Grids for Real Time Data Applications

Geoffrey C. Fox[†‡], Mehmet S. Aktas[†‡], Galip Aydin[†‡], Hasan Bulut[†‡], Harshawardhan Gadgil[†‡], Sangyoon Oh[†‡], Shrideep Pallickara[‡], Marlon E. Pierce[‡], Ahmet Sayar[†‡], and Gang Zhai[†‡]

† Computer Science Department, School of Informatics
‡ Community Grids Laboratory
Indiana University
Bloomington, IN 47405 USA
{gcf@grids.ucs, maktas@cs, gaydin@cs, hbulut@cs, hgadgil@cs, ohsangy@cs, spallick@grids.ucs, mpierce@cs, asayar@cs, gzhai@cs} .indiana.edu

Abstract. We describe our work in building support for streaming data services for Geographical Information System Grid services. We examine how streaming approaches may be used to increase data service performance for transporting XML messages. Similarly, streaming versions of traditional static map services may be combined with general audio/video session management capabilities to build collaborative, annotatable shared maps. Distributed services linked through messaging substrates require information and broker management capabilities, and we describe our research here. Finally, we discuss efficient XML representation techniques that can be used to increase performance of Web Services and support Web enabled devices.

1 Introduction

The implications of messaging and real-time data streaming in the core standards of Service Oriented Architectures [1] are just beginning to be investigated. SOAP intermediaries and distributed messaging systems may potentially greatly alter the nature of the Grid applications, creating an "Application Internet" on top of the core Internet.

As we have reviewed elsewhere [2], service oriented systems (i.e. Grids) are characterized by distinct, distributed services with well defined public interfaces. These services communicate through the exchange of messages. Both service definitions and message formats are expressible in XML using WSDL [3] and SOAP [4], respectively. Perhaps underappreciated so far is the importance of the message-based nature of service-oriented systems. Most applications have focused on the "remote procedure call" view of Web Services, and this approach has been successful in building file-system based scientific application Grids. However, remote procedure call implementations are only a convention. We may also exploit messaging approaches to handle streaming and real time data. One of the primary characteristics of this approach is the use of messaging substrates [5] to support the routing of the messages, as well as provide various qualities of service such as reliability and security.

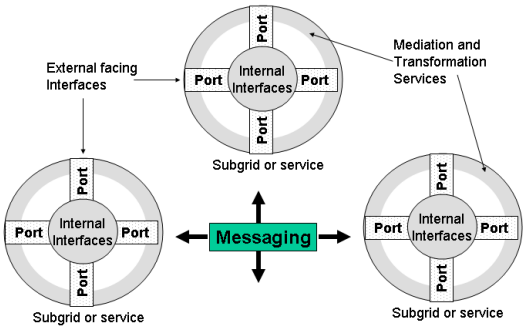

Fig. 1. Self-contained services, or collections of services, communicate through messages mediated by a messaging substrate. All communications (method invocation and responses, event notifications, data exchanges, and data streams) are messages.

We illustrate the basic concepts in Fig. 1. We may consider the individual services as distinct structures on a messaging foundation (Fig. 1 represents a top-down view in our analogy). The individual services expose public interfaces to the rest of the Grid services. Resource-specific internal interfaces and bridges (i.e. access to a particular database or simulation application) are not public: the service mediates access to these resources. All communication between services uses distinct messages that are managed by the messaging substrate that abstract network transport protocols (TCP/IP, UPD, etc.). Services publish or subscribe to messaging channels, constructing or consuming messages as appropriate, but are not otherwise responsible for message routine or message quality of service. Similar approaches have been used to manage inter-service communication of state changes and other notifications [6], but as we advocate in this paper, the message substrate approach should apply to all messages. The system may support several modes, including client-server and peer-to-peer (see 2.3)

Fig.1 is self-similar in that we may build Grids hierarchically, where each collection of services that constitutes a particular subgrid may itself expose a single external interface. Such systems are made possible through workflow and lightweight information systems that together can be used to define these subgrids. This work is described in more detail in Section 3.

