Messaging in Web Service Grid with Applications to Geographical Information Systems

Geoffrey Fox, Shrideep Pallickara, Galip Aydin and Marlon Pierce
(gcf, spallick, gaydin, marpierc)@indiana.edu
Community Grids Lab, Indiana University

1. Introduction

As standards such as SOAP 1.2, WSDL 2.0, and WS-Addressing become widely implemented and deployed, the initial concepts and implementations of Web Services as "remote procedure calls for the Web" are giving way to a more message-oriented, service-oriented approach. Such systems place an emphasis on managing secure, reliable messages that may be delivered in any number of ways across multiple routing SOAP intermediaries.

As we discuss in this article, all communications in SOA-based systems are messages. Further, a powerful way to implement these systems is to place the service "islands" on a software-level messaging substrate that implements efficient routing, security, reliability and other qualities of service. As we will show, such systems support messages of all types, from infrequent update notification events to continuous streams. We suggest that in the complex evolving technology scene today, not only services but their collection into systems of higher functionality should be as decoupled as possible in architecture and tight timing constraints. This we call the principle of building "Grids of Grids of Simple Services"

Many important Grid applications in real-time data mining involve all of these message types. We discuss a GIS (Geographical Information System) example from our SERVOGrid (Solid earth Research Virtual Observatory) work that uses the NaradaBrokering messaging system for managing data streams from GPS stations. We are in the process of connecting these to RDAHMM, a time series data analysis program useful for mode change detection. These streaming services form one sub-Grid in the "Grid of Grids" system supporting solid earth science and also containing (sub)-Grids involving code execution services and information/metadata services.

2. Service Oriented Architectures for Grids

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. Typical basic (atomic) services include data access (logically wrapping storage technologies such as databases and file systems) and the ability to run and manage remote applications. More complicated services may be composed of these basic services using workflow expression languages coupled with workflow engines.
2. Services communicate using messages. Messages are usually encoded using SOAP [26]. 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 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.

These issues have direct 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 with real time streaming data [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, as we discuss below.

In our discussion of these topics, we emphasize the message orientation of SOAs and Web Service Grids. Messages may range from infrequent notification events to remote method invocations/responses to streaming data. Various Web Service specifications (Sections 2 and 3) are intended to provide the underpinnings of such a system, allowing for asynchronous communication, reliability, message routing, and security. We discuss our implementation of these specifications on top of NaradaBrokering, a general purpose messaging software substrate (Sections 4 and 5). The implication of SOA principles is that service implementations will proliferate in various domains, which may be composed in various ways for specific applications. We have dubbed this composition of services as a "Grid of Grids," discussed in Section 6. Finally, in Sections 7 and 8 we present an example application within a Grid of Grids: streaming data control of GPS stations to support real-time data mining.

3. Messaging in Web/Grid Environments

Messaging is a fundamental primitive in distributed systems. Entities communicate with each other through the exchange of messages, which can encapsulate information of interest such as application data, errors and faults, system conditions, search and discovery of resources. A related concept is that of *notifications* where entities receive messages based on their registered interest in certain occurrences or situations. Messaging and notifications are especially important in the Service Oriented Architecture (SOA) model engendered by Web Services. Here, Web Services interact with each other through the exchange of messages.

In this section, we first discuss message exchange patterns and also briefly review two specifications in the area of notifications. There are two main entities involved in a notification: the *source* which is the generator of notifications and the *sink* which is interested in these notifications. A sink first needs to register its *interest* in a situation, this operation is generally referred to as a *subscribe* operation. The source first wraps *occurrences* into notification messages. Next, the source checks to see if the message satisfies the constraints specified in the previously registered subscriptions. If so, the source routes the message to the sink. This routing of the message from the source to the sink is referred to as a notification. It should be noted that there could be multiple sources and sinks within the system. Furthermore, each sink could register its interests with multiple sources, while a given source can manage multiple sinks. The complexity of the subscriptions registered by a sink could vary from simple strings such as "Weather/Warnings" to complex XPath or SQL queries. Some application examples are given in Section 8.

We take the point of view that all communications within an SOA-based Grid should be treated as messages. This applies equally well to event notifications as to data streams. The capabilities of the messaging substrate, and the associated Web Service standards that define qualities of service, may be applied equally to all of these messages.

3.1 WSDL Message Exchange Patterns

Messaging is fundamental to Web Services, and WSDL [6], which describes these services, facilitate the description of various message exchange patterns (hereafter MEP) that are possible between service endpoints. Since these MEPs are defined to be part of the WSDL document, any node wishing to interact with the service knows both the *sequence* and the *cardinality* of messages associated with a given WSDL operation. WSDL 1.1 defined a basic set of MEPs; this has been expanded upon in WSDL 2.0.

WSDL 1.1 describes four MEPs defining the sequence and cardinality of abstract messages -- *In, Out, Fault* – that are part of a WSDL operation. The MEPs governing the exchanges between a service **S** and a node **N** are *one-way*, *request/response*, *notification* and *solicit*. A one-way message comprises a single *Out* message from a service **S** to node **N**. A request/response comprises an *In* message sent by a node **N** that is followed by an *Out* message by the service **S**. The notification MEP is simply an *Out* message from a service **S** to a node **N**. Finally, a solicit MEP is an *Out* message from service **S** followed by an *In* message from node **N**. It must be noted that the *Out* message in the notification MEP and the *In* message in the solicit MEP can also be a Fault message.

WSDL 2.0 has defined 4 additional MEPs *Robust In-Only*, *In-Optional-Out*, *Robust Out-Only* and *Out-Optional-In* which are extensions to the four MEPs that were defined in WSDL 1.1. These patterns occur because of the new fault propagation rules that are part of WSDL 2.0. The MEPs with the *optional* tag within them are patterns that comprise one or two messages, with the second message being a *Fault* that was triggered because of the first message in the pattern. The MEPs with the *robust* tag within them are patterns with exactly one message, however a fault may be triggered because of the first message.

3.2 WS-Eventing

Figure 3 depicts the chief components in WS-Eventing [7] a specification from Microsoft and IBM. When the sink subscribes with the source, the source includes information regarding the subscription manager in its response. Subsequent operations -- such as getting the status of, renewing and unsubscribing -- pertaining to previously registered subscriptions are all directed to the subscription manager. The source sends both notifications and a message signifying the end of registered subscriptions to the sink.

