1.4 Sun VM with a high-resolution timer for computing delays.

The richer the constraints, the greater the CPU-cost associated with the matching process. As can be seen the average delays for matching increases progressively (String to SQL to XPath in that order) as the complexity of the matching increases. For String based matching, as depicted in Figure 1, the average delay for matching subscriptions generally increases as the size of the topic String increases. The increase in delays for matching as the topic String size doubled from 16 to 32 was in the range of 1 microsecond. The results demonstrate that it is feasible to have real time interactions that are based on the simple String constraints.
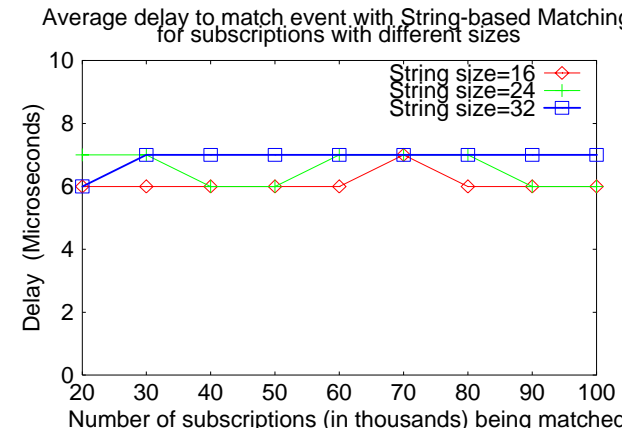

Figure 1: Plots for String based matching

Figure 2 contrasts the costs involved in matching JMS events to stored SQL-92 based selectors on the properties contained within the JMS message and XML events to stored XPath conforming constraints. Of course these costs can vary significantly depending on the type of the query. For our experiments we used XPath and SQL queries, which we felt were comparable. The cost of a single matching operation

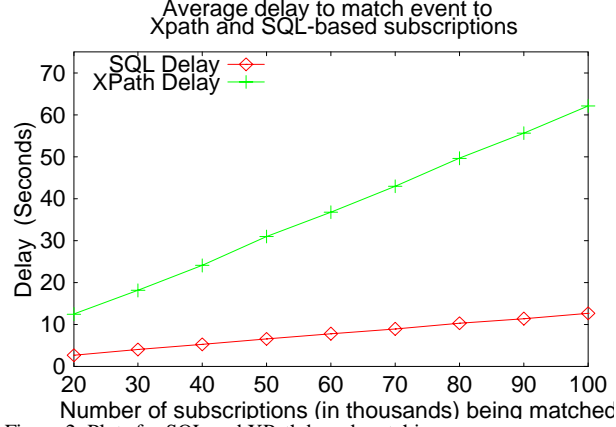involving an XML event and an accompanying XPath query is around 3 milliseconds.
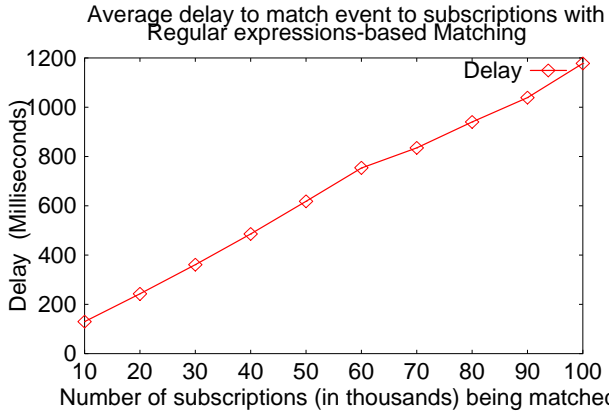

Figure 2: Plots for SQL and XPath based matching

Figure 3: Regular expressions based matching

Figure 3 depicts the costs involved in matching an event to regular expression based constraints. The subscriptions stored were in the following format: [Tt]he [Qq]uick [Bb]rown [Ff]ox [Jj]umps while the topic was of the form "The quick brown fox jumps upon the lazy dog". The matching costs varied from 130 milliseconds for 10000 subscriptions to 1178 milliseconds for 100000 subscriptions. Comprehensive results for other NaradaBrokering matching engines and a discussion regarding the implications of these matching engines can be found in Ref [4].

C. Services within NaradaBrokering

In NaradaBrokering entities can also specify constraints on the Quality of Service (QoS) related to the delivery of events. The QoS pertain to the reliable delivery, order, duplicate elimination, security and size of the published events and their encapsulated payloads. NaradaBrokering provides reliable delivery [5] of events to authorized/registered entities. The delivery guarantee is satisfied in the presence of both link and node failures. Entities are also able to retrieve events that were missed during failures or prolonged disconnects. The scheme also facilitates exactly-once ordered delivery of events.

1)   *The reliable delivery scheme and achieving exactly-once-delivery*

The NaradaBrokering substrate's reliable delivery guarantee holds true in the presence of four conditions.

1. Broker and Link Failures: The delivery guarantees are satisfied in the presence of individual or multiple broker and link failures. The entire broker network may fail. Guarantees are met once the broker network (possibly a single broker node) recovers.
2. Prolonged Entity disconnects: After disconnects an entity can retrieve events missed in the interim.
3. Stable Storage Failures: The delivery guarantees must be satisfied once the storage recovers.
4. Unpredictable Links: Events can be lost, duplicated or re-ordered in transit over individual links.

To ensure the reliable delivery of events (conforming to a specific template) to registered entities three distinct issues need to be addressed. First, there should be exactly one Reliable Delivery Service (RDS) node that is responsible for providing reliable delivery for a specific event template. Second, entities need to make sure that their subscriptions are registered with RDS. Finally, a publisher needs to ensure that any given event that it issues is archived at the relevant RDS. In our scheme we make use of both positive (ACK) and negative (NAK) acknowledgements. We may enumerate the objectives of our scheme below.

*   Storage type: Underlying storages could be based on flat files or relational/XML databases.
*   RDS instances: There could be multiple RDS instances. A given RDS instance can manage reliable delivery for one or more templates.
*   Autonomy: Individual entities can manage their own event templates. This would involve provisioning of stable storage and authorization of entity constraints.
*   Location independence: A RDS node can be present anywhere within the system.
*   Fast Recovery schemes: The recovery scheme needs to efficiently route missed events to entities.

a)   *The Reliable Delivery Service (RDS)*

RDS can be looked upon as providing a service to facilitate reliable delivery for events conforming to any one of its managed templates. To accomplish this RDS provides four very important functions. First, RDS archives all published events that conform to any one of its managed templates. Second, for every managed template, RDS also maintains a list of entities for which it facilitates reliable delivery. RDS may also maintain information regarding access controls, authorizations and credentials of entities that generate or consume events targeted to this managed template. Entity registrations could either be user controlled or automated.

Third, RDS also facilitates calculation of valid destinations for a given template event. This is necessary since it is possible that for two events conforming to the same template, the set of valid destinations may be different. RDS maintains a list of the profiles and the encapsulated constraints (subscriptions) specified by each of the registered entities. For

each managed template the service also hosts the relevant matching engines, which computes entity destinations from a template event's *synopsis* (content descriptor). Finally, RDS keeps track not only of the entities that are supposed to receive a given template event, but also those entities that have not explicitly acknowledged receipt of these events.

RDS also archives information pertaining to the addition, removal and update of constraints specified by registered entities. For every archived event or entity profile related operations, RDS assigns monotonically increasing sequence numbers. These sequence numbers play a crucial role in error detection and correction, while also serving to provide audit trails.

