Towards Flexible Messaging for SOAP Based Services

Geoffrey Fox[1], Shrideep Pallickara[1] and Savas Parastatidis[2]
Community Grids Lab, Indiana University[1]
School of Computing Science, University of Newcastle[2]
gcf@indiana.edu, spallick@indiana.edu and Savas.Parastatidis@newcastle.ac.uk

1. 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 the system 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 protocols 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 build two distinct ways of interfacing NaradaBrokering with Web services. The first involves using a NaradaBrokering-proxy that acts as an interface between services and the NaradaBrokering messaging substrate. 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 WS approach such as IIOP or native Java) and services based on Web Services.

The second approach provides 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 Java (Apache Axis and Sun's JAX-RPC), C++ (gSOAP) and Perl (Soap::Lite).

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 6 we investigate how the scheme outlined here can be deployed to augment Grid applications. In section 7 we provide an overview of related work in the area of distributed publish/subscribe and peer-to-peer systems. Finally, in section 8 we outline our conclusions and future work.

2. NaradaBrokering substrate

NaradaBrokering [1-9] 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 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.

2.1. 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.

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, which eliminates continuous echoing. The broker network maps (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.

2.2. Services within NaradaBrokering

In NaradaBrokering entities can also specify constraints on the 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.

2.2.1 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.

2.2.1.1 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.

The experimental setups are depicted Figure 1. 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 connectionless one such as UDP. The publishing/subscribing entities (hosted on the same machine to account for clock synchronizations and drifts), brokers and RDS are all hosted on separate machines (1GHz, 256MB RAM) with each process 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 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.

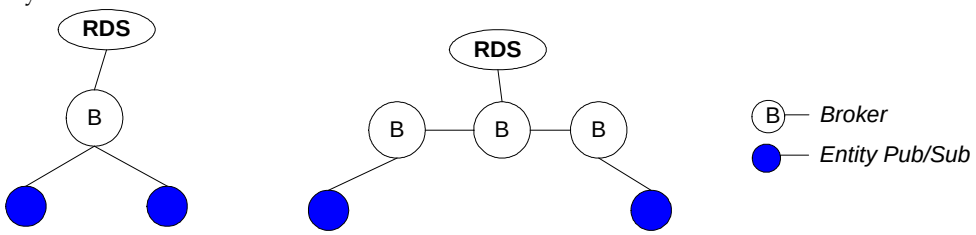

Figure 1: Experimental Setups

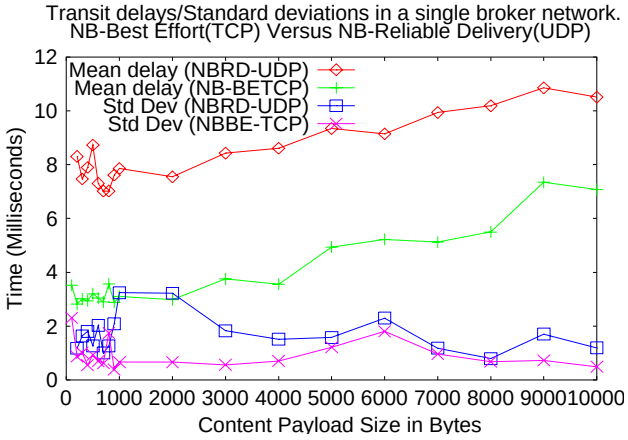

Figure 2: Results from the single broker setting

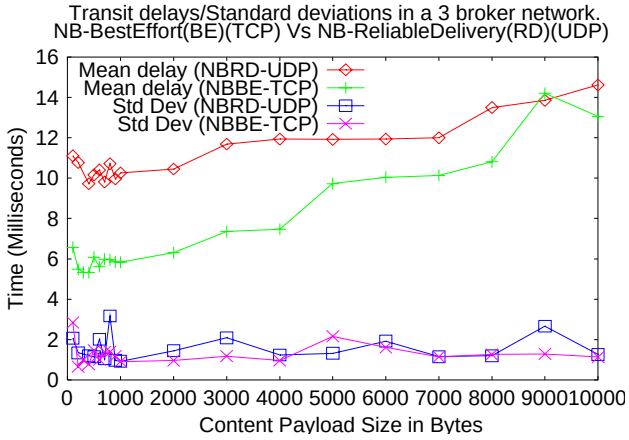

Figure 3: Results from the 3 broker setting

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 2 depicts the transit delay and standard deviation associated with a single broker network, while Figure 3 depicts the same for the 3 broker network.

