

# Messaging and the Web Service Distributed Operating Environment

Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil  
(gcf@, spallick@, marpiere@, hgadgil@cs) indiana.edu  
Community Grids Laboratory  
Indiana University

## Abstract

Grid application frameworks have increasingly aligned themselves with the developments in Web services. Web services are currently the most popular infrastructure based on Service Oriented Architecture (SOA) paradigm. There are three core areas within the SOA framework: a set of capabilities that are remotely accessible, communications using messages, and metadata pertaining to the aforementioned capabilities. In this paper, we focus on issues related to the messaging substrate hosting these services; we base these discussions on the NaradaBrokering system. We outline strategies to leverage capabilities available within the substrate without the need to make any changes to the service implementations themselves. We also identify the set of services needed to build Grids of Grids. Finally, we discuss another technology, HPSearch, which facilitates the administration of the substrate and the deployment of applications via a scripting interface.

## 1. Introduction

With the advent of the Open Grid Computing Architecture (OGSA) [1] and the UK e-Science program, Grid computing has aligned itself with Web Service standards activities: Grid infrastructure will be Web Service infrastructure, although the aggressiveness in developing and adopting extensions is a matter of debate. The current general consensus is that Web and Grid Services should follow Service Oriented Architecture (SOA) principles, such as discussed by the World Wide Web Consortium's Web Service Architecture working group. We summarize key SOA features as follows, following Ref. [2]:

1. SOAs are composed of services that present programmatic access to resources to remote client applications (or "agents"). Typical basic (atomic) services include data access (logically wrapping storage technologies such as databases and file systems), the ability to run and manage remote applications. More complicated services may be composed of these basic services using such expression languages as coupled with workflow engines.
2. Services communicate using messages. Messages are usually encoded using SOAP. The asynchronous nature of messaging is one of the keys to Grid and Web service scalability beyond the intranets.
3. SOAs are metadata rich. We must describe service interfaces, provide descriptions of services so that we know how to use them, provide look-up registries to find service URLs, and so forth.

Much debate has gone into refining concepts such as stateful conversations between agents and stateful resources accessed through services [3]. However, we believe that the other two characteristics, messaging and metadata, have been somewhat overlooked. In this paper, we are particularly interested in the messaging infrastructure needed to realize such things as the SOAP message processing model (particularly in SOAP 1.2), which allows for multiple intermediaries that will need to process header information required by WS-Addressing and WS-Security. We extend this to argue that SOAP headers combined with messaging middleware enable a revision of the standard Internet protocol stack.

These issues have directly relevance to scientific Grid applications, which need to go beyond remote procedure calls in client-server interactions to support integrated distributed applications that couple databases, high performance computing codes, and visualization codes [4, 5]. These coordinated, composite applications are asynchronous by their nature: applications may take hours or days to complete. Message-based Grids, events, and service coordination are not just abstract Grid research issues: they are needed to meet the requirements of real science application grids. We note further that service-based scientific grids imply a higher level of encapsulation: one can think of this as a "grid shell" for programming different services. The processing engines for such shells are essentially workflow engines. These higher level languages are also suitable for putting grid application development directly in the hands of science application developers.

In this paper we investigate issues pertinent to the intersection of most popular incarnation of SOA, Web services, and a distributed messaging infrastructure, NaradaBrokering (NB). We first begin by providing an overview of the NB infrastructure along with a discussion of its current capabilities in section 2. In section 3, we discuss how NB can be deployed to augment Web service interactions and capabilities. Section 4 outlines our work on building a

Grid of Grids, this section also includes a discussion of the infrastructural issues related to deploying such systems. Section 5 includes a discussion of the HPSearch technology that provides scripting support to a variety of operations. Finally in [section 6](#) we outline future work.

