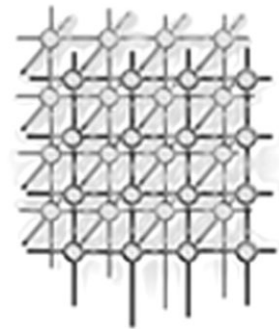


Management of real-time streaming data Grid services

Geoffrey Fox, Galip Aydin, Hasan Bulut,
Harshawardhan Gadgil, Shrideep Pallickara,
Marlon Pierce*,† and Wenjun Wu



*Community Grids Laboratory, Indiana University,
501 North Morton Street, Suite 224, Bloomington, IN 47404, U.S.A.*

SUMMARY

We discuss our message-based approach to managing real-time data streams and building higher level services to produce and consume them. Our messaging system acts as a substrate that can be used to provide qualities of service to various streaming applications ranging from audio–video collaboration systems to sensor Grids. The messaging substrates are composed of distributed, hierarchically arranged message broker networks. Services such as filters are deployed along the edges of the network. We discuss the role of management systems for both broker networks and filter services: broker network topologies must be created and maintained, and distributed filters must be arranged in appropriate sequences. These managed broker networks may be applied to a wide range of problems. We discuss applications to audio–video collaboration in some detail and also describe applications to streaming Global Positioning System data streams. These provide specific application filters that can transform and republish message streams to the broker system. Copyright © 2006 John Wiley & Sons, Ltd.

Received 13 March 2006; Accepted 4 May 2006

KEY WORDS: real-time systems; Web services; Grid computing

1. INTRODUCTION

A growing number of applications involve real-time streams of information that need to be transported in a dynamic, high-performance, reliable, and secure fashion. Examples include sensor nets for both science and military applications, mobile devices on *ad-hoc* networks, and collaborative applications.

*Correspondence to: Marlon Pierce, Community Grids Laboratory, Indiana University, 501 North Morton Street, Suite 224, Bloomington, IN 47404, U.S.A.

†E-mail: marpier@indiana.edu

Contract/grant sponsor: U.S. Department of Energy (SciDAC Program); contract/grant number: DE-FC02-02ER2615

Contract/grant sponsor: NASA (Advanced Information Systems Technology Program); contract/grant number: 1253656



In the latter case, the streams consist of a set of ‘change events’ for a collaborative entity multicast to the participating clients. They could be the frames of audio–video streams, encoded changed pixels in a shared display, or high-level semantic events such as signals of PowerPoint slide changes. Here, we describe our research into ways of managing such streams that we think are a critical component of applications as diverse as sensor nets and real-time synchronous collaboration environments.

We develop real-time streaming technology assuming that the sources, sinks, and filters of these streams are Web or Grid services. This allows us to share the support technology between streaming applications and to benefit from the pervasive interoperability of a service-oriented architecture. Furthermore, this allows a simple model of collaborative Web and Grid services acquired by ‘just’ sharing the input or output ports. As services expose their change by explicit messages (using what we call a message-based Model–View–Controller architecture [1]), it is much easier to make them collaborative than traditional desktop applications whose change is often buried in the application. We have shown how traditional collaborative applications can be made service oriented with, in particular, a set of services implementing traditional H.323 functionality and interoperating with Access Grid and Polycom systems. This required the development of an XML equivalent of the H.323 protocol [2,3]. Our other major motivation is the sensor networks of military, scientific, and social infrastructure. These are well-suited to a service architecture as exemplified by the U.S. Department of Defense’s Global Information Grid with its service-based NCOW (Network Centric Operations and Warfare Architecture) [4–8].

We have developed general-purpose, open-source software to support distributed streams, described in Section 2. NaradaBrokering [9,10] forms a distributed set of message brokers that implement a publish-subscribe software overlay network. This environment supports multiple protocols (including UDP, TCP, and parallel TCP) and provides reliable message delivery with a scalable architecture. The infrastructure can supply message-oriented middleware support for Web services through its support of WS-Eventing, WS-Addressing, WS-Reliable Messaging, and WS-Reliability. We note the architecture links both peer-to-peer and Grid paradigms and has been used in either mode, although the peer-to-peer experience is limited.

We have used several applications that drive the development of our technology. These include collaboration services with audio, video, and shared display streams, as well as linkages of real-time Global Positioning System sensors to Geographical Information Systems implemented as Web services. Other examples include integration of hand-held devices into a Grid [11] and the linkage of annotations to video streams showing how composite streams can be supported for real-time annotation [12]. The first two applications are described in Sections 4 and 5 and illustrate the need for high-level session-and-filter infrastructure on top of the messaging infrastructure.

There are several technical challenges to building infrastructure that is efficient and can support time-sensitive operations such as the instant replay and annotation of sensor streams in an emergency. One needs to archive events (such as video frames) to support replay. The location of replicated archives (for fault tolerance) and the linkage to annotation material illustrate system metadata that is very dynamic but must be accessible to the system with low latency. This requires metadata approaches that we discuss separately [13], optimized for dynamic responsiveness in modest-sized subsystems rather than approaches such as distributed hash tables optimized for scalability.

Our architecture supports the interesting concept of hybrid streams where multiple ‘simple streams’ are intrinsically linked; examples are linkage of a stream of annotation whiteboards with original audio–video stream [12] and the combination of lossless and lossy codec streams (using potentially parallel TCP and UDP, respectively) to represent a large dynamic shared display.



The messaging infrastructure supports the application services with their filters, gateways, and sessions, reflecting both the collaborative and workflow functions. However, we have found the need for a set of services that manage the messaging itself and hence, control broker deployment and Quality of Service (QoS). This is discussed in Section 3, which describes the integration of the management of messaging and higher-level services.