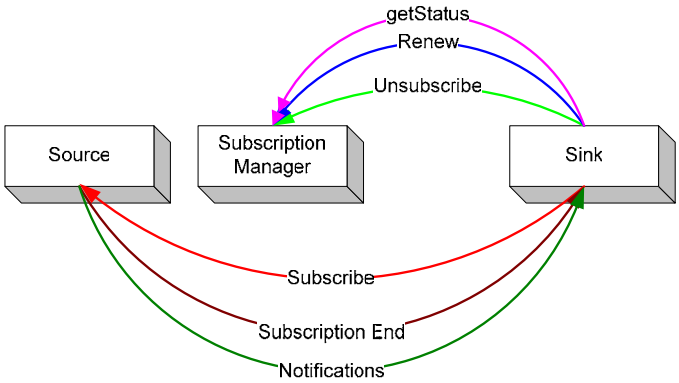


Figure 1: WS-Eventing - Chief components

3.3 WS-Notification

The WS-Notification specification refers to a set of specifications comprising WS-BaseNotification [8], WS-Brokered Notification [9] and WS-Topics [10]. WS-BaseNotification standardizes exchanges and interfaces for producers and consumers of notifications. WS-Brokered Notification facilitates the deployment of Message Oriented Middleware (MOM) to enable brokered notifications between

producers and consumers of the notifications. WS-Topics deals with the organization of subscriptions and defines dialects associated with subscription expressions; this is used in the conjunction with exchanges that take place in WS-BaseNotification and WS-Brokered Notification. WS-Notification currently also uses two related specifications from the WSRF specification; WS-ResourceProperties [11] to describe data associated with resources, and WS-ResourceLifetime [12] to manage lifetimes associated with subscriptions and publisher registrations (in WS-BrokeredNotifications).

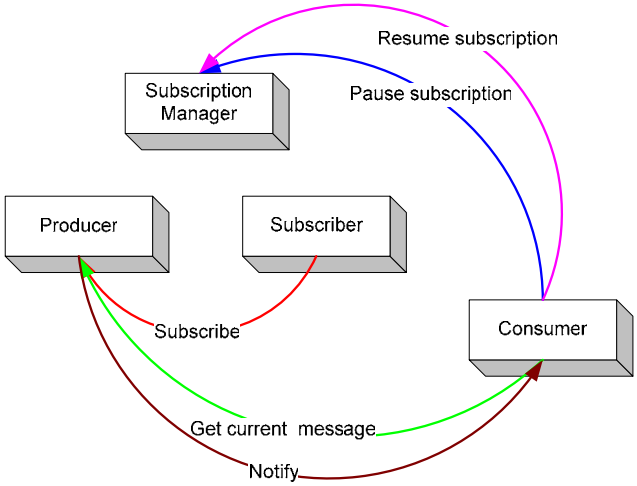


Figure 2: WS-BaseNotification - Chief components

Figure 1 depicts the chief components of the WS-BaseNotification specification. Also, depicted in this figure are the interactions (along with the directions) that these components have with each other. In WS-BaseNotification, a subscriber registers a consumer with a producer, which in turn includes information regarding the subscription manager in its response. Consumers can pause and resume subscriptions, with no messages being delivered while the subscription is in a paused state. Resumption of subscriptions after a pause can entail replay of all notifications that occurred in the interim. After a disconnect, either due to a scheduled downtime or failure, a consumer may also retrieve the last message issued by a producer. Finally, notifications from the producer are issued directly to the consumer. In WS-Notification each subscription is considered to be a resource (more appropriately a WS-Resource [13]). A consumer can use WS-ResourceLifetime or WS-ResourceProperties to manage lifetimes and properties associated with these subscriptions.

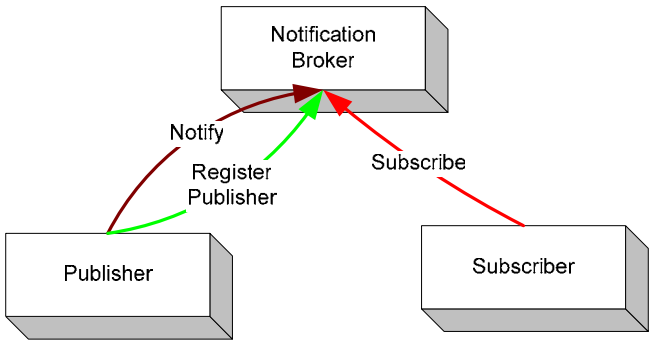


Figure 3: WS-BrokeredNotification - Chief components

Figure 2 depicts the chief components of the WS-BrokeredNotification specification. The notification broker interface performs the function of an intermediary between the producers and consumers of content. The broker is responsible for managing the subscriptions and also for routing the notifications to

the subscriber. Furthermore, the broker also maintains a topic space (based on the WS-Topics specification) that allows consumers to review the list of topics to which publishers publish. It should be noted that each topic is also a resource and can be inspected for its properties such as *dialect* and *topic expressions*.

4. Reliable Messaging in Web/Grid Environments

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 (hereafter WSR) [14] from OASIS and WS-ReliableMessaging (hereafter WSRM) [15] from IBM and Microsoft among others

4.1 WS-ReliableMessaging and WS-Reliability

The specifications – WSR and WSRM – both of which are based on XML, address the issue of ensuring reliable delivery between two service endpoints. Both the specifications use positive acknowledgements to ensure reliable delivery. This in turn implies that error detections, initiation of error corrections and subsequent retransmissions of "missed" messages can be performed at the sender side. A sender may also proactively initiate corrections based on the non-receipt of acknowledgements within a pre-defined interval. WSRM also incorporates support for negative acknowledgements which facilitates sender side error corrections.

The specifications also address the related issues of ordering and duplicate detection of messages issued by a source. A combination of these issues can also be used to facilitate exactly once delivery. Both the specifications facilitate guaranteed exactly-once delivery of messages, a very important quality of service that is highly relevant for transaction oriented applications; specifically banking, retailing and e-commerce.

Both the specifications also introduce the concept of a group (also referred to as a sequence) of messages. All messages that are part of a group of messages share a common group identifier. The specifications explicitly incorporate support for this concept by including the group identifier in protocol exchanges that take place between the two entities involved in reliable communications. Furthermore, in both the specifications the qualities of service constraints that can be specified on the delivery of messages are valid only within a group of messages, each with its own group identifier.
The specifications also introduce timer based operations for both messages (application and control) and group of messages. Individual and group of messages are considered invalid upon the expiry of timers associated with them. Finally, the delivery protocols in the specifications also incorporate the use of timers to initiate retransmissions and to time out retransmission attempts.

In terms of security both the specifications aim to leverage the WS-Security [16] specification, which facilitates message level security. Message level security is independent of the security of the underlying transport and facilitates secure interactions over insecure communication links.
The specifications also provide for notification and exchange of errors in processing between the endpoints involved in reliable delivery. The range of errors supported in these specifications can vary from an inability to decipher a message's content to complex errors pertaining to violations in implied agreements between the interacting entities.