Publishing entities make use of *companion events* and a series of *negotiations* to ensure delivery of published events to the relevant RDS. Publishing entities need information regarding generation of companion events. This information is maintained both at the publishing entity and RDS. During recovery this information can be retrieved from RDS.

Entities within the system use *invoice events* (which can encapsulate both ACKs and NAKs) to detect and fix errors in delivery sequences. Associated with every entity within the system is an epoch. This epoch is advanced by RDS and corresponds to the point up until which that entity has received events. Since the epoch can be used to determine order and duplicate detection it is easy to ensure guaranteed exactly once delivery of events.

b)   Experimental Results

We performed two sets of experiments involving a single broker and three brokers. In each set we compared the performance of NaradaBrokering's reliable delivery algorithms with the best effort approach in NaradaBrokering. Furthermore, for best effort all entities/ brokers within the system communicate using TCP, while in the reliable delivery approach we had all entities/brokers within the system communicate using UDP.
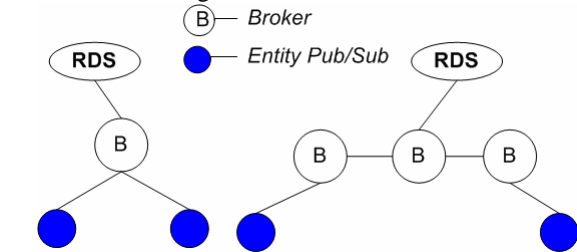

Figure 4: Experimental Setups

The experimental setups are depicted in Figure 4. The lines connecting entities signify the communication paths that exist between the entities; this could be a connection oriented protocol such as TCP or a connection less one such as UDP.The publishing/subscribing entities (hosted on the same machine to accounting for clock synchronizations and drifts), brokers and RDS are all hosted on separate machines (1GHz, 256MB RAM) with each processes running in a JRE-1.4 Sun VM. The machines involved in the experimental setups reside on a 100 Mbps LAN. Currently, in the Reliable Delivery Service (RDS) node we support flat-file and SQL based archival. The results reported here are for a scheme where the

RDS utilizes MySQL 4.0 for storage operations. We found that the archival overheads were between 4-6 milliseconds for payloads varying from 100 bytes to 10 KB.
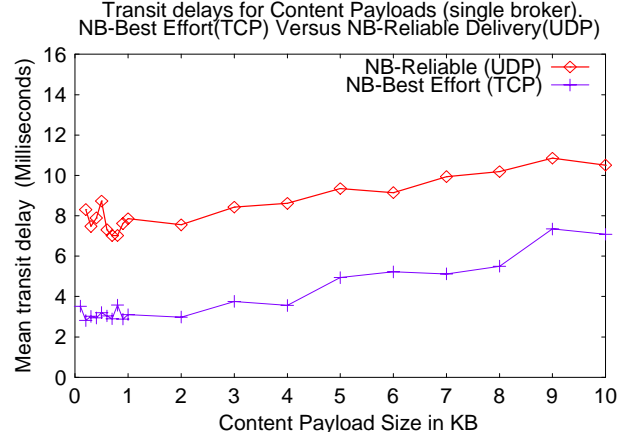

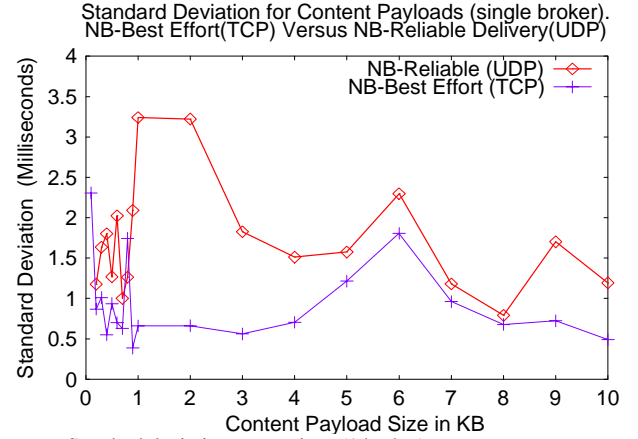Figure 5: Transit Delay comparison (single broker)

Figure 6: Standard deviation comparison (1 broker)

We computed the delays associated with the delivery of best-effort and reliable delivery schemes. The results reported here for the reliable delivery case correspond to the strongest case where the event is not delivered unless the corresponding archival notification is received. Figure 5 and Figure 6 depict the transit delay and standard deviation associated with a single broker network, while Figure 7 and Figure 8 depict the same for the 3 broker network.
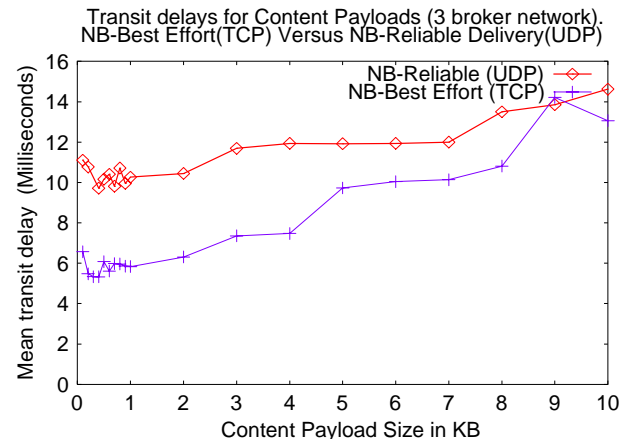

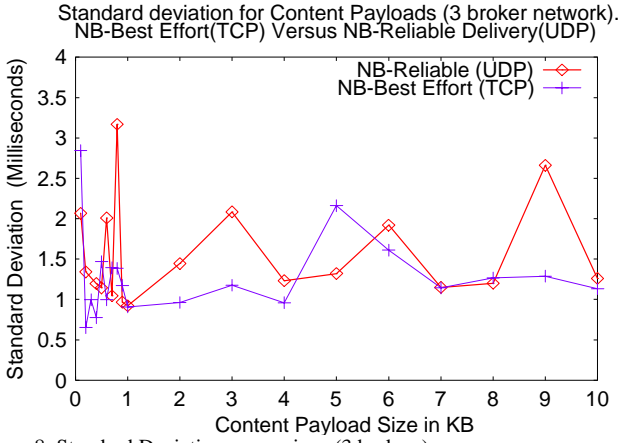Figure 7: Transit Delay comparison (3 brokers)

Figure 8: Standard Deviation comparison (3 brokers)

In the reliable delivery case there is an overhead of 4-6 milliseconds (depending on payload size) associated with the archival of the event, with an additional variable delay of 0-2 milliseconds due to *wait()-notify()* statements in the thread which triggers archival. These factors, in addition to retransmissions (NAKs) triggered by the subscribing entity due to lost packets, contributed to higher delays and higher standard deviations in the reliable delivery case. It should be noted that we can easily have an optimistic delivery scheme which does not wait for archival notifications prior to delivery. This scheme would then produce overheads similar to the best effort case.

2)   Dealing   with   large   payload   sizes: Compression/Fragmentation
To deal with events with large payloads, NaradaBrokering provides services for compressing and decompressing these payloads. Additionally there is also a fragmentation service which fragments large file-based payloads into smaller ones. A coalescing service then merges these fragments into the large file at the receiver side. This capability in tandem with the reliable delivery service was used to augment GridFTP to provide reliable delivery of large files across failures and prolonged disconnects. The recoveries and retransmissions involved in this application are very precise.