2. NARADABROKERING: A DISTRIBUTED MESSAGING SUBSTRATE

NaradaBrokering [9,10,14] is a messaging infrastructure that is based on the publish/subscribe paradigm. The system efficiently routes messages [15] from the originators to the consumers that are interested in the message. The system places no restrictions on the size and the rate at which these messages are issued. Consumers can express their interests (or specify subscriptions) using simple formats such as character strings. Subscriptions may also be based on sophisticated queries involving XPath, SQL, or regular expressions. Support for these subscription formats enables consumers to precisely narrow the type of messages in which they are interested. The substrate incorporates support for enterprise messaging specifications such as the Java Message Service [16]. The substrate also incorporates support for a very wide array of transports (TCP, UDP, Multicast, SSL, HTTP, and ParallelTCP, among others) that enable the infrastructure to be leveraged by entities in a wide variety of settings. To cope with very large payloads, the system leverages ParallelTCP at the transport level and services such as compression and fragmentation to reduce individual message sizes. The fragments (compressed or otherwise) are reconstituted by appropriate services (coalescing and de-compression) prior to delivery to the application.

The most fundamental unit in NaradaBrokering is a message. A stream can be thought of as being composed by a series of messages, each with causal and ordering correlations to previous messages in the stream. The interbroker latency for routing typical messages is around 1 ms. In a controlled cluster setting a single broker was found to support up to 400 UDP-based audio–video clients concurrently with adequate latency [17]. Among the services most relevant for collaboration within the system are the following.

1. Support for a replay and recording services. The recording service is used to store messages reliably to the archival system. The recording is done in such a way that all events issued by the recording entity are stored in the order in which they were published. The replay service facilitates the replay of these previously stored messages. The replay service support replays in multiple flavors. Entities may request replays based on sequencing information, timing information, content of the message, or based on the topics to which these messages were published. In some cases one or more of the parameters can be combined in a single request.
2. Support for consistent global timestamps [18] through an implementation of the Network Time Protocol (NTP). This implementation ensures that timestamps at the distributed entities are within a few milliseconds of each other. This allows us to ensure that we can order messages based on these global timestamps. This is especially useful during replays when we can precisely determine the order in which messages should be released to the application.
3. Support for buffering and subsequent time-spaced release of messages to reduce jitters. The typical lower bound for time space resolution is a millisecond. However, we have also been able to successively time-space events in the order of several microseconds. By buffering and releasing messages, we reduce the jitters that may have been introduced by the network.

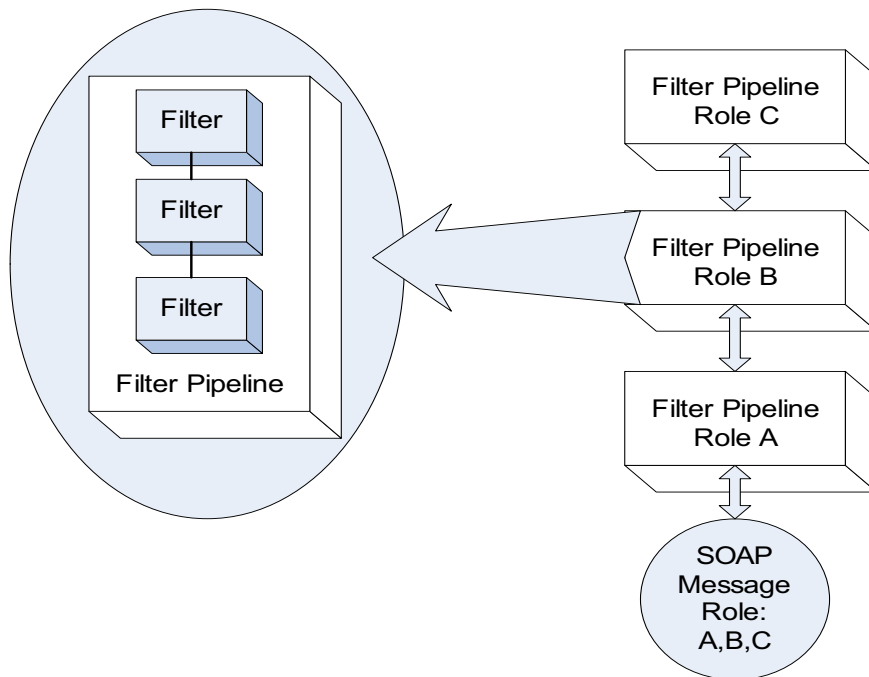


Figure 1. Filter pipelines are used to process SOAP messages in NaradaBrokering.

More recently, we have incorporated support for Web services within the substrate. Entities can send SOAP messages directly to the brokers that are part of the messaging infrastructure. We have incorporated support for Web service specifications such as WS-Eventing [19], WS-ReliableMessaging [20] and WS-Reliability [21]. Work on implementing the WS-Notification [22] suite of specifications is currently underway. The implementation of these specifications also has had to cope with other specifications such as WS-Addressing and WS-Policy that are leveraged by these applications. In addition to the rules governing SOAP messages and the implemented protocols, rules governing WS-Addressing are also enforced.

In our support for SOAP within NaradaBrokering we have introduced filters and filter-pipelines (Figure 1). A filter is the smallest processing unit for a SOAP message. Several filters can be cascaded together to constitute a filter-pipeline. Here, the filters within a filter-pipeline can be dynamically shuffled and reorganized. The system allows a filter-pipeline to be registered for every role that the node (functioning as a SOAP intermediary) intends to perform.

Upon receipt of a SOAP message that is targeted to multiple roles (as indicated by the SOAP 1.2 role attribute) the corresponding filter-pipelines are cascaded so that the appropriate functions are performed. The SOAP message is first parsed to determine the roles that need to be performed. Next, we check to see if there are any pipelines registered for a specific role. The scheme allows developers



to develop their own filters and filter-pipelines and target them for specialized roles. For example, in some cases a developer may wish to develop a filter that performs message transformations between the competing notification specifications: WS-Eventing and WS-Notification. By providing an extensible framework for the creation of filters and the registration of roles, sophisticated applications can be built.

3. HPSEARCH: MANAGING BROKER NETWORKS AND SERVICE GRIDS

As discussed in the previous section, NaradaBrokering provides a software messaging infrastructure. In a related project, we have developed HPSearch [23] as a scripting-based management console for broker networks and their services. At one end of the spectrum are services that help manage the messaging middleware, while at the other end are services that leverage capabilities of the middleware (WSProxy). The management of both sets of services is handled by a scripting medium that binds Uniform Resource Identifiers (URI) to the scripting language. By binding URI as a first-class object we can use the scripting language to manage the resource identified by the URI. We discuss these functions in detail below.