5. Messaging Infrastructures for SOA

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. A more detailed discussion of these topics is given in [17]. 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. The Community Grids Lab has for several years been developing a messaging substrate NaradaBrokering [18]-24. NaradaBrokering is an open-source, distributed messaging infrastructure. The smallest unit of this distributed messaging infrastructure intelligently processes and routes messages, while working with multiple underlying communication protocols. We refer to this unit as a *broker*. In NaradaBrokering communication is asynchronous and the system can 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 on the size, rate, or scope of the interactions encapsulated within these events or the number of entities present in the system.

In NaradaBrokering we impose a hierarchical, cluster-based structure on the broker network [19]. This cluster-based architecture allows NaradaBrokering to support large heterogeneous client configurations. The routing of events within the substrate is very efficient [21] 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.

5.1 Services within Messaging Infrastructures

In messaging systems, entities should be able to specify constraints on the Quality of Service (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 delivery [22] 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.

5.1.1 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 NaradaBrokering 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.

5.1.2 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 [23]. 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.

5.1.3 Time and Buffering Services

Proper time sequence ordering of messages and events is of utmost importance in many applications, such as audio/video collaboration systems. The NaradaBrokering system provides this capability through an implementation of the Network Time Protocol (NTP). The NaradaBrokering TimeService [24] allows NaradaBrokering processes (brokers and entities alike) to synchronize their timestamps using the NTP 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.

5.1.4 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 [25] 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.

5.2 Broker Discovery

Since accesses to services are mediated through the distributed broker substrate it is essential that an entity connect to a broker that maximizes its ability to utilize the hosted services. Furthermore, since the broker network is a very dynamic and fluid system, where broker processes may join and leave the broker network at arbitrary times and intervals, it is not possible for an entity to assume that a given broker is available at all times. Static solutions to this problem might result in a certain known remote broker being accessed over and over again. This in turn causes degradations due to poor bandwidth utilizations. The broker discovery process in the NaradaBrokering substrate operates on the current state of the broker network and ensures that a discovered broker is the *nearest* available one; where nearest corresponds to network proximities or latencies. In this scheme newly added brokers within overloaded

broker *clusters* in the substrate are assimilated faster since the discovery process allows these brokers to be preferentially selected. This scheme thus allows brokers to be added to enable to the system to scale.

5.3 Support for Web/Grid Service specifications

The substrate has recently incorporated support for Web/Grid Services. The substrate incorporates support for several Web/Grid service specifications such as WS-Eventing, WS-ReliableMessaging and WS-Reliability. Work on the implementation of the WS-Notification suite of specifications is currently an on-going effort. It must be noted that almost all Web/Grid Service specifications leverage the SOAP [26] specification. We are currently also incorporating support for SOAP within the substrate. This would allow the substrate to perform certain services for SOAP messages, function as a SOAP intermediary, and also facilitate the routing of SOAP messages. Web/Grid Services can then send SOAP messages directly to the substrate. Another area that we intend to research further is the support for high-performance transport of SOAP messages.

6. Support for SOAP within messaging substrates

SOAP has emerged as the de facto standard for encapsulating and transporting various Web Services interactions. SOAP, along with WSDL and UDDI [27], 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. 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 QoS provided to the NaradaBrokering 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 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 and WS-Notification, 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.

6.1 Federation of competing specifications

The substrate can facilitate federation between competing specifications in the same target area. Examples of such scenarios include WS-ReliableMessaging (WSRM) and WS-Reliability 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.

6.2 Functioning as a SOAP intermediary

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.

6.3 Support for Filters/Handlers

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.

To facilitate incremental addition of capabilities to service endpoints one can also configure *filters* (examples include filters for encryption, compression, logging etc.) 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 use. 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 used to configure filter pipelines.

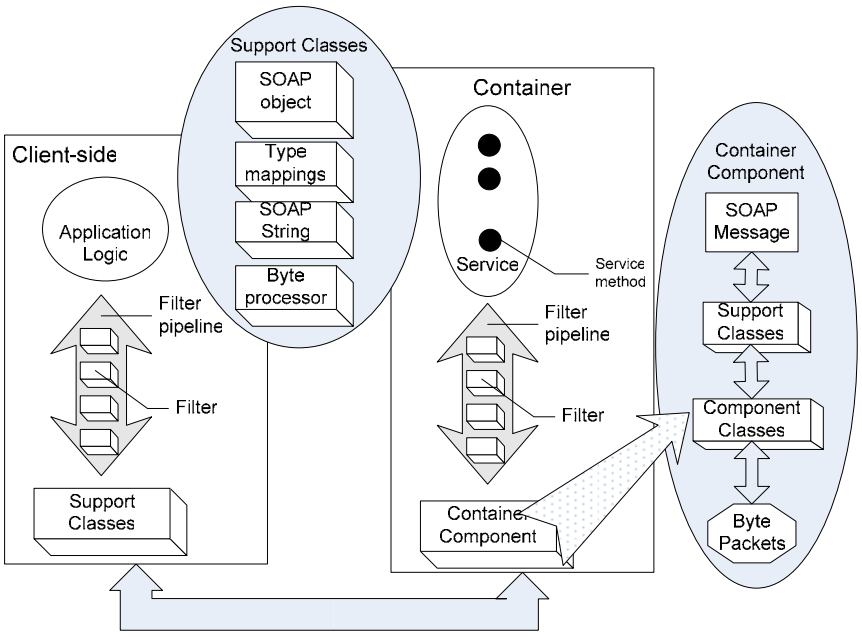


Figure 4: Deployment of services and filter-pipelines

Filters within a pipeline operate on SOAP messages encapsulating invocation requests or responses. In the case of a service this pipeline is configured between the 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 both 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 and 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 performance monitoring filter would in turn notify the substrate at regular intervals 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 information related to the transports available at the service endpoint. Additionally, these filters can also leverage the substrate's capabilities to communicate across NAT (Network Address Translator), firewall and proxy boundaries.

6.4 High performance transport of SOAP

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.

Another area that we intend to research further is the support for high-performance transport of SOAP messages. Here we will leverage the XML Infoset; by separating the SOAP message context from its XML syntax, we can freely move between the binary and classic angle-bracketed representations of SOAP messages without content loss. Another area for further investigation is the efficient binary

representations of XML Infosets such as SOAP Message Transmission Optimization Mechanism (MTOM) [28] and XML-binary Optimized Packing (XOP) [29]. We are developing schemes which allow two endpoints to first negotiate the best-available transport and then proceed to use it for transfers. To accommodate legacy systems that do not use the XML format, the Data Format Description Language (DFDL) [30] is an XML-based language that describes the structure of binary and character-encoded files and data streams so that their format, structure, and metadata can be exposed. This can also be used in tandem while transferring binary data using SOAP.