## **2. Web Service Messaging Infrastructure: Internet-on-Internet**

The SOAP processing model supports a general purpose messaging strategy of multiple, distributed SOAP processing nodes that can act as intermediaries, routing nodes, and final destinations. This model goes well beyond the standard client-server, remote procedure call methodology that many current Web Service implementations use. In this section, we review the general requirements for building a message oriented middleware (MoM) that will realize the SOAP processing model as well as several Web Service extensions (such as security, addressing, reliability, and discovery). A more detailed discussion of these topics is given in [\[6\]](#). Such middleware messaging substrates may, in addition, provide additional levels of support that are logically separate from services and messages, such as performance, fault tolerance, and reliability. Here we describe our messaging substrate NaradaBrokering [\[7,8\]](#). Communication within NaradaBrokering asynchronous, thus facilitating the design of loosely coupled systems. As we have shown [\[9\]](#), messaging systems may achieve communication speeds with millisecond latencies. Messaging systems typically are used to communicate specialized messages called *events* to all parts of a system. 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.

### **2.1. Routing 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 has been added.

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 traverse through the broker network are updated to snapshot its dissemination within the broker network 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 [\[10\]](#) 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. 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.

### **2.2. Services within Messaging Infrastructures**

In messaging systems, entities should be able to specify constraints on the QoS related to the delivery of messages. The QoS pertain to the reliable delivery, order, duplicate elimination, security and size of the published events and their encapsulated payloads. We have researched these issues for NaradaBrokering [\[11\]](#) 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 Reliable Delivery Service and Replay of events**

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.

The scheme also facilitates ordered and exactly once delivery of events. More recently the reliable delivery framework has been extended to incorporate support for multiple replications. Any of these replicas could be used for recovery from failures or to ensure reliable delivery. The replicas themselves may fail and a recovering replica arrives at a consistent after exchanging a series of control messages with the other replicas.

The NB reliable delivery scheme has been extended to provide support replays of events. A variety of replay requests formats are supported. Furthermore, a time differential service which preserves the time-spacing between successive events in the replay is also available.

### **2.2.2 Dealing with large payload sizes: Compression/Fragmentation**

Web Service messaging systems that support science grids should provide a means for managing very large data transmissions. Compression and decompression are obviously desirable capabilities. Additionally, message fragmentation/coalescence can be used to verify completed and uncorrupted large transmissions, and also support partial re-transmissions in the case of failures. The latter efficiently eliminates the need to re-transmit the entire message in the case of a few incorrectly delivered fragments. Fragmentation also allows for parallel transmission within the MoM.

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 [12]. 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**

Proper time sequence ordering of messages and events is of utmost importance in many applications, such as audio/videocollaboration systems. The NB system provides this capability through an implementation of the Network Time Protocol (NTP). The NaradaBrokering TimeService [13] 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 Security Services**

Messaging systems possess many interesting requirements not present in client-server systems. The latter may be suitably handled by transport level security, but in MoMs the messages may pass through many intermediaries and may be destined for multiple recipients. The NaradaBrokering security framework [14] 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. Entities can verify the integrity and source of these events, before proceeding to process the encrypted payload.

## **3. Incorporating Support for Web Services within the Messaging Substrate**

SOAP has emerged as the de facto standard for encapsulating and transporting various Web services interactions. SOAP [15], along with WSDL [16] and UDDI [17], has been included as part of the WS-I Basic Profile [18]. Addressing support for SOAP within the substrate is thus central to our strategy. Subsequent sub-sections describe our approach to providing support for Web services within the substrate.

### **3.1. Incorporate the SOAP processing stack into the substrate**

By incorporating the SOAP processing stack into the substrate applications residing in different hosting environments (C++ based gSOAP, .NET-based WSE, or Perl-based SOAP::Lite) can interact with the substrate. Furthermore, so long as these Web services are connected to the substrate they can partake from all the Qualities of Service accorded to the NB clients. This includes features such as failure resilience and recovery from failures. This approach requires the substrate to function as a SOAP node which conforms to the SOAP processing model governing the actions that need to be taken upon receipt of a SOAP message. Specifically in SOAP 1.2 the substrate needs to deal with the role (in SOAP 1.1 this corresponds to the actor attribute), mustUnderstand and the relay attributes. The substrate will issue a fault if the message contains any headers targeted to its role, with the mustUnderstand attribute set, which it cannot process.