Management of messaging middleware

In order to deploy a distributed application that uses NaradaBrokering, the middleware must be set up and a broker network topology must be deployed. Broker network topology may also be changed at runtime using HPSearch by adding or deleting links between brokers. Once the middleware is set up, we leverage the broker network to deploy the distributed application as detailed in the next section.

To fulfill this requirement we have been developing a specialized Web service called the Broker Service Adapter (BSA) that helps us deploy brokers on distributed nodes and setup links between them. The BSA is a Web service that enables management of the middleware via WS-Management [24]. Errors and other conditions are similarly handled and notified to the management engine using WS-Eventing [19]. We discuss this management architecture in more detail in Section 3.1.

Management of data streams and services

HPSearch uses NaradaBrokering to route data between components of a distributed application. This data transfer is managed transparently by the HPSearch runtime component, the Web Service Proxy (WSProxy) [25]. Thus, each of the distributed components is exposed as a Web service which can be initialized and steered by simple SOAP requests. WSProxy can either wrap existing applications or create new data processing and data filtering services. WSProxy handles streaming data transfer using NaradaBrokering on behalf of the services, thus enabling streaming data transfer for any service. The streaming data is enabled using NaradaBrokering middleware, a distributed routing substrate. Thus, there are no central bottlenecks and failure of a broker node routes the data stream through alternate routes, if available. Furthermore, NaradaBrokering supports reliable delivery via persistent storage [26] thus enabling guaranteed delivery for data streams.

All interactions with the WSProxy are made using simple SOAP calls, and thus the service wrapped by the WSProxy can be steered using any client that can talk SOAP. The WSProxy service has



been written using Apache Axis and can be deployed in any Web service container such as Apache Tomcat. HPSearch adds initialization and steering of WSPProxy-based Web services using a scripting interface. HPSearch creates temporary topics using the NaradaBrokering system and correctly ties up all components, thus establishing a data flow between various components of the distributed application. More details on how a distributed application can be modeled using WSPProxy and deployed and managed using the HPSearch system are presented in [27].

3.1. Requirement for management

We build our management architecture on service-oriented architecture principles. The essential entity requiring management is the NaradaBrokering broker. Management of a broker entails configuring the broker to use specific transports, start services, and initialize the broker. Furthermore, a single broker can only effectively serve a limited number of clients. Tests [28] show that a single broker can service 1500 simultaneous audio streams or 400 simultaneous video streams before the quality of service starts deteriorating. In order to improve the scalability of the system, one frequently employs a network of brokers. Various network connection topologies exist, each differing in their scalability and fault-tolerance characteristics. For instance, a fully connected network of brokers provides the maximum fault-tolerance but incurs an overhead on the number of links between nodes. A linear topology would be much less demanding in terms of resources; however, it would partition the network should a link or node fail. Thus, selection of a particular topology is dependent on the application needs, and therefore we provide a framework to rapidly deploy broker network topologies via WS-Management.

3.2. WS-management architecture

Our initial architecture is based on the June 2005 draft of WS-Management [29]. The architecture comprises mainly of the managed entities (brokers), the service adapter (which is a framework to allow processing of WS-Management messages), and the manager (an entity that requests specific management actions). The service adapter is a wrapper over the managed entity that provides a WS-Management interface on the outside and a managed entity specific interface on the inside. Its job is to translate WS-Management commands to entity-specific management actions.

WS-Management allows management of specific resources associated with an entity. This may be specified using the `ResourceURI` header defined in WS-Management. The framework primarily allows creation and deletion of resources using the verbs defined in the WS-Transfer [30] specification. We allow creation of brokers and links between brokers. The `Delete` capability provides flexibility in terms of tearing down the existing broker network and redeploying the broker network with, possibly, a different configuration. Configuration of brokers can be queried and manipulated using the `Get` and `Set` verbs, respectively. WS-Management also provides the capability of enumerating contents of large containers such as log files via the WS-Enumeration [31] interface. We are investigating if this capability can be utilized for managing the broker network. WS-Management also provides the capability of heartbeat messages to determine liveness of managed entities. Such a capability is crucial to the correct functioning of architecture and we are working towards employing WS-Eventing [32] based notifications to enable heartbeat events.

To use this basic architecture we also employ the capability of using different transports. Using SOAP over HTTP is the norm; however, this might be restrictive in cases where the service



endpoint is not directly accessible. To mitigate this situation we employ a set of bootstrap broker nodes and wrap the SOAP messages as NaradaBrokering events. These events are then routed appropriately to the correct endpoint. When the processing is done, the response is routed back to the service requestor. As described above, NaradaBrokering provides support for different transports such as TCP, UDP, NIOTCP, HTTP, SSL. This capability provides us with the ability to use multiple transports in case a specific transport is not available. Specifically, it allows us to manage brokers behind firewalls by leveraging the HTTP tunneling capability in NaradaBrokering.

Finally, to tie up all the components (manager, service adapter, managed entity), we employ a registry that stores runtime metadata. The information maintained in this registry is usually the management tasks to be carried and a list of available service adapters and their respective endpoint addresses.

3.3. Results

We have tested our architecture to deploy a network of eight brokers in a linear fashion. Our tests indicate that the average overhead is about 75 ms per broker. This is attributed to marshaling and unmarshaling of SOAP messages and their associated responses along with the network overhead. This overhead may be reduced by employing faster methods [33–35] in dealing with SOAP messages. Gadgil *et al.* [36] provide detailed test results.