These latter topics are particularly important in several application areas. We have primarily been interested in efficient message representation in order to support PDA and other end clients, which are typically reached over much lower speed networks and have limited memory and processing power. However, the same ideas should scale up for Web Service-based scientific computing, since efficient message compression and high performance processing are required for moving non-trivial data sets and messages. Finally, these transport mechanisms are also important to real-time processing. As we have emphasized, all communications are messages moving through the substrate. These messages may range from infrequent events to remote method invocations to negotiated streams of time-sequenced data.

7. Grids of Grids

We may view it as a collection of capabilities provided by different organizations that have banded together to form a "Virtual Organization" [24]. 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 Ref [31] 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.

Grids of Grids are composed of applications and services from many different domains. In the next section, we take an extended example from Geographical Information Services, which combine streaming data sources with data filters and online data mining applications.

8. Applications to Earthquake Science and Geographical Information Systems Grids

8.1 A Motivating Example: Data Mining Live GPS Streams

SERVO Grid [4, 5] is a Grid system for earthquake modeling and simulation and includes a diverse set of applications, but we will focus on Robert Granat's Regularized Deterministic Annealing Hidden Markov Model (RDAHMM) application [32]. An earlier, non-streaming version of this application was discussed in [33]. RDAHMM may be used to analyze arbitrary time ordered data, such as GPS position measurements and seismic records, to identify underlying modes of the system. This occurs in three distinct methods of operation:

1. **Training and mode analysis:** this phase applies to historical data. The RDAHMM application is applied to a particular data set (i.e. an archived time sequence from a GPS station) in order to determine the historical modes. These may be compared to known physical processes, but the mode identification process does not involve fixed parameters.
2. **Change detection:** once RDAHMM has initially identified a system's historical modes (a one-time operation), it can be used to detect mode changes in new, incoming data streams. Typically we need one logical RDAHMM application per data source (i.e. GPS station). RDAHMM clones may be periodically retrained on the updated historical data sets.
3. **Event Accumulation and Notification:** Methods of operation (1) and (2) apply to specific data sources, but we will also be interested in network-wide events. For example, there may be several causes for individual GPS stations to undergo mode transitions, but simultaneous mode change events in several stations in the same geographic region may be associated with underlying seismic events. Such network-wide changes need to spawn additional notifications, to both humans and other application codes.

Data mining of live data streams [34, 35, 36, 37] is an important scientific Grid application in many areas of crisis management and homeland security. As we have outlined in the previous sections of this chapter, Service Oriented Architecture-based Grids implemented with Web Service standards will meet many of the requirements of real-time Grids, providing a system based open and extensible standards. As we have emphasized above, messages are a key component of SOA systems, and a software messaging substrate such as NaradaBrokering may be used to implement the qualities of service demanded by sophisticated Grids. As we emphasize again here, there is no difference between notifications, events, and data transfers from the point of view of the messaging substrate. Substrates such as the NaradaBrokering system may be applied equally well to problems in Web Service Eventing/Notification and to streaming data. Web Service Architectures likewise may be adapted to streaming applications and associated message patterns, just has they have been applied to remote procedure call-style patterns.

For the problem at hand, we may identify several important components, which we review in more detail below. First, the GPS station network is an example of a Geographical Information System (GIS). It requires a diverse set of services for such tasks as accessing archival data, accessing streaming data,

querying metadata that describes various members of the GPS network, and so on. These may be coupled to more traditional science Grid services for running and managing applications. Second, we have a diverse set of messages and services in the system: GPS stations provide streaming data, but we must also manage a) metadata services that describe individual stations in the network, b) less frequent messages (change events) that indicate a station has changed modes (which may occur only a few times per year), c) other GIS services for generating maps used for user interfaces; and d) services for managing application codes (such as RDAHMM) that are in the loop.

8.2 Geographic Information Systems and GIS Grids

Advances in Internet and distributed systems helped academia, governments and businesses to provide access to a substantial amount of geospatial data. The GIS community must face the following challenges:

1. Adoption of universal standards: Over the years organizations have produced geospatial data in proprietary formats and developed services by adhering to differing methodologies;
2. Distributed nature of geospatial data: Because the data sources are owned and operated by individual groups or organizations, geospatial data is in vastly distributed repositories,
3. Service interoperability: Computational resources used to analyze geospatial data are also distributed and require the ability to be integrated when necessary.

The Open Geospatial Consortium, Inc (OGC) represents a major effort to address some of these problems. The OGC is an international industry consortium of more than 270 companies, government agencies and universities participating in a consensus process to develop publicly available interface specifications. OGC Specifications support interoperable solutions that "geo-enable" the Web, wireless and location-based services, and mainstream IT. OGC has produced many specifications for web based GIS applications such as Web Feature Service (WFS) [38] and the Web Map Service (WMS) [39]. Geography Markup Language (GML) [40] is widely accepted as the universal encoding for geo-referenced data. In addition to the more traditional HTTP request/response style services, the OGC is also defining the SensorML family of services [41].

The GIS community quite obviously represents a major sub-domain in the “Grid of Grids” picture. By architecting GIS services using Web Services, and be placing these services within a SOA messaging substrate, we may integrate GIS Grid Services with other applications. Our work on GIS services as Web Services is described in more detail in [4, 5].

GIS applications developed by various vendors and academic institutions have become more complex as they are required to process larger data sets, utilize more computing power and in some cases need to collect data from distributed sources. Traditionally GIS applications are data centric: they deal with archived data. However, with sensor-based applications gaining momentum the need of integrating real-time data sources such as sensors, radars, or satellites with high end computing platforms such as simulation, visualization or data mining applications introduces several important distributed computing challenges to GIS community.

Although commercial GIS applications provide various solutions to these problems, most of the solutions are based on more traditional distributed computing paradigms such as static server-client approaches. Traditional point to point communication approaches tend to result in more centralized, tightly coupled and synchronous applications which results in harder management practices for large scale systems. Modern large scale systems on the other hand require more flexible asynchronous communication models to cope with the high number of participants and transfer of larger data sets between them.

Defining a Common Data Format

The first step for building such services is to decide appropriate encodings for describing the data. The importance of the data format lies in the fact that it becomes the basic building block of the system which in turn determines the level of interoperability. Use of a universal standard like XML greatly increases the number of users from different backgrounds and platforms who can easily incorporate our data products into their systems. Furthermore, services and applications are built to parse, understand and use this format to support various operations on data. So in a sense the type and variety of the tools being used in the development and data assimilation processes depend on the format initially agreed.