Here, we had a proxy collocated with the GridFTP client and the GridFTP server. This proxy, a NaradaBrokering entity, utilizes NaradaBrokering's fragmentation service to fragment large payloads (> 1 GB) into smaller fragments and publish fragmented events. Upon reliable delivery at the server-proxy, NaradaBrokering reconstructs original payload from the fragments and delivers it to the GridFTP server. Additional details can be found in Ref [6].

3)   Time and Buffering Services
The substrate also includes an implementation of the Network Time Protocol (NTP). The NaradaBrokering time service allows NaradaBrokering processes (brokers and entities alike) to synchronize their timestamps using the NTO algorithm with multiple time sources (usually having access to

atomic time clocks) provided by various organizations, like NIST and USNO.

The NaradaBrokering time service plays an important role in collaborative environments and can be used to time order events from disparate sources. The substrate includes a buffering service which can be used to buffer replays from multiple sources, time order these events and then proceed to release them.

4)   Performance Monitoring Services
Connections originating from a broker are tracked by a monitoring service. The factors measured on individual links include loss rates, standard deviations and jitters. This can then be used to augment the weights associated with edges in the BNMs to facilitate real-time responses, by the routing algorithms, to changing network conditions.

5)   Security Service
The NaradaBrokering security framework [7] provides a scheme for end-to-end secure delivery of messages between entities within the system. The scheme protects an event in its traversal over multiple, possibly insecure, transport hops. Topics have symmetric keys associated with them, which are distributed to authorized entities using the entity's personal public-key. Entities use these symmetric keys to secure events and use their personal asymmetric keys for signing the digests computed from the payloads. Entities can verify the integrity and source of these events, before proceeding to process the encrypted payload.

D.   The transport framework
NaradaBrokering incorporates an extensible transport framework and virtualizes the channels over which entities interact with each other. Entities are thus free to communicate across firewalls, proxies and NAT boundaries which can prevent interactions from taking place. Furthermore, NaradaBrokering provides support for multiple transport protocols such as TCP (blocking and non-blocking), parallel TCP streams, UDP, Multicast, SSL, HTTP and RTP. The typical delays involved with NaradaBrokering's transport framework in LAN settings is around a 1 millisecond. Additional information and measurements regarding the transport framework can be found in Ref [8].
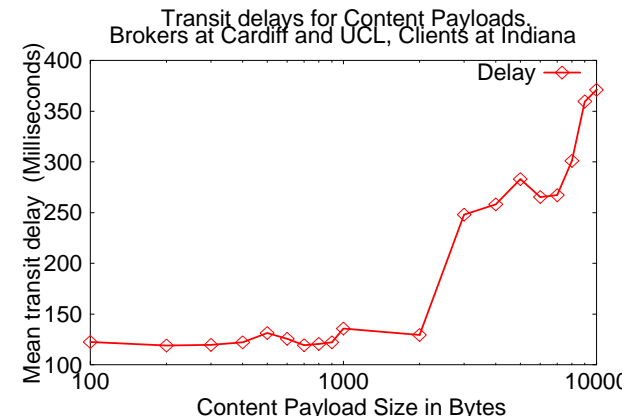

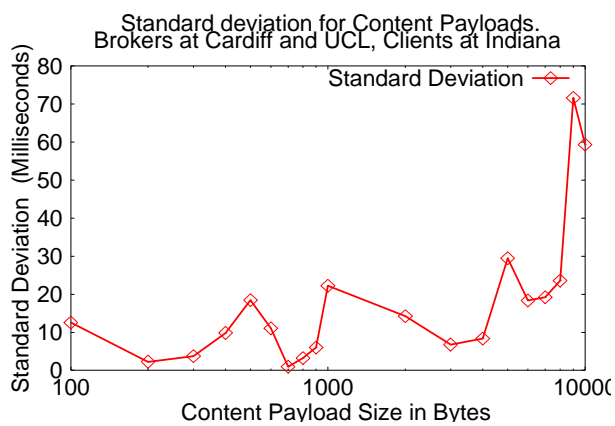Figure 9: Transit delay for message samples (UCL, Cardiff and IU)

Figure 10: Standard deviation for message samples (UCL, Cardiff and IU)

In an experiment involving performance of the NaradaBrokering transport framework over trans-Atlantic links we had machines from the University College London (UCL), Cardiff University and Indiana University involved in the set up. A broker network comprising two brokers, one each at UCL and Cardiff was set up. The machine at UCL was a SPARC Ultra-5 running SunOS 5.9, while the one at Cardiff was a 1 GHz Pentium-III with a 256MB RAM running Linux. The machine at Indiana hosting the publishing/receiving clients was a 1.5 GHz AMD with 256 MB RAM running Linux. The JVM for all processes was 1.4.1. Figure 9 and Figure 10 depict the mean transit delay and standard deviation associated with message samples involved in individual test cases (each comprising 50 messages). The results varied from 122 milliseconds for 100 bytes to 371 milliseconds for 10 KB messages. The delays increased with increase in payload sizes. The standard deviation was also higher at higher payload sizes.

E.   Applications using NaradaBrokering

1)   Audio-Video Conferencing applications
The GlobalMMCS project (http:.//www.globalmmcs.org) provides an audio/video conferencing environment that allows participants with different codecs/protocols such as H.323, T.120 and Access Grid to efficiently collaborate with each other. GlobalMMCS uses NaradaBrokering for delivering control packets and media streams in real-time. Experimental results have demonstrated that a single broker is capable of efficiently handling about 200 concurrent video streams.

2)   Grid Applications
More recently we have integrated GridFTP with the NaradaBrokering system. The advantage of this approach is that the source code of the GridFTP client need not be changed and existing client code can be used with NB system without any modification. Due to its integration with the NaradaBrokering system, files can be recovered after failures or even prolonged disconnects. Files can be partially transmitted and if there is a failure the transmission begins at a point closer to where the failure occurred. The latest release of NaradaBrokering also includes a bridge to Globus Toolkit-3.

3)   Collaborative applications
NaradaBrokering has been deployed in real-time settings by providing back end support for the Anabas (http://www.anabas.com) conferencing software running in JMS emulation mode. Several online seminars have been conducted using this Anabas-NaradaBrokering combination and the number of users collaborating concurrently has been in the excess of 30 users several times. Anabas uses NaradaBrokering to support shared display, text chats and all session control functions. Please note that shared display involves sharing large frame buffers– sometimes in the order of multi-megabits.

The SVGArena (http://www.svgarena.org) project has used NaradaBrokering to provide collaborative and peer-to-peer capabilities within shared Scaleable Vector Graphics (SVG) browser environments. Games such as Solitaire and Chess that have been developed as part of this project have used NaradaBrokering as the underlying messaging infrastructure.

4)   Mobile computing
The CAROUSEL project at Indiana University (http://grids.ucs.indiana.edu/ptliupages/projects/carousel/) has developed a protocol that provides seamless access from mobile devices to NaradaBrokering. This has been registered as a transport in the NaradaBrokering substrate, which allows these devices to interact with each other in collaborative settings.

III.   SERVICE ORIENTED ARCHITECTURES AND THE NARADABROKERING SUBSTRATE

The emerging Web Services stack comprising XML – the lingua franca of the various standards, SOAP [9] and WSDL [10] have facilitated sophisticated interactions between services. Service definitions are XML-based descriptions in WSDL.