4. GLOBAL-MMCS: AUDIO AND VIDEO STREAM SERVICES AND MANAGEMENT

In the previous two sections we have outlined the core messaging software and its management system. We now turn to applications that can be built on this substrate. Global-MMCS, as a service-oriented multimedia collaboration system, mainly processes multimedia streams: video, audio, whiteboard, and so on. ‘Events’ in video or audio are usually called video frames or audio samples. Generally speaking, there are many similarities between multimedia streams and other data streams such as sensor data. All streaming data require significant QoS constraints and dynamic filtering. These are both particularly demanding and well-understood for multimedia streams for both communication and processing. Due to the high bandwidth generated by raw multimedia bit-streams, complicated codecs must be used to compress the streams and transmit them over the Internet. Furthermore, multimedia streams are typically used collaboratively and so, stress the infrastructure needed to support the efficient software or hardware of multicasting required by the delivery of a given stream to multiple clients. Owing to the diversity of collaboration clients supported by Global-MMCS, the services for multimedia streams need to adapt the streams to different clients. We note that many relevant Web service specifications such as those for reliable messaging and notification appear to be inappropriately designed for scalable efficient multicasting as required by Global-MMCS. Thus, we suggest that multimedia collaboration is an excellent proving ground for general streaming data Grid infrastructure.

A media service or filter is a functional entity that can receive one or multiple media streams, perform some processing, and output one or multiple media streams. Each service is characterized by a set of input and output stream interfaces and a processing unit. According to the number of fan-in and fan-out filters, they can be divided into three categories: one-in-one-out filters, multiple-in-one out filters, and one-in-multiple-out. In addition, there is a final ‘sink’ filter category. We discuss each of these below.



One-in-one-out filters

Such a filter implements the basic transformation operation. For instance, a filter can receive as input a video stream in YUV4:1:1 format, resize it and deliver the modified video as output. Each filter provides a very basic adaptation on a stream in an intermediate format. Complex stream transformations can be built by combining several basic filters and creating a filtering workflow pipeline. Below are examples of one-in-one-out filters.

- *Decoder/encoder transcoder filters* aim at compressing/uncompressing the data into a chosen intermediate format (e.g. RGB24, YUV4:1:1, linear audio). Common codecs include H.261, H.263, MPEG1, MPEG2, MPEG4, H.264, and RealMedia. Transcoding generates a new stream, which is encoded in the format wanted by the user. For example, if a RealPlayer user needs to receive a video encoded in H.261 RTP, a RealStream producer is needed to first decode the H.261 video and generate a new RealFormat stream.
- *Image-scaling filters* resize video frames. This is useful for adapting a stream for devices with limited display capacities. They are sometimes required to enable transcoding operations, for example MPEG videos may be transmitted in any size while H.261 videos require predefined sizes such as CIF, QCIF or SQCIF.
- *Color-space-scaling filters* reduce the number of entries in the color space, for example from 24 to 12 bits, gray-scale or black-and-white.
- *Frame-rate filters* can reduce the frame rate in a video stream to meet low-end clients such as a PDA. For example, we can discard B-frame or P-frame in a MPEG-4 video stream with 24 fps to create a new stream with a lower frame rate.

Multiple-in-one-out filters

Mixer filters combine multiple streams. A video mixer can create a mixed video stream resulting from several input sources. Each element of the resulting mixed video (typically displayed as a Grid of images) results from an image-scaling adaptation of a particular stream. An audio mixer can create a mixed audio stream by summing up several input sources. Audio mixing is very important to those clients that cannot receive multiple RTP audio streams and mix them. A video-mixing service improves the visual collaboration especially for those limited clients, who can only handle a single video stream.

Multiplexors/Demultiplexors are used to aggregate/separate audio and video data in a multimedia stream. For instance, an MPEG multiplexor allows merging an MP3 audio and an MPEG-1 video in a MPEG2 stream. Multiplex and demultiplex are quite useful for guaranteeing stream synchronization in unpredictable network environments.

One-in-multiple-out filters

Duplicator filters are used to replicate an output media stream. Duplication is useful when a stream has different targets with different requirements. In most cases, multiple simple media filters should be organized in a media filter chain. Filters can be either as simple as bit-stream parsing, or as complicated as decoding and encoding. Composite media services are usually acyclic computation graphs consisting of multiple filter chains.



Sink filter services

There is also another type of bit-stream service, called a sink service, which does not change bits in the stream. Examples of sink services include buffering and replaying services. These can buffer real-time multimedia streams in memory caches or disk storage, and allow users to replay or fast-forward these streams through RTSP session. Sink filters can handle single or multiple streams. When multiple streams flow into a sink entity, all the streams can be synchronized and replayed. Based on such a composite sink service, an annotation service can be developed. Through annotation, users can attach text and image streams to the original video and audio stream to convey additional meaning in collaboration. Stream annotation is discussed in [12].

4.1. Global-MMCS architecture

Figure 2 shows our architecture for managing streaming services and their workflow. It is built around NaradaBrokering, which offers a powerful RTP event (message) delivery quite critical to multimedia streaming. We have developed XGSP [37] as the framework to specify stream schema and offer support for sessions, end-points, filters, replay collaboration, and their integration in streaming workflow. The stream schema uses a similar syntax to SMIL [38], describing the source (URI), the format, and QoS requirement of each stream. The function of filters can be defined by specifying the input and output stream streams for them. The whole workflow is a collection of filter chains and the available streams.

Media service and workflow

There is substantial literature on Grid and Service-based workflow (Grid workflow is summarized in [39], with extended papers to appear in [40], and the editorial can be found at [41]; see also [42]). Unlike many of these systems, Global-MMCS's streaming workflow, especially conferencing workflow, is implicit and can be determined by the system at run time based on the specified (in XGSP) sinks and sources and their QoS. For example, when a PDA with limited network and processing capability wants to receive an H.261 encoded, 24 fps, CIF video stream, a customized workflow is needed to transcode the H.261 stream to a JPEG picture stream or low-bitrate RealMedia Stream. An intelligent workflow engine can easily build a filter chain automatically based on the format description of the source stream and capability description of the PDA. Such an engine usually follows a graph search algorithm and tries to find a route from the graph node representing the format of the source stream to the destination node representing the format needed by the receiver.