For these reasons we use GML, a commonly accepted XML based encoding for geospatial data, as our data format in GIS-related applications. One important fact about GML is that, although it offers particular complex types for various geospatial phenomena, users can employ a variety of XML Schema development techniques to describe their data using GML types. This provides a certain degree of flexibility both in the development process and in the resulting data products. For instance, depending on the capability of the environment schema developers may exclusively use certain XML Schema types and choose not to incorporate more obscure ones because of incompatibility issues. As a result a particular geospatial phenomenon can be described by different valid GML schemas.

By incorporating GML in our systems as de facto data format we gain several advantages:

1. It allows us to unify different data formats. For instance, various organizations offer different formats for position information collected from GPS stations. GML provides suitable geospatial and temporal types for this information, and by using these types a common GML schema can be produced. (See http://www.crisisgrid.org/html/servo.html for sample GML schemas for GPS and Seismic data)
2. As more GIS vendors are releasing compatible products and more academic institutions use OGC standards in their research and implementations, OGC specifications are becoming de facto standards in GIS community and GML is rapidly emerging as the standard XML encoding for geographic information. By using GML we open the door of interoperability to this growing community.
3. GML and related technologies allow us to build general set of tools to access and manipulate data. Since GML is an XML dialect, any XML related technology can be utilized for application development purposes. Considering the fact that in most cases the technologies for collecting data and consecutively the nature of the collected data product would stay the same for a long period of time the interfaces we create for sharing data won't change either. This ensures having stable interfaces and libraries.

8.2.1 Data Binding

Establishing XML or some flavor of it as the default message/data format for the global system requires consideration of a Data Binding Framework (DBF) for generating, parsing, marshalling and un-marshalling XML messages. Marshalling and un-marshalling operations convert between XML-encoded formats and (typically Java) binding classes that can be used to simplify data manipulation.

Being able to generate XML instances and parsing them in a tolerable amount of time is one of the criteria while choosing such a framework, because message processing time would affect overall system performance as well as the performance of the individual XML processing component.

Another criterion to consider is the ability of the binding framework to successfully generate valid instances according to the Schema definitions. This is a major problem for DBFs since not all of the XML Schema types can be directly mapped to Object Oriented Programming constructs. Some of the XML Schema types (such as Substitution Groups which are heavily used in GML Schemas) do not correspond to types in Object Oriented world and this causes difficulties while processing the XML documents. Various Data Binding Frameworks offer different solutions, some of which are more elaborate than the other and depending of the nature of the data a suitable framework must be chosen.

8.2.2 Data Services

GIS systems are supposed to provide data access tools to the users as well as manipulation tools to the administrators. In principle the process of serving data in a particular format is pretty simple when it is made accessible as files on an HTTP or FTP server. But additional features like query capabilities on data or real-time access in a streaming fashion require more complicated services. As the complexity of the services grows, the client's chance of easily accessing data products decreases, because every proprietary application developed for some type of data require its own specialized clients. Web Services help us overcome this difficulty by providing standard interfaces to the tools or applications we develop.

No matter how complex the application itself, its WSDL interface will have standard elements and attributes, and the clients using this interface can easily generate methods for invoking the service and receiving the results. This method allows providers to make their applications available to others in a standard way.

The usefulness of Web Services is constrained by several factors. They can be used in several cases such as

- The volume of data transferred between the server and the client is not high. Actual amount of data can be transferred depends on a number of factors like the protocol being used to communicate or maximum allowed size by HTTP;
- Time is not a determining factor. Despite the obvious advantages, current HTTP-based implementations do not provide desirable results for systems that require fast response and high performance. This is simply due to the delays caused by data transfer over network, network constraints, and HTTP request-response overhead.

Most scientific applications that couple high performance computing, simulation or visualization codes with databases or real-time data sources require more than mere remote procedure call message patterns. These applications are sometimes composite systems where some of the components require output from others and they are asynchronous, it may take hours or days to complete. Such properties require additional layers of control and capabilities from Web Services which introduces the necessity for a messaging substrate that can provide these extra features.

9. SOPAC GPS Services: Real Time Streaming Support for Position Messages

To demonstrate the use of technologies discussed earlier we describe GPS Services developed for the Scripps Orbit and Permanent Array Center (SOPAC) GPS data networks. Two of SOPAC's GPS networks are distributed in San Diego Counties and Riverside/Imperial Counties, respectively, and provide publicly available data. Raw data from the GPS stations are continuously collected by a Common Link proxy (RTD server) and archived in RINEX files.

The data collected from the GPS stations are served in 3 formats:

- **RAW**: For archiving and record purposes, not interesting for scientific applications, not available in real-time.
- **RTCM**: Published real-time and no records are kept. This is useful for RTCM capable GPS receivers as reference.
- **Positions**: Positions of the stations. Updated and presented every second. GPS Time Series can be produced using these positions and they can be in different epochs such as hourly, daily, etc.

Position information is used by RDAHMM and other applications. The RTD server however outputs the position messages in a binary format called RYO. This introduces another level of complexity on the client side because the messages have to be converted from binary RYO format.

To receive station positions, clients are expected to open a socket connection to the RTD server. An obvious downside of this approach is the extensive load this might introduce to the server when multiple clients are connected.

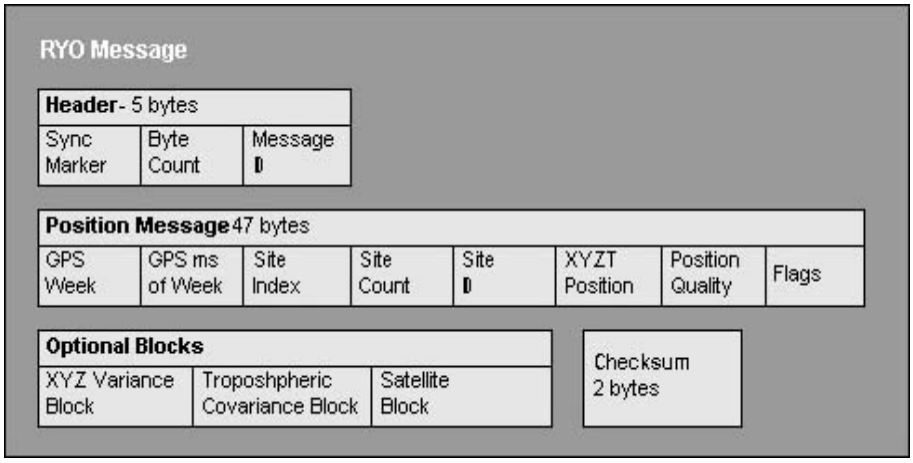
After the RTD server receives raw data from the stations it applies filters and for each network generates a message. This message contains a collection of position information for every individual station from which the position data has been collected in that particular instant. In addition to the position information there are other measurements in a message such as quality of the measurement, variances etc. For each GPS network, the RTD server broadcasts one position message per second through a port in RYO format. This is depicted on the left hand sides of Figures 5 and 6.