Finally it must be noted that the substrate may forward or interact with other SOAP intermediaries inside or outside the substrate to accomplish certain functions. The SOAP 1.2 model allows the relay attribute to be incorporated into SOAP message headers to facilitate such an interaction. In some cases, such as WS-Eventing [19] and WS-Notification [20], the substrate can provide support for delegated interactions such as information regarding the list of topics, management of subscriptions and their lifetimes, and replays of notification messages to recovering endpoints. Another related capability is that of a *proxy* where the substrate can interact with other Web services on behalf of a non-Web service endpoint.

### **3.2. Provide services for SOAP messages**

The substrate can provide a variety of services to SOAP messages. This includes support for compressing and decompressing, fragmenting and coalescing data encapsulated in the SOAP body, and logging of messages for subsequent replays among others. A substrate operates in variety of roles. A SOAP message can include such processing directives for the substrate through SOAP headers targeted to it as a SOAP intermediary. Additionally the substrate can provide support for conversion between different encoding schemes that may be employed in a SOAP message.

### **3.3. Facilitate federation between specifications**

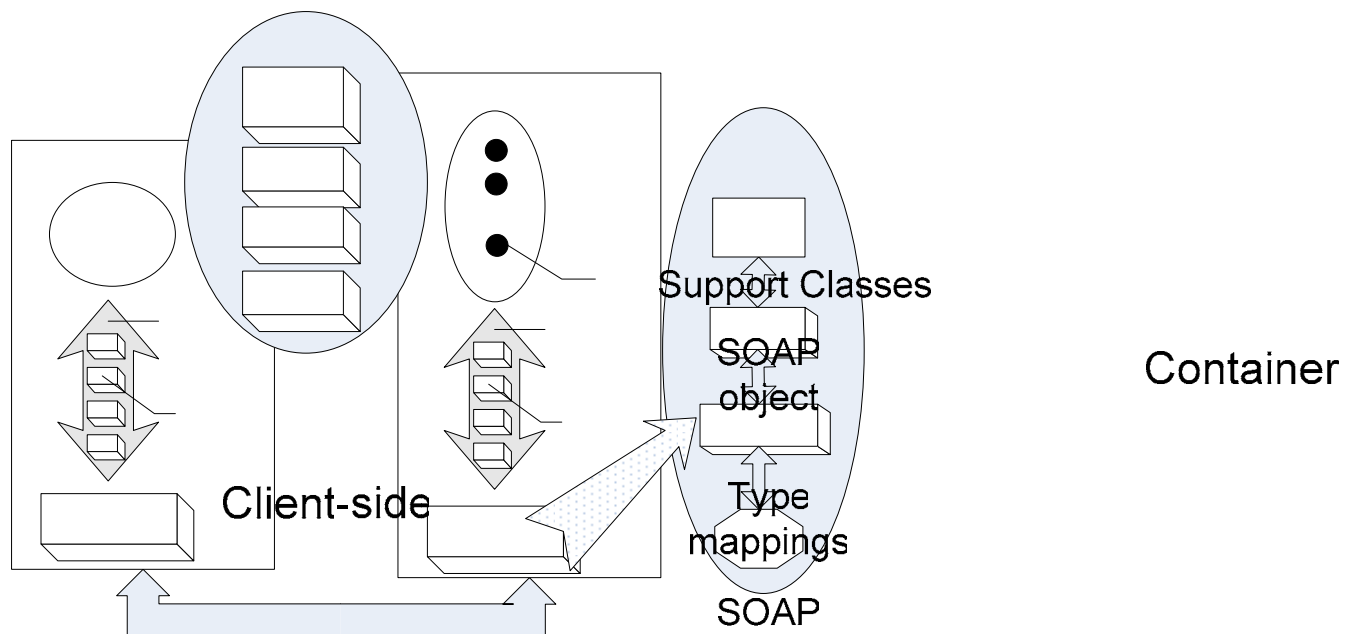
The substrate can facilitate federation between competing specifications in the same target area. Examples of such scenarios include WS-ReliableMessaging [21] and WS-Reliability [22] in the reliable delivery area and WS-Eventing and WS-Notification in the area of notifications. Such a federation would enable service endpoints from competing specifications to interoperate with each other. This capability requires the substrate to map not only the structural elements of the SOAP messages but do so while ensuring that the semantics encapsulated within the original message are also mapped accordingly. It is entirely possible that in some cases it might not be possible to find a semantically equivalent operation in a target specification; here we may either throw faults or provide for custom extensions. The substrate incorporates support for Web Services specifications such as WS-Eventing and WSRM. Applications may choose to leverage these stand-alone Web Service capabilities available within the substrate.