No user involvement is needed for defining explicit workflow. Furthermore, in order to minimize the traffic and delay, most one-in-one-out filter chains should be constrained in a single-service container. One needs a distributed implementation to orchestrate multiple-in and multiple-out filters for different clients. Therefore the key issue in Global-MMCS media service management is how to locate the best service container, based on streaming QoS requirement, and making the service provider shared by participants in XGSP sessions.

Computation and storage resources connected with NaradaBrokering brokers are service containers that can host both media processing and session management services. The XGSP framework specifies

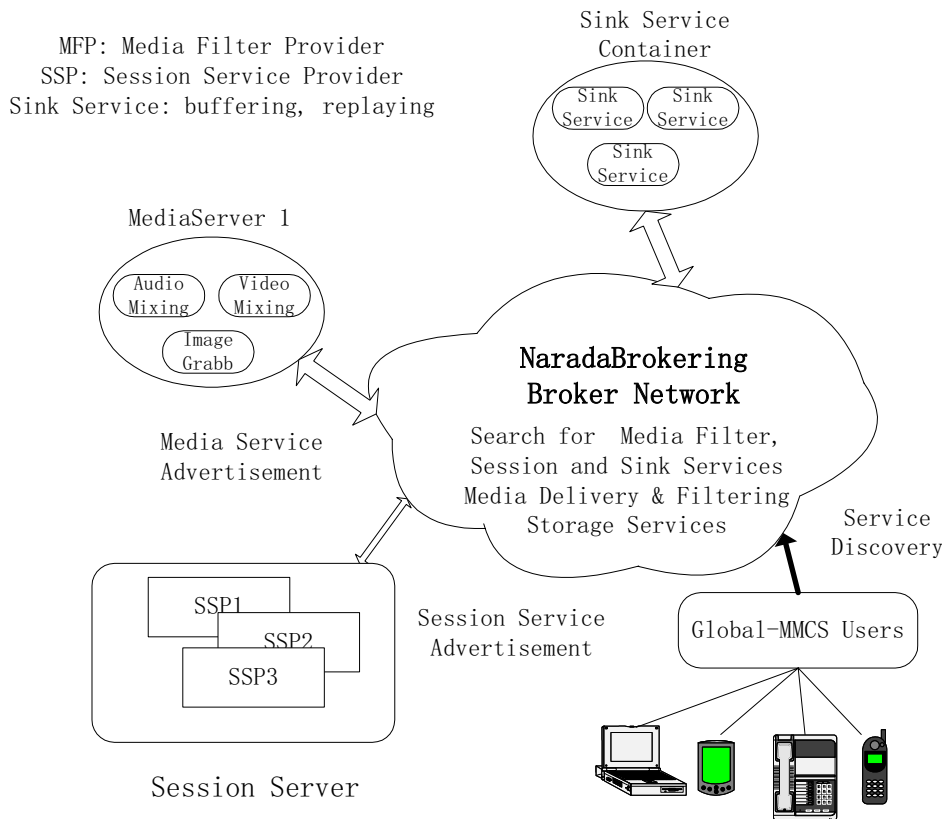


Figure 2. Global-MMCS streaming workflow management.

the XML scheme for describing the media-processing and session-management services. Each service provider can advertise its service XML description to distributed service registries such as we discuss in more detail in a companion paper [13].

Each broker may have a registered media service container called *MediaServer*, which hosts various computationally intensive media processing services. All service providers implement the interface to be able to run inside the service container. *MediaServer* can create, start, and stop media service instances. We are currently investigating adding support for UDDI to allow the *MediaServer* to advertise these service providers and report the status information to the distributed metadata registry regarding the load on that machine. XGSP audiovisual session servers can locate the best container and request a service instance to execute in the container.



4.2. Global-MMCS metadata management

We need to define collaborative sessions describing both the group of people and their clients as well as the associated media services. XGSP audiovisual sessions have five states: created, canceled, activated, deactivated, and finished. The XGSP AV session service manages these states. An XGSP user can initiate an audio-visual session while the session server can activate this created session at the meeting time after the needed service instances are created. The metadata or ‘context’ for the session has both static and dynamic parts.

Static metadata

Collaboration users need to know how many active sessions are available and their associated detailed information. The conference announcement can be implemented either in the XGSP session protocol or through the out-of-band communication. The XGSP framework divides the conference advertisement information into two levels: one is the collaborative conference calendar, which describes high-level metadata about the organization of the conference, including meeting time and topic. The other is the detailed information needed by audiovisual clients to join the conference, as, for example, the session identification in the system and transport addresses associated with the session. The high-level conference calendar is implemented as a Web service running in the XGSP Web server. Each active entry in the calendar has a link to the detailed session description.

Real-time metadata

In addition to the static metadata described above, a XGSP session has much real-time context. There are three important entities in a XGSP session: participants, streams, and services or filters. For each type of entity, a dynamic list has to be maintained. A participant list should keep the ID of each joined user and its multimedia capability and preference. A service or filter list should keep track of the activated services and their load. The stream list is more complex as it must keep track of source streams and filtered streams such as duplicated streams, transcoded streams, and mixed streams. For buffering and annotated streams, it also keeps the description of linkage in hybrid streams (see Section 1) and how they are stored. The latter is particularly critical for real-time replay [12].

For a real-time conferencing application, static and high-level metadata should be organized using standard calendar models and allows this conference calendar service to interact with other public, group, or private calendars. As WS-Context can manage session metadata between multiple participants in Web-Service interactions, real-time metadata of XGSP sessions may also be managed by WS-Context style service and implemented in an efficient manner [13].

4.3. Global-MMCS session and workflow management

NaradaBrokering can publish performance monitoring data in the form of XML on a topic that is subscribed to by the AV session server. From these performance data and broker network maps, the session server can estimate the delay and bandwidth between the service candidates and the requesting user. Based on the workload of the media service providers and estimated performance metrics, the session server can find the best service providers and initiate a media-service instance.



Furthermore, the AV session server has to monitor the health of each media-service instance. Through a specific NaradaBrokering topic, an active media-service instance can publish status metadata to notify the session server. If it fails to respond within a period of time, the AV session server is responsible to restart it or locate a new service provider and start a new instance. Note that the messaging infrastructure supports both TCP control and UDP media streams and their reliable delivery; the session can choose separate QoS for each type of stream.