As we discuss below, to make the position information available to the clients in a real-time streaming fashion we are using the NaradaBrokering messaging system. Additionally we developed applications to serve position messages in ASCII and GML formats. This allows applications to choose the format that they want for applications will additionally allow us to implement more finely grained network subscriptions: users and applications don't have to process an entire network's stream to receive the subset of GPS stations that they want. RDAHMM provides a specific example for this: we need to apply RDAHMM change detection to individual GPS station signals.

9.1.1 Decoding RYO Messages

As shown in Figures 5 and 6, the incoming data streams must be converted into various formats. This is done by using developed specialized services that subscribe to specific topics and republish the decoded data to topics associated with the new format.

For example, the RYO Message Type 1 starts with a 5-byte Header which is followed by a 47-byte GPS Position message. Three types of optional blocks may follow the Position Message and a 2-byte checksum is located at the end of the message.



A non-blocking Java Socket connection is made to RTP server to collect RYO messages. We use thread programming techniques for this purpose. An RYO Decoder application which uses binary conversion tools converts RYO messages into text messages. Furthermore since we do not expect clients to know about the GPS time format we convert GPSWeek and GPSmsOfWeek values to Gregorian calendar format (i.e. 2005-19-07/04:19:44PM-EST). Additionally since we anticipate some clients to expect position information in terms of Latitude and Longitude, we calculate Latitude, Longitude and Height values from XYZT Position.

9.1.2 GML Schema for Position Messages and Data Binding

We have developed a GML conformant Schema to describe Position Messages. The Schema is based on RichObservation type which is an extended version of GML 3's Observation model. This model supports Observation Array and Observation Collection types which are useful in describing SOPAC Position messages since they are collections of multiple individual station positions. We follow strong naming conventions for naming the elements to make the Schema more understandable to the clients.

We used Apache XML Beans for data binding purposes: these convert ASCII data streams into XML. SOPAC GML Schema and sample instances are available here: http://www.crisisgrid.org/schemas

9.1.3 Integrating NaradaBrokering with Streaming GPS Measurements

After we have services for decoding position information into three different formats we may integrate these services with NaradaBrokering to provide real-time access to data. The following figures depict the use of NaradaBrokering topics in the system. Figure 5 depicts the flow of data to interested subscribers: applications like RDAHMM, databases for permanent storage, and portal systems (such as QuakeSim) for human interaction. To support these various consumers, we must provide different versions of the data stream.

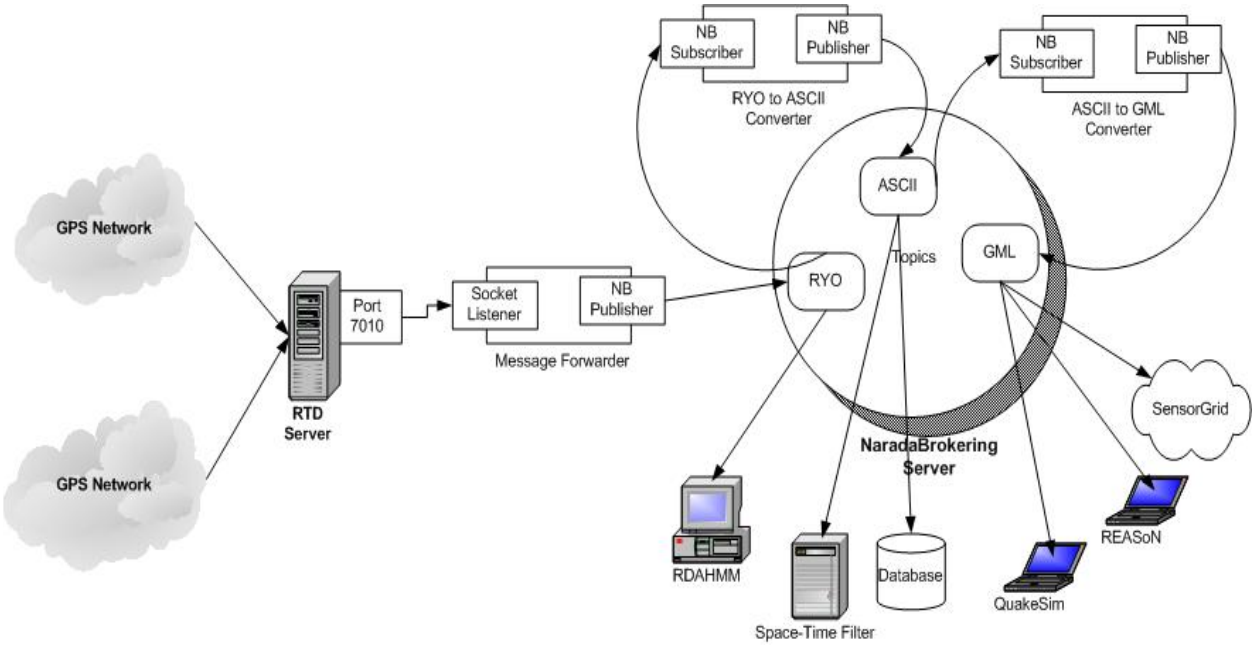


Figure 5 GPS streams are delivered to interested end clients.

Figure 6 expands Figure 5 to illustrate the basic routing techniques. The GPS network data streams are collectively made available by Scripps through ports 7010 and 7011. These two ports serve all the data from two distinct networks, each with 15 stations. The data is published in RYO format. We intercept this data through Java proxies that act as publishers on the topics RYO1 and RYO2 to a NaradaBrokering node. Subscribers to this topic may be any number of applications capable of handling these binary formats, including translation programs. As shown in Figure 6, these streams are translated into ASCII text formats by RYO Decoders. These decoders then publish the data back to the broker network on new topics, Positions/Text1 and Positions/Text2 in the figure. Any number of listening applications may receive this data, including (as shown in the figure), GML Converters that transform the ASCII streams into GML suitable for GIS applications.