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.

2.2.2 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. Additional details can be found in Ref [6]. 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.

2.2.3 Time and Buffering Services

The substrate also includes an implementation of the Network Time Protocol (NTP). The NaradaBrokering TimeService [7] 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.

2.2.4 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.

2.3. 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), UDP, SSL, HTTP and RTP. The typical delays involved with NaradaBrokering's transport framework in LAN settings is around 1 millisecond. Additional information regarding measurements within the transport framework can be found in Ref [8].

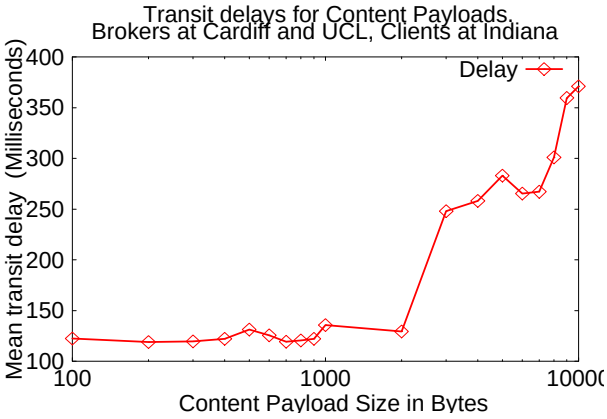

Figure 4: Transit delay for message samples (UCL, Cardiff and IU)

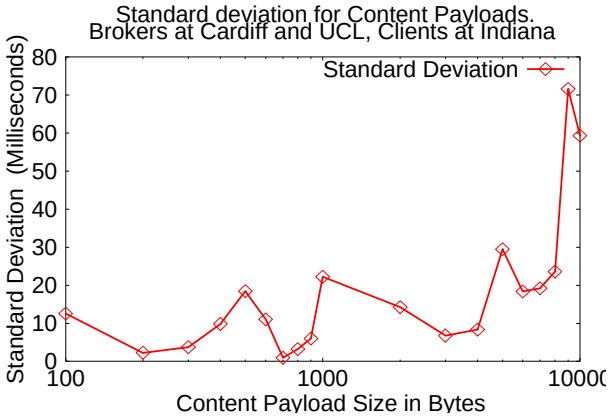

Figure 5: Standard deviation for message samples (UCL, Cardiff and IU)

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 4 and Figure 5 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.

3. 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. WSDL describes message formats and message exchange patterns for services using XML. Interactions are facilitated through the exchange of SOAP messages. The use of XML throughout the Web Services stack of specifications allow interactions between services running on different platforms, containers, implemented in different languages, and over multiple transports.

Existing or emerging Web Services specifications deal with the issues of service discovery, all aspects of message-based security, transactions, notification, reliability, etc. These specifications can be composed together to facilitate the incremental addition of features and capabilities when building distributed applications. 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 (i.e., SOAP and WSDL). Forcing the user to choose one specification over another can be quite complicated and error-prone. The overarching goal is the automation of the negotiation of specifications used for interactions. Automated negotiations eliminate human intervention and concomitant errors resulting from this. Furthermore, these automated negotiations would be adaptive, where the negotiations could interoperate between (or deploy the best available specification from a set of) possibly competing specifications. These automated negotiations thus make these interactions easier, simpler and more reliable.

These negotiations are facilitated by the substrate, which permeates the hosted services. By hosted services, we mean those services, which have been registered with the substrate. The substrate will enable services to interact, discover, compose and utilize other hosted services. 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 scheme that is proposed 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 and it uses this information to locate services. Since the clients/services interact directly with the substrate they have access to all the services provided by it. We 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 online.
b) The substrate can cache the results of an invocation and retrieve them in case the same invocation is made.
c) 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.
d) End-to-end secure interactions. The approach within the substrate is consistent with that deployed in WS-Security which relies on message-level security.
e) Inevitably the realms across which entities communicate involve firewalls, proxies and NAT boundaries. The substrate facilitates communications over such boundaries.

f) In the event of large payloads/attachments the substrate provides services such as compression, fragmentation of payload into smaller ones and NaradaBrokering-enhanced GridFTP.