In this paper, we describe our efforts to build service oriented, real time systems based on the NaradaBrokering substrate [7]. NaradaBrokering is a distributed, topic-based publish/subscribe system that may be used to route arbitrary message payloads. We begin with an examination of several applications: Geographical Information System (GIS)-based applications for accessing data archives, using video collaboration techniques for developing streaming map videos within collaborative sessions, and annotating video streams. We then

examine two fundamental services needed to support messaging Grids: information management using WS-Context, stream and broker management using HPSearch. We conclude with an examination of techniques for improving Web Service performance by using more efficient XML representations.

2 Geographical Information System (GIS) Data Service Applications

2.1 Improving Web Feature Service Performance:

The Open Geospatial Consortium (OGC) [8] standard specification Web Feature Service (WFS) [9] defines standard interfaces for web-based clients and servers to access geospatial feature data. WFS and other OGC based services use the Geography Markup Language (GML) [10] to encode geospatial data, and this provides a common language for both providers and consumers. The original WFS specification is based on HTTP Get/Post methods, but this type of service has several limitations such as the amount of the data that can be transported, the rate of the data transportation, and the difficulty of orchestrating multiple services for more complex tasks. Web Services help us overcome some of these problems by providing standard interfaces to the tools or applications we develop. We have developed a Web Service version of WFS and are testing in several scenarios where scientific data analysis tools such as Pattern Informatics [11] require fast access to large amount of data.

Our experience shows that although by using Web Services we can easily integrate several GIS and other services into complex tasks, providing high-rate transportation capabilities for large amounts of data remains a problem because the pure Web Services implementations rely on SOAP messages exchanged over HTTP. This conclusion has led us to an investigation of topic-based publish-subscribe messaging systems for exchanging SOAP messages and data payload between Web Services. We have used NaradaBrokering which provides several useful features besides streaming data transport such as reliable delivery, ability to choose alternate transport protocols, security and recovery from network failures.

Our streaming WFS uses standard SOAP messages for receiving queries from the clients; however, the query results are published (streamed) to a NaradaBrokering topic as they become available. Our initial implementation uses MySQL database for keeping geographic feature data, and we employ a capability in MySQL that streams the results row by row, allowing us to receive individual results and publish them to the messaging substrate instead of waiting for whole result set to be returned. The initial performance results show that (especially for smaller data sets) streaming removes a lot of overhead introduced by object initializations. Table 1 gives a comparison of the streaming and non-streaming versions of our WFS implementations. The data requested is the Southern California seismic records for the eventful year of 1992, initially obtained from Southern California Earthquake data center [12] and converted into GML for our Web

Feature Service. The first column is the minimum magnitude of the earthquake, the second column shows the data size of the query result. Timings for Streaming WFS contains two columns; the first column shows the time it takes to generate and stream out GML feature collection, the second column shows the total response time. The fourth column shows the total response time for non-streaming WFS. The difference between streaming and non-streaming WFS versions is that streaming version does not accumulate the query results and stream as soon as they become available. The timings are in milliseconds and include object initializations, query processing, database query and transport times.

Table 1. The performance of streaming and non-streaming versions of the Web Feature Service is compared. Timings are in milliseconds. Numbers in parentheses are standard deviations.

Event Magnitude Lower Bound	Data Size (KB)	Streaming WFS		Response Time Non-Streaming WFS
		Streaming the result	Total-Response Time	
3	880	2414	4570 (360)	5663 (31)
3.5	287	827	3405 (48)	4414 (39)
4	106	320	2945 (50)	4099 (71)
4.5	36	100	2661 (27)	3917 (38)
5	11	31	2425 (38)	3913 (77)

We can deduce from the table that for larger data sets when using streaming our gain is about 25%. But for the smaller data sets this gain becomes about 40% which is mainly because in the traditional Web Services the SOAP message has to be created, transported and decoded the same way for all message sizes which introduces significant overhead. We are investigating new methods for reducing the overhead in the streaming WFS to further improve the performance.

2.2 Collaborative Map Videos through the Streaming Web Map Services

The Web Map Service [13] is another HTTP GET/POST-based service that we may convert into a Web Service. Map servers are useful for building Web portal interfaces for geophysical Grid applications that integrate Web Feature Service-based data services with remote executables. Examples of this work may be found in [11]. We are also interested in going beyond static images to support the displays of time-dependent geographic data.