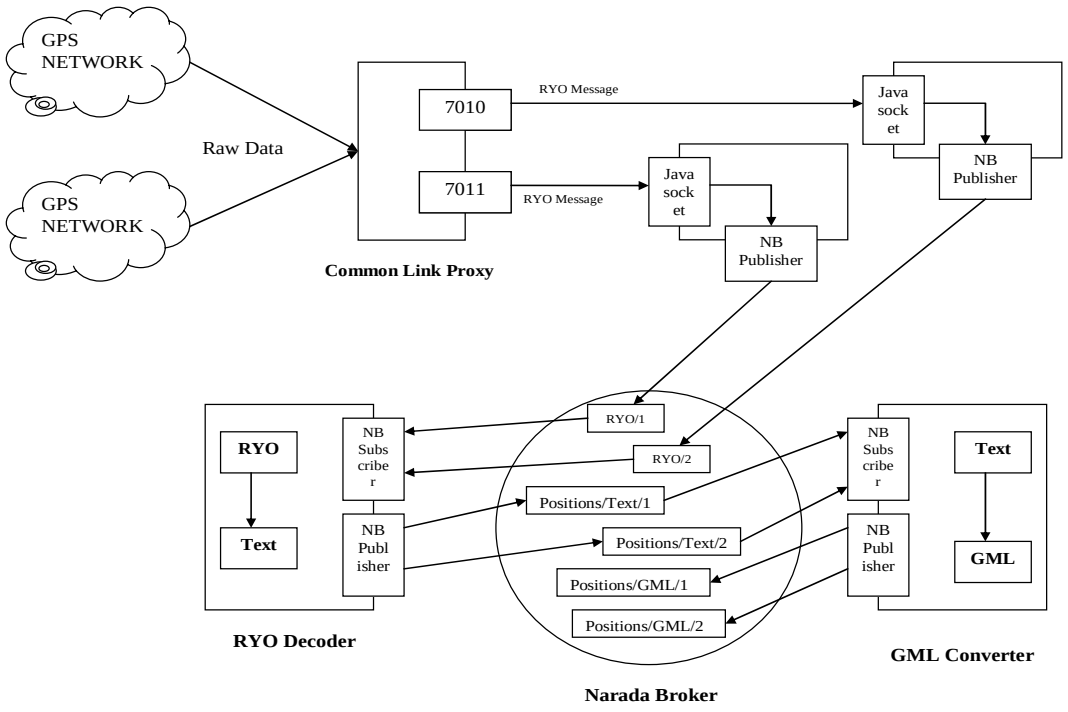


Figure 6 GPS network integrated with NaradaBrokering.

Currently the system is being tested for San Diego Counties and Riverside/Imperial Counties GPS networks. The following tables show the current information for NaradaBrokering Server and topic names:

NaradaBrokering Server address: *xsopac.ucsd.edu:3045*

Format	Topic Name
RYO	*SOPAC/GPS/Positions/SanDiego/RYO*
Text	*SOPAC/GPS/Positions/SanDiego/Text*
GML	*SOPAC/GPS/Positions/SanDiego/GML*

San Diego County

Format	Topic Name
RYO	*SOPAC/GPS/Positions/Riverside/RYO*
Text	*SOPAC/GPS/Positions/Riverside/Text*
GML	*SOPAC/GPS/Positions/Riverside/GML*

Riverside/Imperial County

We may add more filters to the data and develop more finely grained topics. For example, after decoding the binary stream, we may publish the individual GPS station data streams to individual topics.

9.2 Building a Sensor Grid

We are developing a Service Oriented Architecture to support real-time integration of sensor data with scientific applications such as simulation, visualization or data mining software.

Scientific applications that require processing of huge data sets are increasing in number with the evolution of computing resources, network bandwidth, and storage capabilities etc. At the same time some of the applications are being designed to run on real-time data to provide near-real time results; such applications are gaining ground in systems like Crisis Management or Early Warning Systems because they allow authorities to take action on time. Earthquake data assimilation tools are good examples of this group since they use data from Seismic or GPS sensors. However, in SERVO Grid, most of these tools currently consume data from repositories and they do not have access to real-time data due to several reasons.

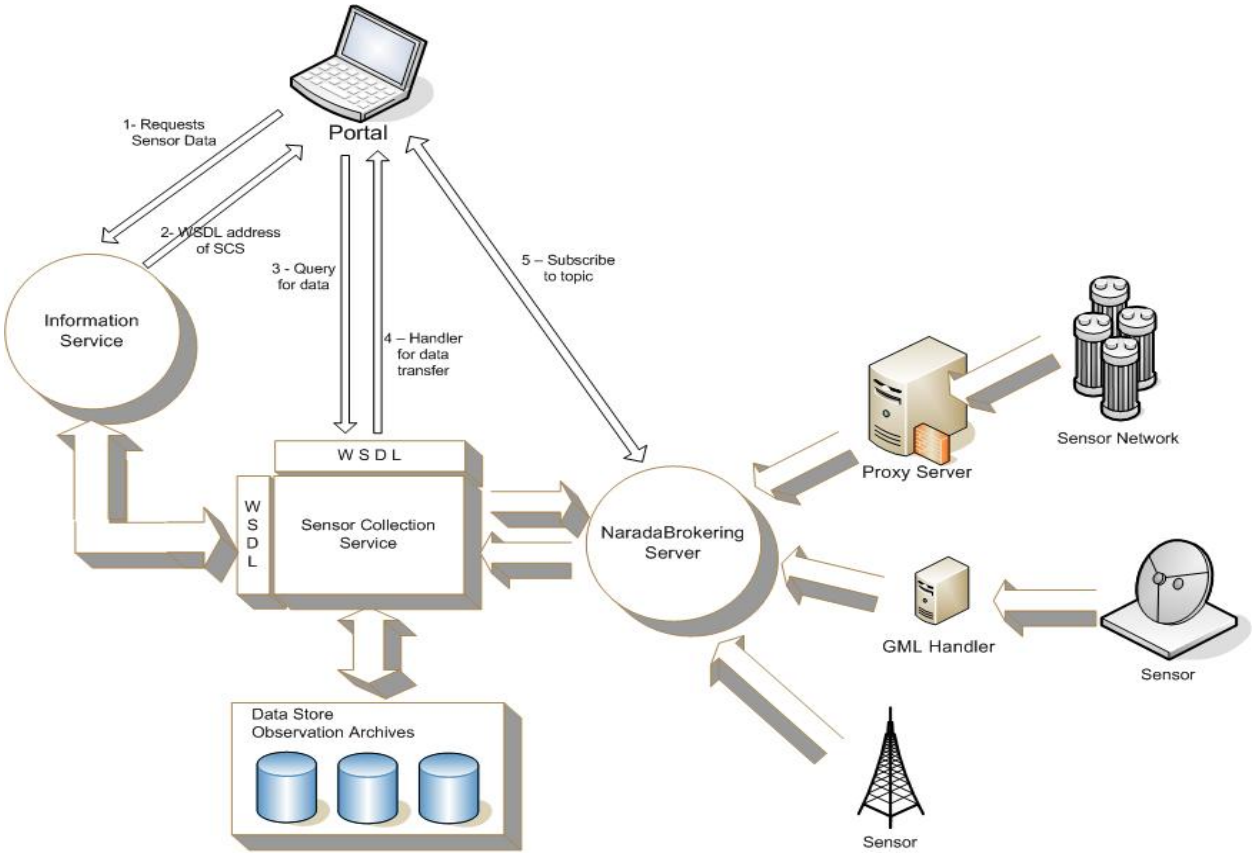


Figure 7Sensor Grid services integrated with streaming data sources.