Interactions, invocations and data-interchange are encapsulated in SOAP messages. Services can be implemented in different languages and can bind to multiple transports. The XML-based specifications pertain to exposing services, discovering them and accessing these securely once the requestor is authenticated and authorized. These specifications allow interactions between service requestors and services running on different platforms, containers and implemented in different languages.

As Web Services have matured the interactions that the services have between themselves have gotten increasingly complex and sophisticated. Web Services can be composed easily from other services, and these services can be made to orchestrate with each other dynamically. Web Services specifications have addressed issues such as security, trust, notifications, service descriptions, advertisements, discovery and invocations among others. These specifications can leverage, extend and interoperate with other specifications to facilitate incremental addition of features and capabilities. In some cases there are competing specifications, e.g. in the reliable messaging area. These competing specifications tend to rely on the same subset of specifications in the Web Services stack to achieve its objectives. Forcing the user to

make these decisions/choices can be quite complicated and error-prone. It is desirable to automate several of aspects of these interactions such as the negotiation of standards or specifications (in some cases competing), thus making these interactions reliable, simpler and easier to use.

The substrate permeates the services that are provided. It should be noted that these services could continue to exist in a stand alone mode and be accessed in ways similar to traditional Web Services. In fact because of the nature of the Web Service bindings, a service could continue to be bound to other transports and not have access to certain features.

The substrate need to be able to enable services to interact, discover, compose and utilize services hosted over the substrate. The substrate comprises a large network of broker nodes. The substrate provides services such as guaranteed delivery, ordered delivery, compression, fragmentation and secure end-to-end delivery of events (messages) routed within the substrate.

The proposed scheme is intended to enable Web Services to interact directly with the NaradaBrokering substrate. Here the substrate maintains information regarding the services that can be accessed. The substrate uses this information to locate services. Since the clients/services interact directly with the substrate they have access to all the services provided by the substrate. We may enumerate some of the features of such a system

a) Guaranteed delivery mechanisms within the substrate can facilitate disconnected operations where a client/service can access services even if they are not currently available.
b) The substrate can locate services that are closest (in terms of network latency) and least utilized. Such a scheme facilitates clients/services access service instances that are load-balanced by the substrate.
c) End-to-end secure interactions. The approach within the substrate is consistent with that deployed in WS-Security which relies on message-level security.
d) Inevitably the realms across which entities communicate involve firewalls, proxies and NAT boundaries. The substrate facilitates communications over these boundaries.
e) In the event of large payloads/attachments the substrate provides services such as compression, fragmentation of payload into smaller ones and NaradaBrokering-enhanced GridFTP which would be deployed.

It should be noted that more recently there has been an effort to factor the OGSI [11] functionality to comprise a set of independent Web Service standards. These specifications align OGSI with the consensus emerging from the Web Services Architecture working group of the World Wide Web Consortium. The specifications that comprise the new proposed framework – the WS-Resource Framework (WSRF) [12] – leverages, and can leverage, specifications in the Web Services area such as, but not limited to, authentication, transactions, reliable messaging and addressing among others. WSRF specification also includes WS-Notification [13] which abstracts notifications using a topic based publish/subscribe mechanism.

Throughout our discussions when we refer to services as Web Services the term refers to services that are parts of the Service Oriented Architecture, and includes both Web Services and Grid Services (which is also a Web Service). The strategies that we discuss through this paper are thus applicable to both the domains – traditional business oriented Web Services and the science/traditionally-academia oriented Grid Services architecture. Thus the discussion on load balancing service instances (in a section 5) would be valid for managing stateful resources exposed using the WS-Resource specification.

IV. APPROACHES TO INTERACTING WITH WEB SERVICES

In this section we describe approaches to incorporating support for Web Services within the NaradaBrokering substrate. A simple approach involves using a NaradaBrokering-proxy that acts as an interface between services and the substrate. A sophisticated and far more powerful approach involves incorporating SOAP processing into the substrate. This enables the substrate to function as a SOAP intermediary. Here we also propose the development of an end-point NaradaBrokering "plug-in" that can be used by a Web Service to provide direct connectivity to the NaradaBrokering network. Since the plug-in resides as a handler within the handler-chain associated with the SOAP processing stack at a service endpoint, no changes are needed to either the service implementations or the service requestors. This involves a one-time effort of writing NaradaBrokering handlers for SOAP implementations in different languages such as Apache Axis (such a handler can be used with Sun's JAX-RPC with no changes), gSOAP and Soap::Lite. These handlers are referred to by different names in different systems. For example, gSOAP refers to them as plug-in whereas in .NET they are referred to as SOAP extensions. It should be noted that the schemes outlined in the earlier section, using either the proxy approach or processing based on SOAP messaging, should be able to interoperate with each other.

A. Proxy Approach

This approach is based on SOAP and involves using the proxy architecture to deploy Web Services within the system. Here a client's service invocation, contained in a SOAP message, is intercepted by the proxy. This SOAP message is then encapsulated in a native NaradaBrokering event and the substrate routes it to the proxy associated with the service instance. This proxy then recreates the SOAP message from the native NaradaBrokering event and forwards the SOAP message to the Web Service. Faults and responses generated by the Web Service are routed back using the same principles which govern the invocation scheme.

This approach is a simple one and the costs (specifically network cycles) associated with the additional proxy redirect can be alleviated by collocating the proxy on the same machine as the client and server. This approach has two clear advantages –

1. It requires no change to either the original service or the container hosting that service
2. It can be used to facilitate interactions between generic services (perhaps a non Web Service approach such as IIOP or native Java) and services based on Web Service standards.

This solution however is application dependent and the proxy needs to be rewritten or tweaked for different applications. Also, since only the proxies are interacting with the substrate all the QoS provided by the substrate are accessible only to these proxies and not to the Web Service.

B. Incorporating SOAP processing into the substrate

Incorporating SOAP processing into the substrate eliminates the need to interact with the substrate via a proxy. This is a very useful feature which will eventually allow Web Services to interact directly with the substrate. To achieve this interaction with SOAP services we need to address two issues. First, pertains to the ability to use NaradaBrokering as a transport mechanism for SOAP messages. Second, to interact directly with Web Services the substrate should be able to function as a SOAP intermediary. This would allow messages to be redirected to the substrate for specialized processing.

1) A transport mechanism for SOAP messages

Here we use NaradaBrokering as a transport (depicted in Figure 11) for SOAP messages, which have traditionally been sent over HTTP. There are examples of different protocols that have been used. The Apache AXIS project for example has HTTP, SMTP/POP3, Java-RMI and JMS as registered transports. The substrate facilitates communications over a variety of transports, each with different properties. Depending on the SOAP message being issued (large attachments etc) appropriate lower-level transport would be used for routing the SOAP messages. The SOAP messaging would either be based on request/response semantics inherent in RPC-style service invocations or they could be based on asynchronous one-way messaging.
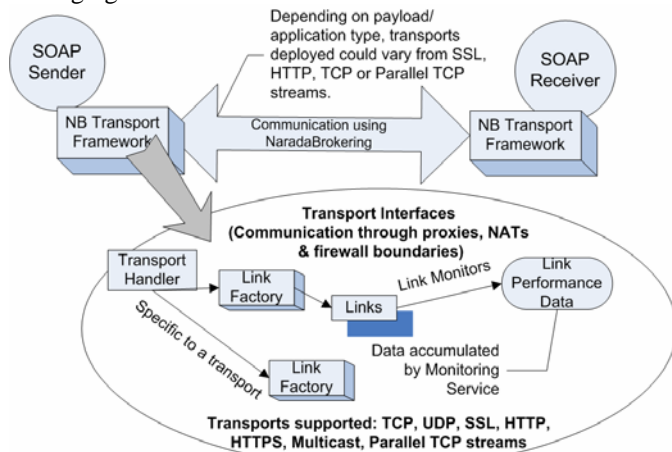

Figure 11: Incorporating support for SOAP in the transport layer

In the asynchronous style of messaging, a lookup service locates the appropriate providers that had previously registered with a naming service. This is the precursor to sending SOAP messages. A related issue is that of binding it to a protocol

stack so that clients can communicate with the service endpoint using the specified port and address.