In our initial investigation of this problem, we generate the video image as a sequence of map images. After generating map images on the Web Map Server for each time slice for the same data layer, the WMS then converts the sequence into a video stream and publishes it to a Real Time Protocol (RTP) [14] session which

is represented as <IP Address, Port Number> pair. The supported video stream formats are H.261 and H.263, which are widely used formats in videoconferencing systems. A client capable of playing those formats can connect to the RTP session and play the stream published. Map video stream can be played in collaborative environments such as AccessGrid [15] and GlobalMMCS [16] sessions.

The map video stream has several configurable parameters which affect the quality of the produced map video stream: frame rate and video format of the stream, update rate of the map images in the video stream. We use the H.261 codec and update map images every 0.5 seconds while we keep the video frame rate at 10 frames per second. This provides sufficient quality for the video stream displayed at the receiving side. The reason frame rate and image update rate are different is that some clients might not be capable of visualizing video streams with low frame rate or can visualize them with very low quality. Keeping frame rate high will improve the quality of the video shown on the player while the map image rate is kept at a different rate.

Map video streams can be published to unicast or multicast RTP sessions. AccessGrid venues are multicast sessions. A video client listening a multicast session can receive and play the stream as long as the underlying network lets client receive multicast packets. GlobalMMCS also provides this map video stream to its clients as unicast video stream.

2.3 Collaboration Tools for e-Annotation

Streaming map servers may be viewed by any compatible client, and by integrating with GlobalMMCS, we may deliver streaming map video to a range of systems (including Access Grid, Polycom, and RealPlayer); see [16]. We may also build our own custom clients and services with extended capabilities for replay and annotation.

The e-Annotation collaborative tools facilitate interactive collaboration and distance education. The e-Annotation collaborative tools work in a peer-to-peer fashion atop of the collaboration architecture based on publish-subscribe messaging middleware. All the participating peers collaboratively work with each other using the NaradaBrokering messaging overlay network to annotate a live or archived video stream. The e-Annotation player is composed of the following components, illustrated in Figure 2.

Stream List Panel: There are three stream lists in this panel: a list of real time live streams, an archived video stream list, and a composite annotation stream list. These three stream lists are dynamically updated from the RTSP server by subscribing to the streaming control info topic from one of the brokers. Each stream list is a topic.

Real time live video play panel: This panel contains a video player which can play the selected real time live video stream that user selects in the real time live stream list. All the real time live video streams are published by GlobalMMCS through different NaradaBrokering topics.

Streaming player panel: This panel has a video player to play the buffered or archived video stream from the NaradaBrokering's buffers and storage nodes.

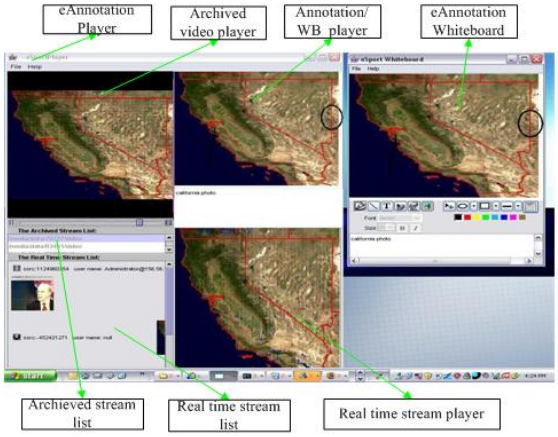

Fig. 2. The e-Annotation user interface allows video image display, capture, annotation, and playback. The image shown is LandSat imagery of the western United States, obtained from the NASA OnEarth project, http://onearth.jpl.nasa.gov/.

This player supports pausing, forwarding and rewinding the video streams with dynamic length (the buffered live video) or fixed length (the archived video stream). It also supports taking snapshots from a playing video stream. When taking a snapshot, the timestamp is associated with that snapshot. These snapshots are loaded to whiteboard to be annotated collaboratively and saved with the original stream as a new composite stream.

Video annotation player panel: This panel contains a player to play the new created composite annotation video stream. When the annotation stream is played back, the original video stream is played in the streaming player panel, and the annotated snapshots (which are streamed by RTSP server) are played in this panel, synchronized with the original video stream by the timestamps associated with them.