3.1. Grid Services and Web Services

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 specifications. 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] – can co-exist with other specifications in the Web Services area such as authentication, transactions, reliable messaging and addressing. The WSRF specification also includes WS-Notification [13] which models notifications using a topic based publish/subscribe mechanism. Work is presently underway to provide support for WS-Notification within NaradaBrokering and a prototype version will be part of a release scheduled for May 2004.

Similarly, the WS-GAF [14] effort in the United Kingdom provides a framework for building Grid applications using existing Web Services specifications while adhering to the principles of service-oriented architectures. The proposed solution demonstrates how issues like stateful interactions, logical resource naming, metadata, and lifetime management can be easily addressed using existing Web Services technologies.

Throughout our discussions when we refer to services as Web Services the term refers to services that are parts of the Service Oriented Architecture. 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 Framework specification.

4. Approaches to interacting with Web Services

In this section we describe approaches to incorporating support for Web Services within the NaradaBrokering substrate. The first involves using a NaradaBrokering-proxy that acts as an interface between services and the messaging substrate. The second approach provides an endpoint 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 such as Apache Axis (such a handler can be used with Sun's JAX-RPC with no changes), gSOAP, Soap::Lite, and ASP.NET. 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.

4.1. Proxy Approach

This approach is based on SOAP and involves using the proxy architecture to deploy Web Services within the system. Here a service invocation using 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 WS approach such as IIOP or native Java) and services based on Web Service standards.

This solution however was application dependent and the proxy had to be rewritten or tweaked for different applications. Also, since only the proxies were interacting with the substrate all the guarantees/services provided by the substrate were accessible only to these proxies and not to the web service client/service.

4.2. 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 into the substrate for specialized processing.

4.2.1 A transport mechanism for SOAP messages

Here we use NaradaBrokering as a transport (depicted in Figure 6) 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.

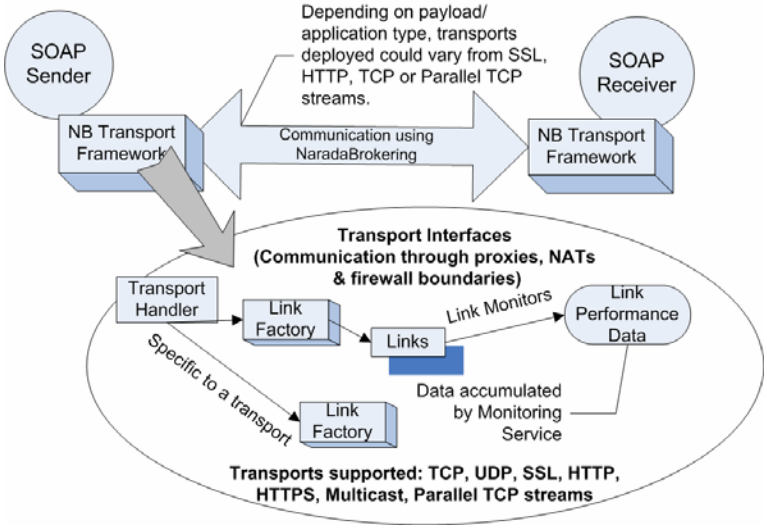

Figure 6: Incorporating support for SOAP in the transport layer

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.

4.2.1.1 Using WSIF

The Web Services Invocation Framework (WSIF) [15] 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 you are 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 bindings 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 you need to have a registered provider for any binding that you may choose to use (Java, EJB, SOAP etc). Microsoft's Indigo approach has been designed and implemented using the same principle.

4.2.2 Interacting directly with Web Services

The approach that is outlined here (depicted in Figure 7) is intended to enable Web Services to interact directly with NaradaBrokering. Unlike the proxy based approach where the SOAP messages were 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.