Each session server may host limited numbers of active XGSP AV sessions. The exact number depends upon the workload and the computational power of the machine. The session initiator will firstly locate the right session provider to create a session service instance for a particular XGSP AV session. Then, this session server will locate the necessary media service resources on demand. In the current implementation, a default audio mixer is created to handle all of the audio in the session. Private audio mixers can be created on demand for private sessions supporting subgroups in the session. Furthermore, multiple video mixers can be created by the session server on the request of the client. An image grabber (thumbnail) service is created when a new video stream is detected in the session. Furthermore, customized transcoding services can be created when a user sends a request to access particular streams. For example, a mobile client such as a PDA connected to Wi-Fi, which only has limited processing power, wants to choose a 24 4-CIF MPEG-4 video, then a transcoding process pipeline consisting of frame rate adapter, video size down-sampler, and color transformation is needed to create this stream. Another example is an H.323 terminal, which can only handle H.261 and H.263 codecs, wants to display a MPEG-4 video: it will ask the session server to start a MPEG-4-to-H.261 transcoder.

Sink services such as buffering, archiving and replaying services can also be initiated by real-time XGSP sessions. Buffering and archiving services store events into distributed cache and file storage attached to NaradaBrokering overlay networks. Once stream data flow into these 'sinks', replaying service can pull the data flow out of the sinks and send to clients, based on the RTSP request of the user. The events are accessed in an ordered fashion and resynchronized using their timestamps, which have been unified using NaradaBrokers NTP service. The list with time-stamps of these archived and annotated streams is kept in the WS-Context dynamic metadata service. Through the recording manager service, a component of AV session server, users can choose streams to be buffered and archived. Through replay and RTSP services, users can initiate RTSP sessions and replay those buffered streams. After the streams are buffered, users can add annotations to the streams and archive the new composite streams for later replay.

5. SUPPORTING REAL TIME SENSOR GRID SERVICES

The basic services needed to support audio–video collaboration, such as reliable delivery, multicasting, and replay, can also be applied to problems in real-time delivery of sensor Grid data. Sensors are being deployed either individually or as part of sensor networks to collect fine-grain information about various entities. The trend in sensor technologies and related research shows us that in the near future we will see a growing need for new software architectures to integrate the sensor observations with data assimilation tools. Here, we describe such an architecture that is being developed based on Web services principles and summarize its application to real-time Global Positioning System (GPS) data. The architecture consists of several major components: GPS stations, distributed data processing units called filters, and publish/subscribe-based messaging infrastructure.

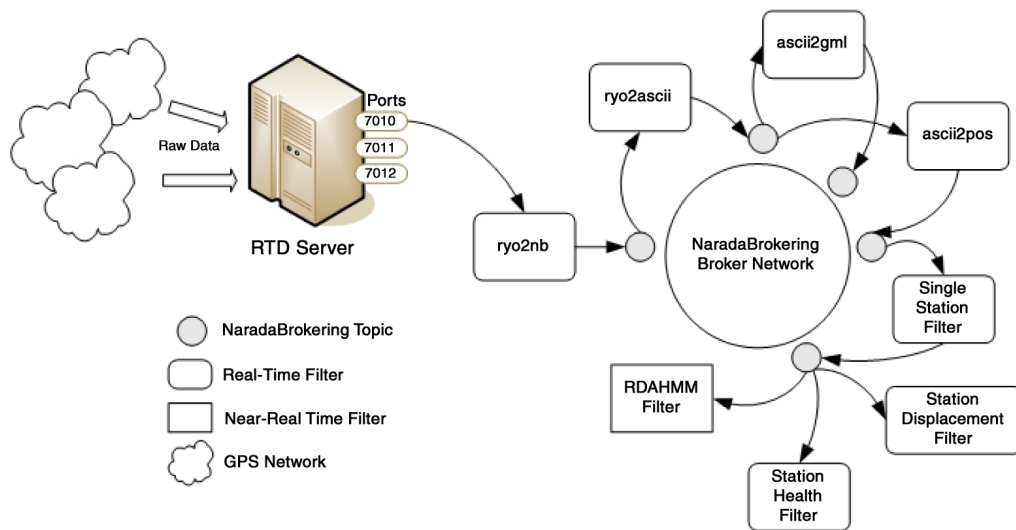


Figure 3. NaradaBrokering may be used to support filters of real-time GPS data.

In Figure 3, we depict our work to develop filters on live GPS data. The raw position messages are collected from stations deployed alongside the active fault lines in Southern California. Several of these GPS networks are maintained by the Scripps Orbit and Permanent Array Center (SOPAC) [43]. The GPS station observations are collected and processed by proxies called RTD servers. The stations broadcast their position information at the rate of 1–2 Hz, and the RTD server creates a single message for the whole network for that epoch. The position messages are made available to the clients via TCP ports in a binary format called RYO. We have developed several real-time filters to convert and republish these binary messages into different formats such as ASCII and Geography Markup Language (GML) encoded data.

The goal behind creating these filter chains is to make the real-time data available to scientific data analysis tools in a streaming fashion. We categorize the filters in our architecture into two categories; real-time and near-real time filters. The real-time filters process and republish the messages before the next message. A basic filter consists of three major parts: a NaradaBrokering subscriber unit to receive the sensor messages, a data processing unit, and a NaradaBrokering publisher unit to publish the processed message. The publisher unit is not used in some filters because not all filters need to republish the processed data. Figure 3 shows some of the filters we currently use to process real-time position messages. Other than format converter filters (ryo2ascii, ascii2gml, and ascii2pos) we have developed simple data-processing filters such as Single Station Filter, which is used to separate individual station positions from the original message that contains multiple stations and more complicated tools for calculating real-time displacements and the RDAHMM filter. RDAHMM is a time-series analysis application that is useful for identifying mode changes in the given dataset. The latest version of



RDAHMM can be trained for a particular station and used to analyze the real-time position messages of that station. Our RDAHMM filter accumulates a certain number of position messages and invokes the actual RDAHMM application to evaluate these data. The result of the application gives us different modes detected in the input data, which can be useful in identifying long-term deformation on the fault line.