### **3.4. Facilitate the discovery of services**

The substrate can facilitate the discovery of services hosted within the substrate. Services advertise their services in an XML based schema. The substrate provides support for XPath queries and regular expressions based queries; this capability can be used to discover services that satisfy the search criteria. Individual queries may also constrain the realms within which the discovery should occur. This allows an entity to control how localized the services should be.

### **3.5. Permeating service endpoints: The filter approach**

In this section we include a brief description of the typical deployment of services and accesses to these services. We also discuss extensions that most hosting environments provide for augmenting the behavior and functionality of service endpoints. This lays the groundwork for our strategy for making the substrate permeate service endpoints.



**Figure 1: Deployment of service endpoints and filter pipelines**

To facilitate incremental addition of capabilities to service endpoints once can also configure filters in the processing path between the service endpoints. Since the service endpoints communicate using SOAP messages these filters operate on SOAP messages. Several of these filters can be cascaded to constitute a *filter pipeline*. Services are generally hosted within a hosting environment also known as a *container*. The container provides a variety of services which the service implementation can avail of. For example, a service implementation need not worry about communication details since this necessary functionality would be implemented within a *container component* such as servlets in the Java J2EE environment. This component in tandem with the container *support classes* is responsible for packaging data received over the wire into data structures that can be processed by the service implementation. An instance of the web component is typically automatically generated by the container during the *deployment* phase of the Web service. This scenario is depicted in **Figure 1**. It is possible to deploy services without a container. In the simplest case one may simply use the TCP protocol for communications and reconstruct SOAP messages from byte packets received over a socket; a custom deployment component can be used to configure filter pipelines.

Filters within a pipeline operate on SOAP messages encapsulating invocation requests or responses. In the case of a service this pipeline is configured between container component and the service, while in the case of clients this is configured between the support classes and the application logic. It should also be noted that individual filters are autonomous entities that have access to the entire SOAP message encapsulating the request/invocation. Individual filters are allowed to modify only the header and body elements of SOAP messages. The order in which filters operate on messages needs to be consistent, for example the stages at which encryption/decryption, compression/decompression take place at the service endpoints should be consistent otherwise unpredictable results/behavior may ensue.

There are three advantages to utilizing the filter approach. First, it entails no changes to the service endpoints: this facilitates incremental addition of capabilities. Second, filters can be developed and tested independent of the service endpoints thus providing greater robustness. Finally, the filter approach promotes code reuse since different filters corresponding to security, compressions, logging or timestamps can be utilized by multiple services.

The substrate can provide additional capabilities by permeating a service endpoint. Specially designed filters allow the incremental addition of capabilities to existing services. These filters encapsulate several of the substrate's capabilities and in some cases allow for richer interaction with the substrate. A heart-beat filter would send a message at regular intervals to the substrate indicating that it is alive, this in turn helps discovery services within the substrate to identify live service instances. A performing monitoring filter would in turn notify the substrate at regular interval about the load that it is experiencing. This in turn allows the substrate to load balance service



requests by routing them to the least overloaded service instance. A filter may also automatically generally service advertisements along with the support of transports available at a service. Additionally, these filters can also leverage the substrate's capabilities to communicate across NAT, firewall and proxy boundaries.

### **3.6. Negotiate and provide an optimal transport for SOAP messages**

The substrate provides support for a very wide array of transports (TCP, UDP, Multicast, SSL, HTTP and ParallelTCP among others). Depending on the size of SOAP message and the nature of continuing interactions appropriate transports will be deployed for communications. The nature of continuing interactions are dictated by issues such as whether the service exchanges messages at a high rate for a long time or whether the service considers reliable delivery to be more important than timely delivery. Filters at an endpoint can negotiate the best possible transport between itself and the substrate. The choice of the transport protocol being deployed is a function of the reliability, volume, rate and security requirements at the endpoint. The transport negotiations are carried out using a set of SOAP messages some of which are used to determine performance metrics such as latency, bandwidth, loss rates and jitters.