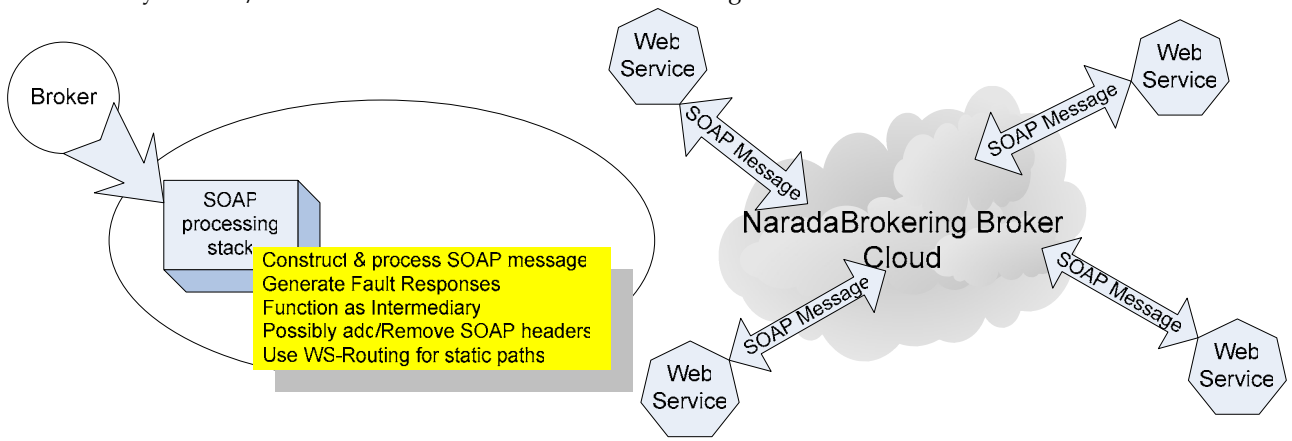

Figure 7: Web Services interacting with the substrate

4.2.2.1 Incorporating 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.

4.2.2.2 Functioning as a SOAP intermediary

SOAP 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 SOAP header (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 include 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 defines the **actor** responsible for processing the message in the chain of SOAP actors. These **actor**s can in turn be configured for special processing on received messages.

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 (**messageUnderstand** set to **1**) or ignore (**messageUnderstand** not set to **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.

4.2.2.3 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.

5. 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.

5.1. 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.

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.

5.2. 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.

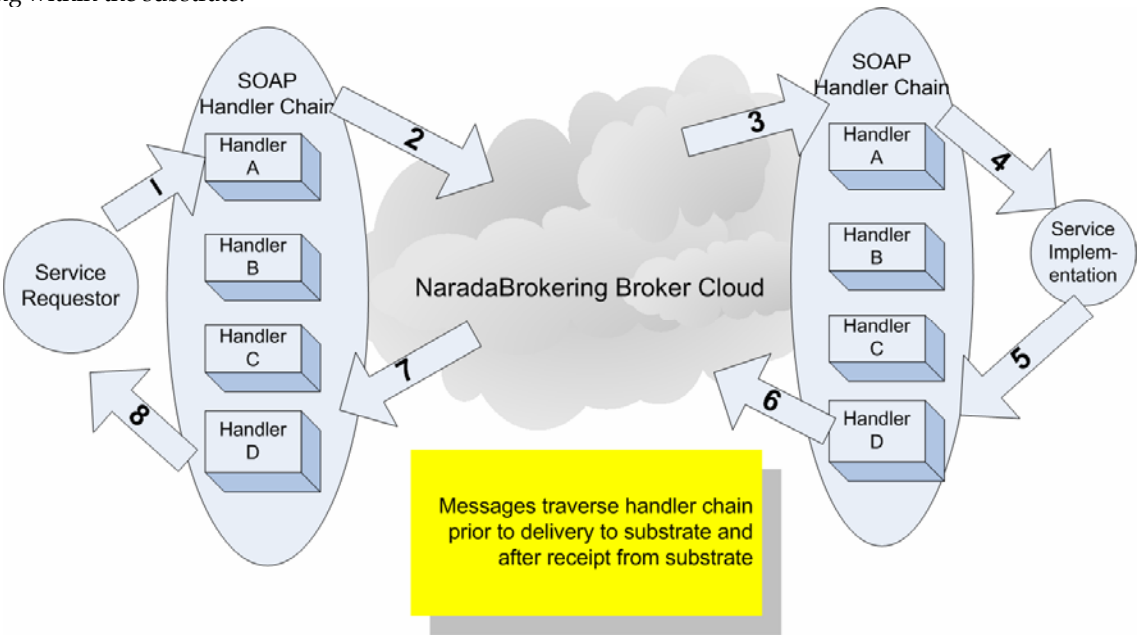

Figure 8: SOAP handler chains and the substrate

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 then 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.

The scheme thus allows Web Services to interact directly with the substrate (depicted in **Error! Reference source not found.**). 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.

5.3. 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 the 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.