Another scheme of interest is WSIF (Web Services Invocation Framework) [14] which considers WSDL to be the standardized representation of a Web Service. The fundamental tenet here is that while SOAP is great for interoperability between disparate systems, it is not necessarily the best approach if one is dealing with say a pure Java environment. The approach is meant to have developers deal with WSDL service descriptions instead of dealing directly with SOAP. There might be multiple implementations of a WSDL service description, WSIF provides an API so that a given client code can access any of these available bindings (whether it is SOAP or IIOP). Of course one needs to have a registered provider for any binding that one may choose to use (Java, EJB, SOAP etc). Microsoft's Indigo approach has been designed and implemented using the same principle. The need for dynamic interpretation of WSDL in WSIF is an issue that needs to be investigated further.

2) Interacting directly with Web Services

The approach that is outlined here (depicted in Figure 12) is intended to enable Web Services to interact directly with NaradaBrokering. Unlike the proxy based approach where the SOAP messages are not inspected, in this scheme the SOAP message is inspected, targeted to specific broker nodes within the substrate, and in some cases the substrate functioning as an intermediary can add/remove header elements in the SOAP message.

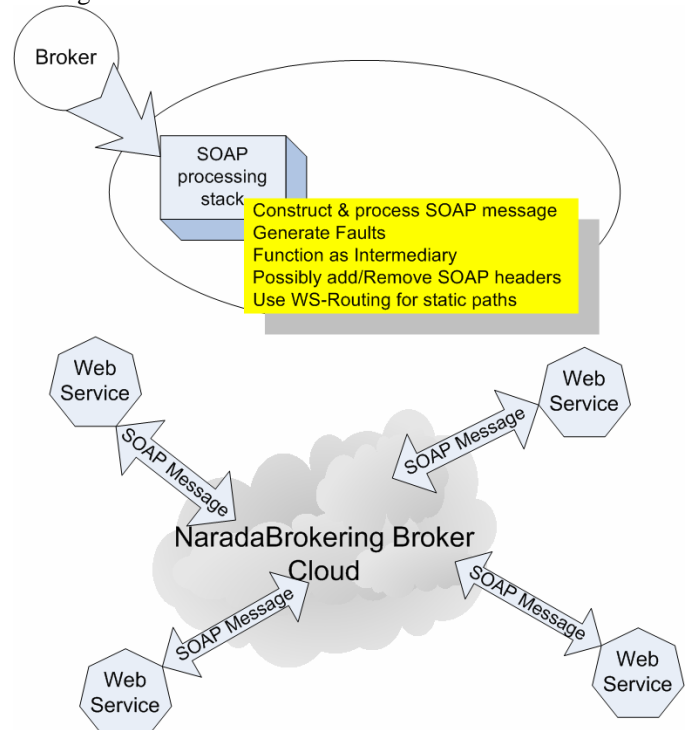Figure 12: Web Services interacting with the substrate

a) Incorporating the SOAP processing stack into individual brokers

To achieve this we first need to include the SOAP processing layer in individual broker nodes. To do this we would have an

interface which would allow us to plug in different SOAP implementations into the system. We are ultimately interested in the availability of, and the capability to process, SOAP messages within the substrate.

b)   Functioning as a SOAP intermediary

SOAP message headers are important since this is where information pertaining to functionality-specific elements is encoded. For example, this is the place where we will place all information pertaining to sequences, acknowledgements and retransmissions of a reliable messaging protocol. A SOAP message may pass through one or more intermediate systems prior to delivery at its ultimate destination (assuming that faults have not been issued en route). Such an intermediate system can examine these SOAP messages and initiate actions – issue faults, reroute to another node/final destination, and finally even update certain headers in the SOAP message.

To facilitate the use of the broker as an intermediary we use the **actor** attribute within the SOAP message's header element. The attribute identifies the system (substrate) that is intended to process the message (element in question). Once the SOAP message is received at the intermediary (a node within the substrate) as indicated in the **actor** attribute, the node is allowed to add additional headers some of which could be another **actor** attribute possibly re-routing processing to another special node and so forth. Such a scheme could be used to compress messages at a special node and archive it for subsequent retrieval or audit trails. The **actor** attribute basically allows us to control where the SOAP message needs to be sent for special processing.

The **mustUnderstand** attribute in the SOAP message is used to control the optional and mandatory elements within the SOAP message headers. This is also useful in the generation of faults. Depending on the header element targeted to the intermediary and the value of the **mustUnderstand** attribute; a substrate node may either generate a fault (**mustUnderstand** set to **1**) or ignore (**mustUnderstand** if it is not **1**) the targeted header that it does not understand. This attribute can be used to impose constraints on the processing of certain header elements. Finally, it must be noted that the substrate can interact with SOAP intermediaries that are not native NaradaBrokering services or for that matter even services that are directly hosted on the substrate.

c)   Use of WS-Routing to facilitate static paths

The one disadvantage of the **actor** attribute is that at a time it is possible to specify only one actor element since the order cannot be determined if multiple intermediaries are specified. This issue can be handled by using WS-Routing [15] which allows us to specify the route a SOAP message will take using the **via** directive. For reverse paths, (**rev**) the path is constructed as the event traverses through the forward path. This is sometimes used for request/response semantics such as Web Service invocations but does not seem necessary for most cases. WS-Routing allows us to do without specifying the reverse path.

3)   The role of WS-Addressing

WS-Addressing [16] is a way to abstract from the underlying transport infrastructures the messaging needs of an application. The substrate sitting at an organization's boundary could use WS-Addressing for making runtime decisions on where a message arriving from the outside world should be delivered within the infrastructure. Finally we could use WS-Addressing for endpoint references to deal with dynamic usage patterns involving WSDL Service descriptions. WS-Addressing is used in this scenario to facilitate the identification and specification of service instances for services that would generated dynamically besides enabling exchange of endpoint information.

C. Using Handlers

Special handlers will be associated with service endpoints so that the substrate can interact with these services. These handlers can then be grouped into a handler chain, and can be inserted in the processing path of either the service requestor, service implementation or both. Depending on the configuration of the handler, these handlers will process control messages initiated by the substrate such as heart-beats, latency measurements and utilization counters among others to facilitate efficient utilization of the resource. It must be noted that these messages need not propagate the entire handler chain. In the case of clients/services invoking other services the handler chain would add headers to enable easier processing within the substrate.

In the absence of the handler approach headers pertaining to the value added services would need to be processed within the application logic. This would entail serious rewrites to the application logic and in some cases the complexity of rewrites would ensure that the feature is rejected as a tradeoff. This would also imply that every application would need to be rewritten for every feature. Furthermore, the application would need to be rewritten, retested and redeployed every time there is a change to the value added service.