Note that the SOAP messages being transported can be based either on the traditional RPC style request/response message or the asynchronous one-way messaging. In the former case of RPC the substrate will facilitate correlations between requests and responses over transports, such as UDP, that do not naturally support a request/response based interaction that is at the heart of HTTP. The substrate will generate a UUID for such messages and include this as a header in the SOAP message. This message identifier when included in responses allows correlation with the original request.

### **3.7. Message Dispatching and enforcement of policies**

Here the substrate sitting at the edge of the organization facilitates the dispatch of messages to the endpoint identified in the WSA endpoint-reference. The substrate also facilitates the creation of these endpoint-references. The dispatcher generates these endpoint references such that they reflect the optimal protocol to reach these endpoints. Also, the substrate can enforce policies that govern the processing of SOAP messages. Here the substrate can automatically add policy information to outbound messages, such a move would allow applications to incorporate policy changes without entailing code rewrites.

## **4. Grid of Grids**

A Grid is not really formally defined, but we may view it as a collection of capabilities provided by different organizations that have banded together to form a "Virtual Organization." A capability is just a Web service, and Grids may be built from collections of Web services. A Grid service is just a Web service, although it may follow more restrictive conventions defined by OGSA. It is actually better to define a Grid by how it is used rather than how it is built. In this section we investigate some of the issues involved in building Grids of Grids.

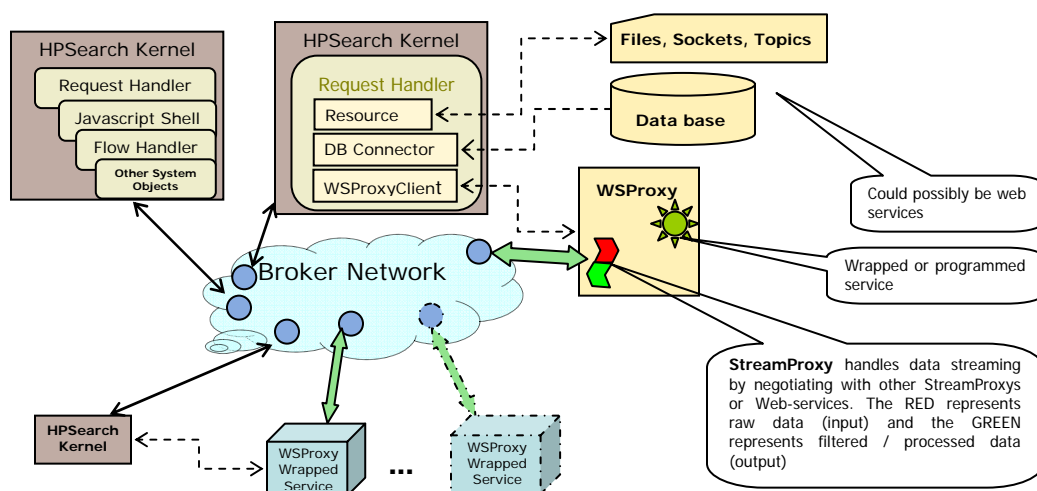
We recommend two sets of services to facilitate such a scenario. Services provided within the substrate constitute the Internet-on-Internet (IOI) services. It is referred to as IOI since it enables us to build an application-level "Internet" of services connected by a messaging substrate that replicates in the application layer many of the desirable features (security, guaranteed delivery, optimal routing) that are normally found in the TCP/IP stack. See [\[23\]](#) for a discussion of why TCP/IP is not enough and thus why IOIs are necessary for SOAP messages. These services have been described in detail in [sections 2 and 3](#). The IOI services will be invisible to the applications that run in it. Applications would simply specify the QoS constraints and the substrate would deal with the complexity of satisfying these constraints. There are a number of higher level services and capabilities that do not belong in the IOI layer: these services typically extend the capabilities available through the IOI layer and are more specifically needed for Web service management and apply to specific domains. Typical examples include service information and metadata management. We refer to this collection of capabilities as the Context and Information Environment (CIE). CIE services broadly fall into the following [5](#) categories.