6. FUTURE WORK

The NaradaBrokering system has been augmented recently with the ability to create topics [44] and discover and allow secure access to topics and the data published on it. We plan to leverage this capability to create secure access to data streams. Furthermore, implementing broker discovery [45] will allow us to select the nearest broker when a WSPProxy publishes/subscribes to data streams. Security for real-time streams is also, of course, a critical challenge but the architecture supports message-based security such as WS-Security [46,47], and we suggest that a form of WS-SecureConversation is natural for streams.

Conventional support of SOAP messages using the verbose ‘angle-bracket’ representation is too slow for many applications. Thus, we and others are researching [11,33,34] a systematic use of ‘fast XML and SOAP’ where services negotiate the use of efficient representations for SOAP messages. All messages rigorously support the service WSDL but transport the SOAP Infoset using the angle bracket form in the initial negotiation but an efficient representation where possible for streamed data.

Another interesting area is structuring the system so that it can be implemented either with standalone services, message brokers, and clients or in a peer-to-peer mode. These two implementations have tradeoffs between performance and flexibility and both are important. The core architecture ‘naturally’ works in both modes but the details are non-trivial and require substantial further research.

ACKNOWLEDGEMENTS

This work is supported by the U.S. Department of Energy’s Scientific Discovery through Advanced Computing (SciDAC) Program through a subcontract to award number DE-FC02-02ER2615 and the National Aeronautics and Space Administration’s Advanced Information Systems Technology Program, contract number 1253656.

REFERENCES

1. Qiu X-H. Message-based MVC architecture for distributed and desktop applications. *PhD Thesis*, Syracuse University, 2 March 2005. Available at: <http://grids.ucs.indiana.edu/ptliupages/publications/qiuPhDthesis.pdf> [12 July 2006].
2. Wu W, Bulut H, Uyar A, Fox G. Adapting H.323 terminals in a service-oriented collaboration system. *IEEE Internet Computing (Special ‘Internet Media’ issue)* 2005; 9(4):43–50. Available at: <http://grids.ucs.indiana.edu/ptliupages/publications/H323GW-IC0.pdf> [12 July 2006].
3. GlobalMMCS Collaboration Environment. <http://www.globalmmcs.org> [12 July 2006].
4. Fox G, Ho A, Pallickara S, Pierce M, Wu W. Grids for the GiG and real time simulations. *Proceedings of DS-RT 2005*, 2005; 129–138. Available at: <http://grids.ucs.indiana.edu/ptliupages/publications/gig/DSRTOct05.pdf> [12 July 2006].
5. Birman K, Hillman R, Pleisch S. Building network-centric military applications over service oriented architectures. *Proceedings of the SPIE Conference on Defense Transformation and Network-centric Systems*, Orlando, FL, 31 March 2005. Available at: http://www.cs.cornell.edu/projects/quicksilver/public_pdfs/GIGonWS_final.pdf [12 July 2006].
6. NCOIC Network Centric Operations Industry Consortium. <http://www.ncoic.org/> [12 July 2006].



7. W2COG World Wide Consortium for the Grid. <http://www.w2cog.org/> [12 July 2006].
8. Levitt B. NCOW RM Development Group update on target technical view—emerging net-centric standards—NCOW Reference Model v1.1. *Proceedings of the Open Group Conference*, San Francisco, CA, 25 January 2005. Available at: http://www.opengroup.org/gesforum/uploads/40/6574/NCOW_TTV_V1.1_Open_Group.ppt [12 July 2006].
9. NaradaBrokering Messaging System. <http://www.naradabrokering.org> [12 July 2006].
10. Fox G, Pallickara S, Pierce M, Gadgil H. Building messaging substrates for Web and Grid applications. *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences (Scientific Applications of Grid Computing Special Issue)* 2005; **363**(1833):1757–1773. Available at: <http://grids.ucs.indiana.edu/ptliupages/publications/RS-CGL-ColorOnlineSubmission-Dec2004.pdf> [12 July 2006].
11. Oh S, Bulut H, Uyar A, Wu W, Fox G. Optimized communication using the SOAP Infoset For mobile multimedia collaboration applications. *Proceedings of the International Symposium on Collaborative Technologies and Systems (CTS05)*, St Louis, MO, May 2005. Available at: <http://www.engr.udayton.edu/faculty/wsmari/cts05/SS1.htm> [12 July 2006].
12. Community GridsLab. Use of Grids in DoD applications. *Internal Presentation*, Indiana University, Bloomington, IN, 25 August 2005. Available at: <http://grids.ucs.indiana.edu/ptliupages/presentations/DoDGridsAug25-05.ppt> [12 July 2006].
13. Aktas MS, Fox GC, Pierce M. Information services for dynamically assembled semantic Grids. *Proceedings of the 1st International Conference on Semantics, Knowledge and Grid (SKG2005)*, Beijing, China, 27–29 November 2005. Available at: <http://grids.ucs.indiana.edu/ptliupages/publications/SKG2005-Aktas.pdf> [12 July 2006].
14. Pallickara S, Fox G. NaradaBrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer Grids. *Proceedings of Middleware 2003*, 2003; 41–61. Available at: <http://www.informatik.uni-trier.de/~ley/db/conf/middleware/middleware2003.html#PallickaraF03> [12 July 2006].
15. Pallickara S, Fox G. On the matching of events in distributed brokering systems. *Proceedings of ITCC 2004*, vol. 2, 2004; 68–76. Available at: <http://www.informatik.uni-trier.de/~ley/db/conf/itcc/itcc2004-2.html#PallickaraF04> [12 July 2006].
16. Happner M, Burrige R, Sharma R. *Java Message Service Specification*. Sun Microsystems: Santa Clara, CA, 2000. Available at: <http://java.sun.com/products/jms> [12 July 2006].
17. Uyar A, Fox G. Investigating the performance of audio/video service architecture I: Single broker. *Proceedings of the International Symposium on Collaborative Technologies and Systems (CTS05)*, St Louis, MO, May 2005. Available at: <http://www.engr.udayton.edu/faculty/wsmari/cts05/SS1.htm> [12 July 2006].
18. Bulut H, Pallickara S, Fox G. Implementing a NTP-based time service within a distributed brokering system. *Proceedings of the 2004 ACM International Conference on the Principles and Practice of Programming in Java*. ACM Press: New York, 2004; 126–134.
19. Microsoft, IBM and BEA. Web Services Eventing. <http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf> [12 July 2006].
20. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). <ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf> [12 July 2006].
21. Web Services Reliable Messaging TC WS-Reliability. <http://www.oasis-open.org/> [12 July 2006].
22. IBM, Globus, Akamai *et al.* Web Services Base Notification (WS-BaseNotification). <ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf> [12 July 2006].
23. HPSearch Project Web site. <http://www.hpsearch.org> [12 July 2006].
24. Web Service Management. <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-management1004.pdf> [12 July 2006].
25. Gadgil H, Choi J-Y, Engel B, Fox G, Ko S, Pallickara S, Pierce M. Management of data streams for a real time flood simulation. *Technical Report*, Community Grids Laboratory, Indiana University, Bloomington, IN, June 2004.
26. Pallickara S, Fox S. A scheme for reliable delivery of events in distributed middleware systems. *Proceedings of ICAC 2004*, 2004; 328–329. Available at: <http://www.informatik.uni-trier.de/~ley/db/conf/icac/icac2004.html#PallickaraF04> [12 July 2006].
27. Gadgil H, Fox G, Pallickara S, Pierce M, Granat R. 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, U.K., 2005. ACM Press: New York, 2005.
28. Uyar A. Scalable service oriented architecture for audio/video conferencing. *PhD Thesis*, May 2005.
29. Sun, Microsoft, Intel, *et al.* Web Service Management (WS-Management), June 2005. <https://wiseman.dev.java.net/specs/2005/06/management.pdf> [12 July 2006].
30. Microsoft *et al.* Web Service Transfer (WS-Transfer), September 2004. <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-transfer.pdf> [12 July 2006].
31. Microsoft, BEA, *et al.* Web Service Enumeration (WS-Enumeration), September 2004. <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-enumeration.pdf> [12 July 2006].
32. Microsoft, IBM & BEA. Web Services Eventing (WS-Eventing), August 2004. <http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf> [12 July 2006].