There are several advantages to using the handler approach. First, it facilitates incremental addition of value added functionality to an existing service. This functionality can easily be reused by other services, tested independently of the application logic. Second, it can be used to ensure that enhanced services continue to interoperate with existing services. Handlers within a handler chain can add, remove, process or ignore headers pertaining to incremental functionality. A handler can those choose not to generate faults for messages that arrive without functionality-specific information, thus enabling interaction with services that do not have the requisite handler to generate that information. The handler can in the same vein be used to impose constraints on services that communicate with it. For example in the face of a coordinate distributed denial-of-service attack a handler (or handlers) – which performs packet inspection, checks authentication information and checks from messages from malicious hosts – can be set up to impose new policy and constraints in an easy incremental fashion.
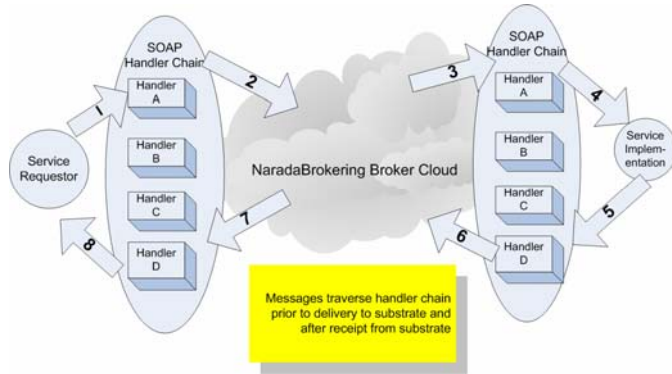
Figure 15: SOAP handler chains and the substrate

The scheme thus allows Web Services to interact directly with the substrate (depicted in Figure 15). Hosted services can utilize substrate properties such as resilience, load balancing etc through the incorporation of the appropriate handlers in the handler chain.   It should be noted that the service implementations will not change. We can write these handlers (a one time effort) for gSOAP, SOAP::Lite, AXIS and JAX-RPC. This would allow us to communicate with messages issued using gSOAP, SOAP::Lite and Axis SOAP.

V.   COMPLEMENTING SERVICE INTERACTIONS

In this section we identify areas such as discovery and load balancing where the substrate can provide additional functionality to the hosted services. These services are a precursor to a system where the substrate can compose services from discovered constituent services each of which would be chosen from a set comprising multiple instances. The substrate then needs to rely on its capabilities to ensure robust delivery to ensure that a task involving coordinated processing between multiple services (identified based on the task specification) is completed in the presence of failures. The substrate's capability to provide order also ensures that the invocations/interactions are consistent. Furthermore, the substrate can also be used to cache the results from prior interactions to optimize service utilizations.

A.   Discovering services using advertisements

SOAP handlers can automatically generate service advertisements on service startup. Web Services connect directly to NaradaBrokering and expose their capabilities through advertisements. The substrate supports a wide variety of querying schemes such as XPath queries, SQL queries (through JMS), Regular expressions and simple string matching. One or more of these will be used to support querying of services hosted on the substrate.

Entities in the system can advertise their services in an XML schema. These advertisements would be stored in the same way that the profiles are stored within the system. Events propagated by interested clients would essentially be either XPath or Regular expressions-like queries. These events would then be matched against the stored advertisements with the matching ones being routed back to the initiating client. The query events can specify the realms within which the

query's propagation might take place, thus allowing individual entities to control how localized their services can be.
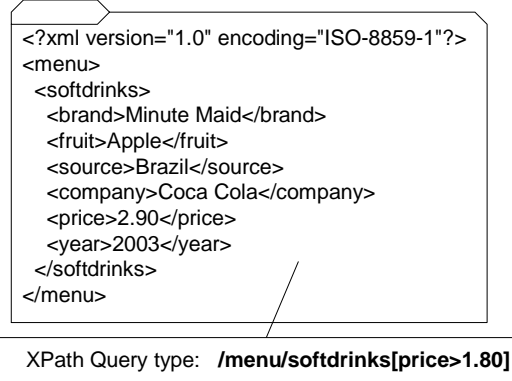

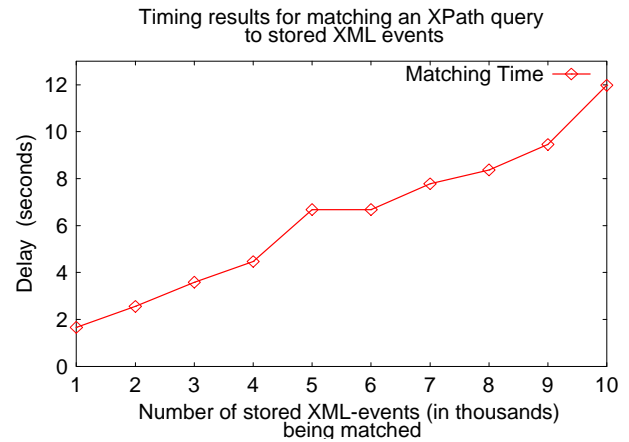Figure 13: The XML event and XPath query type

Figure 14: Matching an XPath query to stored XML events

Figure 14 depicts the matching times for a query against a set of stored XML events/advertisements. For matching XML advertisements, the performance would vary if it is constrained by the number of matched advertisements or stored XML events that need to be included in the query response. The stored XML events (this could of course be a WSDL service advertisement) and the issued XPath query are of the type depicted in Figure 13. For most discovery related operations, similar to those initiated in P2P systems, these numbers indicate adequate performance.

When we receive SOAP messages destined to services (either as requests or responses from other services), we use both the advertisements and the directives specified in the SOAP header elements to determine the route for these messages as well as any special processing (compression, archival, signing, credential delegation etc) that might be needed by these messages.

B.   Ability to load balance services

Load balancing algorithms operate on the ability to keep track of the number of active service instances as well as the load on these instances. In the substrate this is done by exchanging information with the instances at regular instances. In addition to this substrate also facilitates schemes where an instance may choose not to respond to the discovery request

initiated by an entity. This decision is of course predicated on the contents of the discovery request as well as the load at the resource in question. Finally, the substrate also uses round-trip delays from an entity to the replicated resources in question to compute the network distance separating the entity from the resource instance. The resource selection scheme in the substrate utilizes both the usage metric at a resource and the network distance to arrive at a decision regarding the resource to use.

To facilitate the same scheme for services, handlers would be registered with the handler chain associated with both the service requestors and the service implementations. These handlers would perform functions such as –-

- Constructing usage metric: This handler would construct usage metric at a node based on the number of requests processed, the rate of requests and the volume of information transferred. Additionally, this handler would also construct profiles regarding the underlying hardware hosting it – CPU performance, available memory and cache size.
- Creating round trip delay request and responses: This handler would facilitate the creation of round trip delay requests, responses and finally calculation of delay based on the response.
- Creating heart-beat monitors: This handler would send out heart beats at regular intervals so that the substrate can continue to track its availability.

It should be noted that services augmented with this functionality can still continue to interact with services and requestors that do not posses the requisite handlers. Under such circumstances both the service requestor and the service instance involved in the interactions will not have access to certain substrate capabilities. For example, even though a requestor does not have the appropriate handler the substrate will locate the right service; however this service instance might not be the one that it is closest to.