1. Collaboration: Some collaborative applications may place a premium on the ability to pause/replay live streams rather than timely delivery. It is easy to see how the buffering strategies may vary in such scenarios. Strategies for the demarcation and subsequent retrieval of major and minor events may vary in different domains.
2. Authorization and authentication interfaces: Depending on the domain authentication schemes may span the wide spectrum from bio-metrics to text-based passwords. Same is true for trust propagation.

3. Support for specifications in various domains: Prime examples of this include WS-Discovery which is suitable for ad-hoc networks, and WS-Context which maintains contexts for a distributed computation.
4. Metadata Management: Different domains may have different formats for storing metadata and constraints regarding their exchange. In some scenarios custom solutions may be used or some endpoints may choose to use WS-Metadata exchange which facilitates exchange of metadata between two end points.
5. Portal Services: This involves allowing access to all metadata, the management of system deployments, firewall tunnels, performance information, and error-logs. Additionally a portal service may aggregate a set of services and provide a domain specific view of the state of these services.

As we have discussed in the introduction, messaging substrates are important given the asynchronous and collective nature of composite science application services. Building on this messaging system, we need to provide simple ways of expressing these composite application services and the data streaming pipes that connect them in scripting languages. Apart from scripting workflows, such scripting languages and engines have two important advantages: first, they provide a high level language suitable for science application developers to use to program the grid; and second, they provide tools for creating and managing messaging systems.

The HPSearch system assumes that all data that flows through is essentially a stream and uses NaradaBrokering to route these data streams. In effect, processing is done by passing data through various programs that are accessible as services, in a Pipe-Filter fashion. This architecture is popular in UNIX systems. In this architecture every component (also called as a filter) has a set of input and output ports. The filter transforms or filters the data it receives on its input ports. The processed data is sent out on the output port to the next filter. The pipe connecting the two filters is a directional stream of data usually represented by a data buffer to store all the data until the next filter has time to process it.



The system consists of one or more “*HPSearch Kernels*” which contain a Javascript based console application, *FlowHandlers* (for distributing flow components), *RequestHandlers* (for controlling specific types of resources that are components of the flow) and other system objects. Every *Kernel* is itself a Web-service and exposes ports to remotely run scripts, query system information and perform other management tasks. These *Kernels* communicate via messages using the brokering network (shown by black double-headed arrows). The presence of multiple such *Kernels* serves the following purposes a) distributing the handling of tasks for load-balancing b) utilizing a kernel (*RequestHandler* component of *Kernel*) nearest to the resource in question for minimizing the data transfer overhead and c) certain local resources (such as files) may not be accessible, or certain hosts may not be given permission to access resources (for example, for security purposes, only specific hosts may be allowed access to database) on a specific host unless the *RequestHandler* component runs on that particular host.

The *Kernel* runs a *RequestHandler* for every component it handles, such as reading and writing to files, sockets, topics or reading from databases, reading from files using http/ftp and controlling web-service proxys (shown by thick dashed lines). Each of these resources might be accessible using web-services and could be reachable via an endpoint reference. We plan to add a more generic form of addressing every type of resource by implementing support for WS-Addressing mechanisms. An interesting component of the *RequestHandler* is *WSProxyClient* that allows us to control the functioning of *WSProxy* (Web Service Proxy wrapper for components processing streaming data), as explained below. The *WSProxyClient* controls the instantiation of the *WSProxy* service and its operation (shown by the thick dashed line) via normal web-service calls (simple SOAP requests). Since the *WSProxy* is a normal web-service, it can be controlled by any workflow engine and hence is workflow engine and workflow language independent.

The *WSProxy* is essentially a wrapper over an existing application or a data processing code. The *WSProxy* can be deployed in any standard web-service container such as Apache AXIS. The *WSProxy* a) exports life-cycle operations as web-service operations so that the deployed service may be controlled by standard means (such as by sending simple SOAP requests), b) maintains state of the application and allow it to recover from faults in the event of failure, and c) notifies the controlling application (*WSProxyClient* or workflow engine) of errors and other notifications that the wrapped service may produce. A *StreamProxy* (shown by the red + green symbol in *WSProxy*) is a wrapper over data streams. The *WSProxy* may utilize a *StreamProxy* to help negotiate ideal transport characteristics whenever possible (based on the discussion outlined in [section 3.6](#)).