5.4. The role of WS-Addressing

WS-Addressing [17] is a way to abstract from the underlying transport infrastructures the addressing needs of an application. WS-Addressing incorporates support for end point references (EPRs) and message information (MI) headers. EPRs standardize the format for referencing (and passing around references to) both a Web service and Web service instances as well. The MI headers standardize information pertaining to message processing related to replies, faults, actions and the relationship to prior messages. This is especially useful in cases where there would be multiple dedicated entities dealing with these different cases. In fact it has been argued [18] that WS-Addressing facilitates implicit asynchronous communications through the correct use of these MI headers irrespective of whether this service was defined asynchronously or not.

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 be generated dynamically.

5.5. Enhanced messaging features and fault tolerance

Depending on the messaging needs of the application the substrate can deploy appropriate protocols for communications with flexible security and fault tolerance. The transport protocols that can be deployed include TCP, Parallel TCP streams for large payloads, UDP, Multicast and HTTP. In case the communications need to be secured at the transport layer, the substrate can deploy either SSL or HTTPS to facilitate this. Furthermore, the choice of transport to deploy can be computed using the metrics reported by the performance monitoring service.

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. The substrate also provides support for enhanced GridFTP which can be used for robust transfer of large files (> 1 GB).

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 [19] from OASIS and WS-ReliableMessaging [20] 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. 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.

The substrate provides support for P2P style communications such as JXTA and GridTorrent (a derivate of the BitTorrent scheme) and can operate with only the endpoints. The substrate provides support for secure delivery of packets over unreliable links. The scheme deployed within the substrate is one where packets are secured using previously

exchanged secret keys. Individual entities encrypt and sign individual messages. It should be noted that one or more of the transport, fault-tolerant and security features can be incrementally added by the application logic to facilitate custom communication protocols.

5.6. Caching of invocation results and Composition of Services

The substrate can be used as a store to cache results from prior invocations. In some cases, depending on the service in questions, results from the cache will not only be accurate but will eliminate overheads pertaining to processing of the request. The knowledge of hosted services and load balancing features inherited from the substrate can be used to enable efficient service compositions. Specifications like WS-CoordinationFramework, WS-Coordination [21] can be used to ensure the completion of the composed activity. As mentioned in the earlier section, the completion of the activity can be ensured even in the presence of failures.

6. Grid Computing and using the scheme with frameworks

The proposed approach to introducing part of the NaradaBrokering functionality into the SOAP message processing chain makes it possible to leverage the NaradaBrokering features when building Grid applications using different Web Services-based infrastructures, like WS-RF and WS-GAF.

6.1. WS-RF

The Web Services Resources Framework suite of specifications represent a rendering of the OGSI conceptual model for building Grid applications that is based on existing Web Services specifications like WSDL and WS-Addressing. The framework introduces the concept of stateful resource on which Web Services may operate. WS-RF concentrates on modeling the interactions with such stateful resources, defines patterns for managing the lifetime of their states, enables subscriptions to their state, etc.

WS-RF can be combined with other Web Services specification when additional functionality, like message-level security, transactions, coordination, etc., is required. Hence, it is also possible to use the NaradaBrokering SOAP handlers with WS-RF-enabled services allowing the use of features like load-balancing of interactions with particular stateful resources, monitoring of the activity on resources, enabling bridging between different implementations of reliability and notification specifications, etc.

6.2. WS-GAF

The Web Services Grid Application Framework is a proposal on how Grid applications could be built using only existing Web Services specifications and without the need to explicitly model resources. It addresses the same requirements as OGSI but in a way that is consistent with Web Services specifications. Unlike WS-RF which is concerned with the modeling of resources as the building blocks of distributed applications, WS-GAF focuses only on the use of services and messages and encourages the use of stable and widely accepted specifications. Design patterns on how issues with resource identity, stateful/contextualized interactions, metadata, and lifetime management are presented in [14]. WS-GAF is not an infrastructure per se but, rather, a collection of design patterns that software architects could employ when designing Web Services-based distributed applications.

Since WS-GAF does not introduce a new infrastructure, NaradaBrokering SOAP handlers could be used to provide additional QoS features and/or bridges between semantically similar specifications (e.g., in the areas of notification, coordination, etc.) in applications. There is nothing different in the use of the NaradaBrokering SOAP handlers with WS-GAF from what was described in the previous sections.