C. Robustness and disconnected operations

The substrate's robust delivery guarantee – delivery of events after prolonged/intentional disconnects can be used to augment interactions between service requestors and service implementations. Here requestors may initiate requests and disconnect, only to reconnect later on and retrieve responses. Similarly, in the event that the service implementation is offline (either intentional due to a scheduled maintenance or failure) upon recovery the substrate routes all pending requests to the service implementation.

As Web Services have become dominant in the Internet and Grid systems landscape, a need to ensure guaranteed delivery of interactions (encapsulated in messages) between services has become increasingly important. This highly important and complex area was previously being addressed in the Web Services community using homegrown proprietary application specific solutions. It should be noted that the terms guaranteed delivery and reliable delivery tend to be used interchangeably to signify the same concept. Reliable delivery of messages is now a key component of the Web Services roadmap, with two promising, and competing, specifications in this area viz. WS-Reliability [17] from OASIS and WS-ReliableMessaging [18] from IBM and Microsoft among others.

We have analyzed both these specifications. Our investigations were aimed at identifying the similarities and divergence in philosophies of these specifications. Table 1 summarizes some of our investigations. NaradaBrokering already has the essential functionality of these reliable messaging standards but needs to support the particular protocols. This support will be available in pure point-to-point mode (two endpoints linked together) or with the substrate managing interoperations with endpoints conforming to either standard. We are currently in the process of incorporating support for these standards. We believe it is quite possible that these specifications may continue to exist alongside each other. To account for such a scenario we also include a scheme for federating between these specifications. Such a scheme will allow service nodes to belong to either one of these competing specifications and still continue to interact reliably with each other.

Table 1: Comparing some of the features in WS-Reliability and WS-ReliableMessaging

	WS-Reliability	WS-RM
Related Specification	SOAP, WS-Security	WS-Policy, WS-Security, WS-Addressing
Unique Ids	URI based [RFC 2396], the syntax for the message-ID should be based on what is outlined in RFC2392.	URI based [RFC 2396]. No additional requirement. Messages within a sequence are identified based on message numbers.
Sequence numbering initialization	Starts at 0 for the first message in a group.	Starts at 1 for the first message in a group.
Sequence numbering rollover	Generate a new group identifier and begin new sequence only after receipt of last message in old sequence.	No new sequences can be generated. MessageRollover fault is issued.
Presence of numbering information and relation to delivery	REQUIRED only for guaranteed ordering.	Message number is REQUIRED for every message that needs to be delivered reliably.
Acknowledgements	Can be sent upon receipt of messages, as a callback or in response to a poll. Needed upon receipt of every message.	Acknowledgements can be based on a range of messages, and the timing for issuing this can be advertised in a policy. An endpoint may also choose to send acknowledgements at any time.
Requesting acknowledgements	The AckRequested element is REQUIRED in every message for which reliable delivery needs to be ensured.	AckRequested is used to request the receiving entity to acknowledge the message received. This is not REQUIRED.

Correlation associated with an Acknowledgement	The identifier associated with the message being acknowledged.	The identifier associated with the sequence of messages and the message number within that sequence.
Timestamps	Are expressed as UTC and conforms to a [XML Schema Part2: Data Types] `dateTime` element.	No explicit reference to UTC. Uses the dateTime format.
Retransmissions	Triggered after the receipt of a set of acknowledgements. In the event an acknowledgement is not received, the message is retransmitted until a specified number of resend attempts have been made.	Allows the specification of a RetransmissionInterval for a sequence (effects every message in the sequence). The interval can also be adjusted based on the exponential backoff algorithm.
Quality of Service	Is initiated by the sender.	WS-Policy assertions are used to meet delivery assurances.
Delivery sequences supported	Exactly once ordered delivery, reliable delivery. Order is always tied to guaranteed delivery and cannot be separately specified.	At most once, at least once and exactly once. Order is not necessarily tied to guaranteed delivery.
Security	Relies on WS-Security and assorted specifications	Relies on WS-Security and assorted specifications

VI. RELATED WORK

In this section we introduce related work in the area of distributed publish/subscribe and peer-to-peer systems. We compare these systems based on the type of interactions that they support and also on their schemes for robust delivery of events. Different systems address the problem of event delivery to relevant clients in different ways. In Elvin [19] network traffic reduction is accomplished through the use of quench expressions, which prevent clients from sending notifications for which there are no consumers. This, however, entails each producer to be aware of all the consumers and their subscriptions. In Sienna [20] optimization strategies include assembling patterns of notifications as close as possible to the publishers, while multicasting notifications as close as possible to the subscribers. In Gryphon [21] each broker maintains a list of all subscriptions within the system in a parallel search tree (PST). The PST is annotated with a trit vector encoding link routing information. These annotations are then used at matching time by a server to determine which of its neighbors should receive that event. Approaches for exploiting group based multicast for event delivery is discussed in Ref [22].

The Event Service [23] approach adopted by the OMG is one of establishing channels and subsequently registering suppliers and consumers to the event channels. The approach could entail clients (consumers) to be aware of a large number of event channels. The OMG Notification Service [24] addresses limitations pertaining to the lack of event filtering capability. However it attempts to preserve all the semantics specified in Event Service while allowing for interoperability between clients from the two services.

Unlike Elvin and the OMG Event Service, NaradaBrokering provides decoupled interactions between the interacting clients. Furthermore, the organization of subscriptions and calculation of destinations do not result in explosive search spaces. As opposed to the Gryphon approach where all nodes store the complete set of subscriptions at every broker node, in NaradaBrokering none of the nodes store all the subscriptions within the system. Also not every broker in NaradaBrokering is involved in the calculation of destinations. This greatly reduces the CPU cycles expended in NaradaBrokering for computing and routing interactions within the system.

The JXTA [25] (from juxtaposition) project at Sun Microsystems is a research effort to support large-scale P2P infrastructures. P2P interactions are propagated by a simple forwarding by peers and specialized routers known as rendezvous peers. These interactions are attenuated by having TTL (time-to-live) indicators. Pastry [26] from Microsoft incorporates a self-stabilizing infrastructure, which provides an efficient location and routing substrate for wide-area P2P applications. Each node in Pastry has a 128-bit ID and Pastry routes messages to nodes whose Node-Id is numerically closest to destination key contained in the message.

The JXTA approach results in flooding the peer network, with the range being controlled by the TTL indicators contained in the interactions. The NaradaBrokering scheme selectively deploys links for disseminating interactions. In Ref [27] we have demonstrated that we can route JXTA interactions more efficiently than the JXTA core itself.

Distributed Hash Tables (DHTs) have been quite popular in several P2P systems. Here each data object is associated with a key. A lookup service to locate this object returns the IP-address of the node hosting this object. Similar to a traditional hashtable data structure, other operations supported in the DHT include *put* and *get*. In DHT-based P2P overlay networks the nodes are organized based on the content that they possess. Here DHTs are used to locate, distribute, retrieve and manage data in these settings. This scheme provides bounded lookup times. However, most DHT-based P2P overlay networks do not facilitate keyword based searching, the lookups are instead based on identifiers computed by hashing functions such as SHA-1 and are derived from the content encapsulated within the communal resource.

FLAPPS [28, 29] provides a generalized infrastructure for peer network design. Here peers are organized into a peer network comprised of overlapping peer groups with transit peers efficient routing requests. It should be noted that though FLAPPS is not DHT based it can indeed be used to support DHT-style systems, Resilient Overlay Networks (RON) style networks and traditional peer networks. Unlike DHTs, in FLAPPS, key identifiers need not be know prior to data location. FLAPPS-based services define hierarchical, flexible

and decomposable namespaces. Here, service representation of names can vary from delimited text strings to an XML parse tree. Furthermore, this namespace also supports prefix names while locating specific resources.