*WSProxy* encapsulates a service using two interfaces (*Runnable* and *Wrapped*). *Runnable* is suited to quickly creating data filtering applications and provide more control on the life-cycle operations of the service. *Wrapped* provide less control on the lifecycle of the service but allows us to wrap an existing code for creating a pluggable component and exposing it as a web service. *Wrapped* service provides best results if the service being wrapped reads data from STDIN and writes data to STDOUT.

When a user submits a script containing the data flow specification and creates a *Flow* object, the shell invokes the *FlowHandler* which in conjunction with other *Kernels*, decides the best *Kernel* to handle the individual components of the flow. These task descriptions are then distributed to the individual *Kernels* who in turn invoke a *RequestHandler* to process the request. The *RequestHandler* is responsible for handling errors and notifications (if any) from the resource it is handling and taking the appropriate action.

We can compose a distributed data-flow by joining multiple *WSProxy* wrapped services and setting the correct input and output streams. The brokering network can be used to handle the data communication between the services.

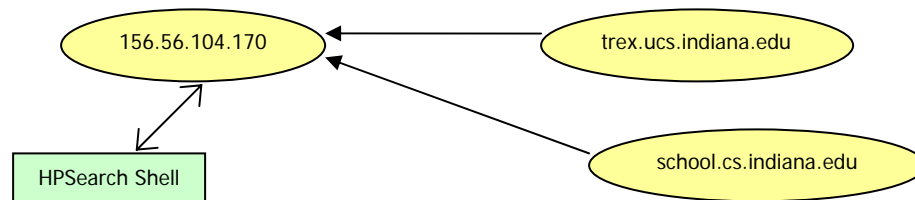
## 5.1. Managing the Messaging Substrate

HPSearch provides an ideal console for system management. By providing appropriate bindings (objects) to various system management operations we can control various features of the brokering substrate from instantiation to management and dynamic manipulation of brokering network characteristics. We highlight here the major parts of this effort: broker topology creation and performance measurements.

**Broker Topology Creation:** Interface to the broker locator service to locate existing brokers and instantiate new brokers for efficient routing. This binding may be combined with the performance metrics and objects for creating new links between brokers to achieve higher throughput by avoiding busy routes.



The NaradaBroker object serves as a scripting front-end to instantiate a broker and to create links between brokers. A partial implementation is available. For example, the following code listing creates a linear topology consisting of three brokers



**Figure 3: Instantiating brokers and creating a broker topology**

```

b = new NaradaBroker("school.cs.indiana.edu");
b.create("");
b_connLink = b.connectTo("156.56.104.170", "5045", "t", "");
b.requestNodeAddress(connLink, "0");

c = new NaradaBroker("trex.ucs.indiana.edu");
c.create("");
c_connLink = c.connectTo("156.56.104.170", "5045", "t", "");
c.requestNodeAddress(c_connLink, "0");
  
```

This is also useful in dynamically creating a virtual broker network for a particular application and then deploying the application over the virtual network

**Measuring Broker Performance:** The PerfMetrics object can be initialized to read the performance metrics published by the Performance Monitoring Service (described in Section 2 above) on a specialized topic for performance data (such as /cgl/narada/perfdata). Further the accumulated metrics may be queried using the query function. As an illustration, the following code queries the accumulated metrics to find the link with an average latency greater than 5.0 and then re-queries to find the jitter for the link.

```

badLink = PerfMetrics.query("//link[avgLatency > 5.0]/@id");
jitter = PerfMetrics.query("//link[@id='" + badLink[0] + "']/jitter");
  
```

Here we can use XPath expressions to query the performance metrics.

## 6. Future Work

In the Grid services community there are currently two frameworks that seek to deploy Grid applications using Web services. The dominant OGSF framework was recently factored into a set of Web service specifications: WSRF. WSRF focuses primarily on providing support for stateful interactions. The central tenet of WSRF is that services have state and these services are modeled as resources with the ability to inspect their properties and lifetimes. WSGAF [25] in turn considers services to be stateless, and provides a set of design patterns to model Grid applications using widely-accepted Web service specifications. We plan to investigate the ability to reconcile the differences between the aforementioned approaches so that applications may themselves be composed of services conforming to either specifications but can continue to deal with each other as if they were part of the framework with which they are aligned.