33. Chiu K, Govindaraju M, Bramley R. Investigating the limits of SOAP performance for scientific computing. *Proceedings of HPDC 2002*, 2002; 246–254. Available at: <http://www.informatik.uni-trier.de/~ley/db/conf/hpdc/hpdc2002.html#ChiuGB02> [12 July 2006].
34. Govindaraju M, Slominski A, Chiu K, Liu P, van Engelen R, Lewis MJ. Toward characterizing the performance of SOAP toolkits. *Proceedings of GRID 2004*, 2004; 365–372. Available at: <http://www.sigmod.org/dblp/db/conf/grid/grid2004.html#GovindarajuSCLEL04> [12 July 2006].
35. Head MR, Govindaraju M, Slominski A, Liu P, Abu-Ghazaleh N, van Engelen R, Chiu K, Lewis MJ. A benchmark suite for SOAP-based communication in Grid Web services. *Proceedings of Supercomputing 2005 (SC'05): International Conference for High Performance Computing, Networking, and Storage*, Seattle, WA, November 2005. Available at: <http://www.sigmod.org/dblp/db/conf/sc/sc2005.html#HeadGSLAECL05> [12 July 2006].
36. Gadgil H, Fox G, Pallickara S, Pierce M. Managing Grid messaging middleware. *Technical Report*, Community Grids Laboratory, Indiana University, Bloomington, IN, February 2006. Available at: <http://www.hpsearch.org/documents/ManagingGridMessagingMiddleware.pdf> [12 July 2006].
37. Fox G, Wu W, Uyar A, Bulut H, Pallickara S. Global multimedia collaboration system. *Concurrency and Computation: Practice and Experience* 2004; **16**(5):441–447.
38. SMIL. <http://www.w3.org/AudioVideo/> [12 July 2006].
39. GGF10 Berlin meeting. <http://www.extreme.indiana.edu/groc/ggf10-ww/> [12 July 2006].
40. Fox G, Gannon D (eds.). Workflow in Grid Systems Special Issue. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1009–1332. Available at: <http://www.cc-pe.net/iuhome/workflow2004index.html> [12 July 2006].
41. Workflow in Grid systems. <http://grids.ucs.indiana.edu/ptliupages/publications/Workflow-overview.pdf> [12 July 2006].
42. Yu J, Buyya R. A taxonomy of scientific workflow systems for Grid computing. *SIGMOD Record* 2005; **34**(3):44–49.
43. Scripps Orbit and Permanent Array Center (SOPAC). <http://sopac.ucsd.edu> [12 July 2006].
44. Pallickara S, Fox G, Gadgil H. On the creation and discovery of topics in distributed publish/subscribe systems. *Proceedings of the IEEE/ACM GRID 2005 Conference*, Seattle, WA, 2005. Available at: <http://cluster2005.org/> [12 July 2006].
45. Pallickara S, Gadgil H, Fox G. On the discovery of brokers in distributed messaging infrastructures. *Proceedings of the IEEE Cluster 2005 Conference*, Boston, MA, 2005. Available at: <http://cluster2005.org/> [12 July 2006].
46. Yan Y, Huang Y, Fox G, Pallickara S, Pierce ME, Kaplan A, Topcu AE. Implementing a prototype of the security framework for distributed brokering systems. *Proceedings of the Security and Management Conference*, 2003; 212–218. Available at: <http://www.informatik.uni-trier.de/~ley/db/conf/csreaSAM/csreaSAM2003-1.html#YanHFPPKT03> [12 July 2006].
47. Pallickara S *et al.* A security framework for distributed brokering systems. *Technical Report*, Community Grids Laboratory, Indiana University, Bloomington, IN, 2003. Available at: <http://www.naradabrokering.org> [12 July 2006].