A more recent effort to assuage the keyword problems in DHT can be found in [30, 31].Unlike DHT systems with consistent hashing schemes data elements in Squid attempt to preserve locality and use the dimension-reducing mapping called Space Filling Curves (SFC). By preserving locality in the DHT index space Squid supports complex queries using partial keywords, wildcards and ranges.

DACE [32] introduces a failure model, for the strongly decoupled nature of pub/sub systems. This model tolerates crash failures and partitioning, while not relying on consistent views being shared by the members. DACE achieves its goal through a self-stabilizing exchange of views through the Topic Membership protocol. This however may prove to be very expensive if the number and rate at which the members change their membership is high. The Gryphon [33] system uses knowledge and curiosity streams to determine gaps in intended delivery sequences. This scheme requires persistent storage at every publishing site and meets the delivery guarantees as long as the intended recipient stays connected in the presence of intermediate broker and link failures. It is not clear how this scheme will perform when most entities within the system are both publisher and subscribers, thus entailing stable storage at every node in the broker network. Furthermore it is conceivable that the entity itself may fail, the approach does not clearly outline how it handles these cases. Systems such as Sienna and Elvin focus on efficiently disseminating events, and do not sufficiently address the reliable delivery problem.

The Fault Tolerant CORBA (FT-CORBA) [34] specification from the OMG defines interfaces, policies and services that increase reliability and dependability in CORBA applications. The fault tolerance scheme used in FT-CORBA is based on entity redundancy [35], specifically the replication of CORBA objects. Approaches such as Eternal [36] and Aqua [37], provide fault tolerance by modifying the ORB. OS level interceptions of have also been used to tolerate faults in applications.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we outlined our scheme to exploit WS-Addressing and the SOAP processing stack to build ways of interfacing the NaradaBrokering substrate with Web services. The advantages to this approach include the fact that it would entail no changes to the service implementations themselves. Services automatically inherit functionalities and capabilities provided by the substrate. It should be noted that such substrate services or other stand-alone services can continue to interoperate with each other.

REFERENCES

[1] The NaradaBrokering Project at the Community Grids Lab: http://www.naradabrokering.org

[2] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.

[3] Geoffrey Fox and Shrideep Pallickara. An Event Service to Support Grid Computational Environments Journal of Concurrency and Computation: Practice & Experience. Special Issue on Grid Computing Environments. Volume 14(13-15) pp 1097-1129.

[4] Shrideep Pallickara and Geoffrey Fox. On the Matching Of Events in Distributed Brokering Systems. Proceedings of IEEE ITCC Conference on Information Technology. April 2004. Volume II pp 68-76.

[5] Shrideep Pallickara and Geoffrey Fox. A Scheme for Reliable Delivery of Events in Distributed Middleware Systems. Proceedings of the IEEE International Conference on Autonomic Computing. pp 328-329. 2004.

[6] G. Fox, S. Lim, S. Pallickara and M. Pierce. Message-Based Cellular Peer-to-Peer Grids: Foundations for Secure Federation and Autonomic Services. (To appear) Journal of Future Generation Computer Systems.

[7] Shrideep Pallickara et al. A Security Framework for Distributed Brokering Systems. Available from http://www.naradabrokering.org

[8] Shrideep Pallickara, Geoffrey Fox, John Yin, Gurhan Gunduz, Hongbin Liu, Ahmet Uyar, Mustafa Varank. A Transport Framework for Distributed Brokering Systems. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. (PDPTA'03).

[9] M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[10] Web Services Description Language (WSDL) 1.1 http://www.w3.org/TR/wsdl

[11] The Open Grid Services Infrastructure (OGSI). http://www.gridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf

[12] The Web Services Resource Framework (WSRF) http://www.globus.org/wsrf/

[13] The Web Services Notification (WS-Notification) http://www-106.ibm.com/developerworks/library/ specification/ws-notification/

[14] Web Services Invocation Framework (WSIF) http://ws.apache.org/wsif/

[15] Web Services Routing Protocol (WS-Routing) http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp

[16] Web Services Addressing (WSAddressing) ftp://www6.software.ibm.com/software/developer/library/ wsadd200403.pdf

[17] Web Services Reliable Messaging TC WS-Reliability. http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf

[18] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf

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

[20] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pages 219–227, Portland OR, USA, 2000.

[21] G. Banavar et al. 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.

[22] Lukasz Opyrchal et. al. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. Middleware 2000: 185-207

[23] The Object Management Group (OMG). OMG's CORBA Event Service. Available from http://www.omg.org/

[24] The Object Management Group (OMG). OMG's CORBA Notification Service. Available from http://www.omg.org/

[25] Sun Microsystems. The JXTA Project and Peer-to-Peer Technology http://www.jxta.org

[26] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. Proceedings of Middleware 2001.

[27] Geoffrey Fox, Shrideep Pallickara and Xi Rao. A Scaleable Event Infrastructure for Peer to Peer Grids. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington. November 2002.

[28] B. Scott Michel, Peter L. Reiher: Peer-through-Peer Communication for Information Logistics. GI Jahrestagung (1) 2001: 248-256

[29] B. Michel and P. Reiher. Peer-to-Peer Internetworking. In OPENSIG, September 2001.

[30] C. Schmidt and M. Parashar. Enabling Flexible Queries with Guarantees in P2P Systems, IEEE Network Computing, Special Issue on Information Dissemination on the Web, IEEE Computer Society Press, Vol. 8, No. 3, pp. 19- 26, May/June 2004.

[31] C. Schmidt and M.Parashar. A Peer-to-Peer Approach to Web Service Discovery,", World Wide Web, Internet and Web Information Systems, Kluwer Academic Publishers, Vol. 7, No. 2, pp. 211-229, June 2004.

[32] R. Boichat, P. Th. Eugster, R. Guerraoui, and J. Sventek. Effective Multicastprogramming in Large Scale Distributed Systems. Concurrency: Practice and Experience, 2000.

[33] S. Bhola, R. Strom, S. Bagchi, Y. Zhao, J. Auerbach: Exactly-once Delivery in a Content-based Publish-Subscribe System. DSN 2002: 7-16

[34] Object Management Group, Fault Tolerant CORBA Specification. OMG Document orbos/99-12-08 edition, 99.

[35] Object Management Group, Fault Tolerant CORBA Using Entity Redundancy RFP. OMG Document orbos/98-04-01.

[36] P. Narasimhan, et al. Using Interceptors to Enhance CORBA. IEEE Computer 32(7): 62-68 (1999)

[37] Michel Cukier et al. AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. Symposium on Reliable Distributed Systems 1998: 245-253.

Geoffrey Fox received his PhD (Theoretical Physics) from Cambridge University in 1967. He is a Professor in the Department of Computer Science, School of Informatics and Physics at Indiana University. He is the Director of the Community Grids Lab at Indiana University. He was also the director of the Northeast Parallel Architectures Center at Syracuse University from 1990-2000.

Shrideep Pallickara received his Masters and PhD (Computer Engineering) from Syracuse University in 1998 and 2001 respectively. He is currently a Post Doctoral Researcher at the Community Grids Lab at Indiana University.