More recently Microsoft has released a set of specifications – WS-Transfer, WS-Enumeration and WS-Eventing – which deal with the ability to have stateful interactions. We expect this approach to coexist alongside WSRF for the foreseeable future. We plan to investigate issues related to the ability to interoperate between these specifications. Finally, the other area that we plan to research further pertains to the management of lifecycle and metadata of service instances in different settings under dynamic usage scenarios.

## 7. References

- [1] [OGSA] I. Foster, C. Kesselman, J. Nick, S. Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.” Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. Available from <http://www.globus.org/research/papers/ogsa.pdf>.

- [2] [WSA] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture." W3C Working Group Note 11 February 2004. Available from <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.
- [3] [WSRF] I. Foster (ed), J. Frey (ed), S. Graham (ed), S. Tuecke (ed), K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, S. Weerawarana, "Modeling Stateful Resources with Web Services v. 1.1." March 5, 2004. Available from <http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf>.
- [4] [SERVO1] Andrea Donnellan, Jay Parker, Geoffrey Fox, Marlon Pierce, John Rundle, Dennis McLeod [Complexity Computational Environment: Data Assimilation SERVOnet](#) 2004 Earth Science Technology Conference June 22 - 24 Palo Alto.
- [5] [SERVO2] Andrea Donnellan, Jay Parker, Greg Lyzenga, Robert Granat, Geoffrey Fox, Marlon Pierce, John Rundle, Dennis McLeod, Lisa Grant, Terry Tullis [The QuakeSim Project: Numerical Simulations for Active Tectonic Processes](#) 2004 Earth Science Technology Conference June 22 - 24 Palo Alto.
- [6] [SC2004] Geoffrey Fox, Shrideep Pallickara and Savas Parastatidis [Towards Flexible Messaging for SOAP Based Services](#). To appear, proceedings of ACM/IEEE Conference on Supercomputing Applications 2004.
- [7] [nb-ref-1] The NaradaBrokering Project at the Community Grids Lab: <http://www.naradabrokering.org>
- [8] [nb-ref-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.
- [9] [nb-perf Ref] Shrideep Pallickara et. al. Performance of a Possible Grid Message Infrastructure. (To appear) Journal of Concurrency and Computation: Practice & Experience. UK e-Science meeting on Grid Performance Edinburgh, UK.
- [10] [nb-routing] 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.
- [11] [nb-reliable-del] 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. 2004.
- [12] [nb-grid-ftp] 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.
- [13] [nb-ftp] Hasan Bulut, Shrideep Pallickara and Geoffrey Fox. Implementing a NTP-Based Time Service within a Distributed Brokering System. ACM International Conference on the Principles and Practice of Programming in Java. Pp 126-134.
- [14] [nb-sec] Pallickara et al. A Security Framework for Distributed Brokering Systems. Available from <http://www.naradabrokering.org>.
- [15] [soap] M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>
- [16] [wsdl] Web Services Description Language (WSDL) 1.1 <http://www.w3.org/TR/wsdl>
- [17] [uddi] Universal Description, Discovery and Integration UDDI.
- [18] [wsi] Web Services Interoperability <http://www.ws-i.org/>
- [19] [ws-eventing] Web Services Eventing (WS-Eventing) Microsoft, IBM and BEA. <http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf>
- [20] [ws-notify] Web Services Notification (WS-Notification). IBM, Globus, Akamai et al. <http://www-106.ibm.com/developerworks/library/specification/ws-notification/>
- [21] [wsrm] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) <ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf>
- [22] [wsr] Web Services Reliable Messaging TC WS-Reliability. <http://www.oasis-open.org/>
- [23] [Vogels2003] W. Vogels, "Web Services Are Not Distributed Objects." IEEE Internet Computing, vol. 7 (6), pp59-66, 2003.
- [24] [hpsearch] HPSearch: Design & Development via Scripting. <http://www.hpsearch.org>
- [25] [WSGAF] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "WS-GAF: A Framework for Building Grid Applications Using Web Services." To appear in *Concurrency and Computation: Practice and Experience*.