A Sensor Grid architecture will couple data assimilation tools with real-time data using GIS standards and Web Services methodologies. The system will use NaradaBrokering as the messaging substrate and this will allow high performance data transfer between data sources and the client applications. The Standard GIS interfaces and encodings like GML will allow data products to be available to the larger GIS community.

Figure 7 shows the major components of Sensor Grid using SensorML components. The client discovers the related Sensor Collection Service (SCS) information by using search interfaces provided by Information Service (IS). IS returns a handler which contains the WSDL address of the SCS that has access to the particular sensor client requests. The client then sends a getData query to SCS. Depending on the nature of the query SCS may take two actions; if the query is for archived sensor data then it requests data from the Observation Archives and returns it to the client. But if the client wants to access real-time data then it returns a data handler which contains the broker information and topic name for the sensor. Also depending on the size of the archived data SCS may choose one of two options for data transfer; if

the result size is relatively small then it is returned via SOAP message, otherwise NaradaBrokering is used. SCS also keeps information about the sensors themselves. This information is encoded in SensorML. After receiving the broker address and the topic name, client may subscribe to the NaradaBrokering server to receive real-time data.

10. Conclusions

As standards such as SOAP 1.2, WSDL 2.0, and WS-Addressing become widely implemented and deployed, the initial concepts and implementations of Web Services as "remote procedure calls for the Web" are giving way to a more message-oriented, service-oriented approach. Such systems place an emphasis on managing secure, reliable messages that may be delivered in any number of ways across multiple routing SOAP intermediaries.

As we have discussed in this article, all communications in SOA-based systems are messages. Further, the correct way to implement these systems is to place the service "islands" on a software-level messaging substrate that implements efficient routing, security, reliability and other qualities of service. As we have shown, such systems support messages of all types, from infrequent update notification events to continuous streams.

Many important Grid applications in real-time data mining involve all of these message types. We have discussed a GIS example from our SERVOGrid work that uses the NaradaBrokering messaging system for managing data streams from GPS stations. We are in the process of connecting these to RDAHMM, a time series data analysis program useful for mode change detection. These streaming services are part of a more comprehensive system involving code execution services and information/metadata services.

11. References

[1] 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] 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] 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] Andrea Donnellan, Jay Parker, Geoffrey Fox, Marlon Pierce, John Rundle, Dennis McLeod Complexity Computational Environment: Data Assimilation SERVOGrid 2004 Earth Science Technology Conference June 22 - 24 Palo Alto.

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

[7] Web Services Eventing. Microsoft, IBM & BEA. http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf

[8] Web Services Base Notification (WS-BaseNotification). IBM, Globus, Akamai et al. ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf

[9] Web Services Brokered Notification Notification (WS-BrokeredNotification). IBM, Globus, Akamai et al. ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BrokeredN.pdf

[10] Web Services Topics (WS-Topics). IBM, Globus, Akamai et al. ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-Topics.pdf

[11] WS-Resource Properties. IBM, Globus, USC et al. http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf
[12] Web Services Resource Lifetime. IBM, Globus, USC et al. http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf
[13] 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.
[14] Web Services Reliable Messaging TC WS-Reliability. http://www.oasis-open.org/
[15] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) ftp://www6.software.ibm.com/software/devel oper/library/ws-reliablemessaging200403.pdf
[16] Web Services Security. OASIS. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
[17] 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.
[18] The NaradaBrokering Project at the Community Grids Lab: http://www.naradabrokering.org
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] Pallickara et al. A Security Framework for Distributed Brokering Systems. Available from http://www.naradabrokering.org.
[26] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. (2003), SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation 24 June 2003. Available from http://www.w3c.org/TR/soap12-part1/
[27] Universal Description, Discovery and Integration UDDI.
[28] SOAP Message Transmission Optimization Mechanism. Microsoft, IBM and BEA. http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/
[29] XML-binary Optimized Packing. Microsoft, IBM and BEA. http://www.w3.org/TR/2005/REC-xop10-20050125/ .
[30] The Data Format Description Language (DFDL) working group. https://forge.gridforum.org/projects/dfdl-wg/.
[31] W. Vogels, "Web Services Are Not Distrubted Objects." IEEE Internet Computing, vol. 7 (6), pp59-66, 2003.
[32] Robert Granat, Regularized Deterministic Annealing EM for Hidden Markov Models, Doctoral Dissertation, University of California Los Angeles, 2004.
[33] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Robert Granat A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications

Proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference (CCGrid 2005). Cardiff, UK May 2005.
[34] Robert L. Grossman, Yunhong Gu, Chetan Gupta, David Hanley, Xinwei Hong, and Parthasarathy Krishnaswamy, Open DMIX: High Performance Web Services for Distributed Data Mining, 7th International Workshop on High Performance and Distributed Mining, in association with the Fourth International SIAM Conference on Data Mining, 2004.
[35] Nguyen, D., et al. *Real-Time Feature Extraction for High Speed Networks*. in *15th International Conference on Field Programmable Logic and Applications*. 2005. Tampere, Finland.
[36] Zaki, M.J., *Parallel and Distributed Association Mining: A Survey.* IEEE Concurrency, Special Issue on Parallel Mechanisms for Data Mining, Dec. 1999. **7**(4): p. 14-25.
[37] Han, J. and M. Kamber, *Data Mining: Concepts and Techniques*. 2001: Morgan Kaufmann.
[38] Vretanos, P (ed.) (2002), Web Feature Service Implementation Specification, OpenGIS project document: OGC 02-058, version 1.0.0.
[39] de La Beaujardiere, Jeff, Web Map Service, OGC project document reference number OGC 04-024.
[40] Cox, S., Daisey, P., Lake, R., Portele, C., and Whiteside, A. (eds) (2003), OpenGIS Geography Markup Language (GML) Implementation Specification. OpenGIS project document reference number OGC 02-023r4, Version 3.0.
[41] Sensor Model Language (SensorML) Project Web Site: http://vast.nsstc.uah.edu/SensorML/.