6.3. VPN Grid and Inter-Enterprise NaradaBrokering communications

It is inevitable that the realms across which we try to communicate would be protected by firewalls, DHCP and NAT boundaries that can stop our elegant application channels dead in their tracks. The substrate facilitates communications across firewall (HTTP tunneling is used if only HTTP traffic is allowed), DHCP and NAT boundaries. Sometimes communications would also be through authenticating proxies. The various authentication schemes currently supported include Basic, Digest and NTLM (a proprietary scheme from Microsoft). The firewalls/packet-inspecting authentication proxies over which we have conducted testing include Microsoft's ISA, Checkpoint firewall software and Apache proxies. The administrative module cycles through a set of protocols before it determines which protocol/authentication-challenge needs to be deployed to facilitate communications.

As VPN technology has matured, it has been increasingly deployed within a wide range of organizations. We are currently planning to incorporate well known VPN protocols such as Point-to-Point Tunneling Protocol (PPTP) from Microsoft and the more recent Layer Two(2) Tunneling Protocol (L2TP) from Cisco/Microsoft. Like PPTP, L2TP requires that the ISP's routers support the protocol. IPSec can be used within both these schemes. Since VPN support is currently available in most organizations, support for these protocols within the substrate will allow us to leverage capabilities in this technology. This will lead to a Virtual Private Grid explored further in Ref [6].

7. 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 [22] 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 [23] 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 [24] 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 [25]. The Event Service [26] approach adopted by the OMG is one of establishing channels and subsequently registering suppliers and consumers to the event channels. The approach could entail consumers to be aware of a large number of event channels.

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 Network Weather System (NWS) [27] collects end-to-end throughput and latency information and uses that information to forecast future performance. Metrics are collected by sensors, which are organized as a hierarchy of sensor sets called cliques in order to prevent contention and also to provide scalability. The measurement intervals can be adjusted so that intrusiveness is limited while ensuring scalability. The sensor interface in NWS is designed such that it can easily incorporate data from other network performance tools. In addition to network metrics, collected over the TCP/IP transport protocol, NWS also accumulates CPU and available non-paged memory information from various nodes. The NaradaBrokering Monitoring service is designed such that it can incorporate results from services such as NWS to facilitate performance based routing (PBR).

The JXTA [28] (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. 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 [29] 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. Similar to a traditional hashtable data structure, other operations supported in the DHT include *put* and *get*. In 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, P2P overlay networks such as Pastry [30] from Microsoft 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.

DACE [31] 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 [32] 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) [33] 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 [34], specifically the replication of CORBA objects. Approaches such as Eternal [35] and Aqua [36], provide fault tolerance by modifying the ORB. OS level interceptions of have also been used to tolerate faults in applications.

The WS-Notification [37] specification from IBM and Globus is a high level messaging specification which aims to abstract notification mechanisms. This specification is still evolving. We are currently incorporating support for WS-Notification within the substrate and will release a prototype of this in May 2004.

8. Conclusions & Future Work

In this paper we outlined our scheme to exploit 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. In the proxy based scheme there would be no changes in the processing stack either. In the plug-in mode services automatically inherit functionalities and capabilities provided by the substrate. Services in either scheme and other stand-alone services can continue to interoperate with each other.

9. 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. pp 68-76 Volume II.

[5] Shrideep Pallickara and Geoffrey Fox. A Scheme for Reliable Delivery of Events in Distributed Middleware Systems. (To appear) Proceedings of the IEEE International Conference on Autonomic Computing. 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] Hasan Bulut, Shrideep Pallickara and Geoffrey Fox. Implementing a NTP-Based Time Service within a Distributed Brokering System. (To appear) ACM International Conference on the Principles and Practice of Programming in Java.

[8] Shrideep Pallickara et al. 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] Web Services Notification http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[14] The Web Services Grid Application Framework (WS-GAF) http://www.neresc.ac.uk/ws-gaf

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

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

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

[18] Impact of WS-Addressing on SOAP. ftp://www6.software.ibm.com/software/developer/library/ws-address.pdf

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

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

[21] Web Services Coordination (WS-Coordination) ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf

[22] 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.

[23] A Carzaniga, DS. Rosenblum, and A. 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, USA, 2000.

[24] 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.

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

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

[27] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. Proceedings of the 6th IEEE Symp. On High Performance Distributed Computing, August 1997.

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

[29] 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.

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

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

[32] S. Bhola, et al. Exactly-once Delivery in a Content-based Publish-Subscribe System. DSN 2002: 7-16

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

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

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

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

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