3 Web Intermediary and Service Management

3.1 Service and intermediary management

We are developing HPSearch [17] as script-based management console for controlling services and NaradaBrokering intermediaries. HPSearch provides dual functionality: 1) it provides a high-level language suitable for application developers to program workflows in a Grid that utilizes the messaging middleware, and 2) it provides tools to manage the messaging middleware.

HPSearch enacts a simple distributed services model by leveraging the capabilities of messaging middleware for routing data in streams between services. WSProxy is a specialized component of the HPSearch system that wraps an

existing service or a program while providing a Web Service interface for steering the service. It also provides an interface to the NaradaBrokering messaging system that transparently maps input / output streams to messages. Thus data is sent between services by publishing the event containing the data on a predetermined topic. Topics are automatically created by the HPSearch engine thus linking distributed services. WSProxy also presents a simple Web Service interface to help steer the service. The HPSearch runtime steers the service while the input / output is transparently managed by NaradaBrokering. NaradaBrokering has been recently augmented with topic creation and discovery [18] and message-level security. We plan on adding handlers in HPSearch to help automatically register secure streams and issue tokens to participating services. This will allow the application programmer to manage the security features of each stream that links distributed services.

HPSearch uses NaradaBrokering to route data streams between distributed services. Accessing a single broker over-and-over usually results in inefficient routing of data due to over loading. To assuage this problem we deploy a set of co-operating broker nodes that together form a virtual broker network. This broker network usually has multiple routes connecting peripheral brokers, thus providing alternate routes avoiding congestion at a single broker node. The distributed services connect to the peers at the periphery of this network. Thus, the brokering network can route data between peers much more efficiently, also providing different quality of service to each peer if required.

The scripting architecture also allows us to aggregate performance metrics in the system. These would allow us to determine congested paths and help decide alternate routes to create. The management architecture may be extended to manage remote brokers and provide means to try alternate configurations for creating links that span firewalls.

Management of such a set of multiple broker nodes and creating links between them poses a scalability issue. To address this issue, we are developing a specialized Web Service called the Broker Service Adapter. The Broker Service Adapter helps us deploy brokers on distributed nodes and setup links between them. Further broker nodes or the links between them may fail. HPSearch provides a scripting interface to instantiate new brokers at runtime and create links between brokers. The routing characteristics may be completely changed by tearing down an existing broker network and instantiating a completely new network.

Recently, management of systems has gained much research interest within the Web Service community. WS-Management [19] and WS-Distributed Management [20] are two competing Web Service specifications that propose management of remote resources based on the Web Service architecture. We plan on extending the Broker Service Adapter architecture to incorporate a simple WS-Management based interaction while fault-tolerance and scalability is automatically handled by the underlying Broker Service Adapter architecture.

3.2 Managing dynamic information with WS-Context

Collections of services such as Geographic Information System and collaboration services, such as described in Section 2, may be thought of as an actively collaborating set of grid/web services where services are put together for a particular goal. Each interaction with a set of services (including both workflows described in [11] and video collaboration sessions described in 2.3) can be modeled as a session. Each session is associated with a life time and maintains rapidly updated information known as context. Simply restated, this means that context plays an important role in enabling services to correlate their activities. We use the term "gaggle" for dynamically assembled grid/web services for a particular functional collection [21]. Gaggles may be gathered at any one time and can be considered as very small part of the whole grid.

We have designed and implemented information services [22] that support dynamically generated context to meet with aforementioned requirements of rich interacting systems. We have extended existing WS-Context Specifications [23] and provided with an implementation of XML metadata services supported by a MySQL database as backend storage. The WS-Context Metadata Service keeps track of context information shared between multiple participants in grid/web service interactions. It maintains user profiles, application specific metadata, information regarding sessions and state of entities in these sessions. In order to provide fault tolerance and scalability, we have also designed distributed metadata management architecture to support dynamically assembled Grid applications where metadata is widely-scattered and dynamically generated [24]. Additional applications of WS-Context services are described in Section 4.

4 High Performance XML Transfer

In a conventional Web Service environment, XML is the presentation format, which provides interoperability to the heterogeneous participating nodes. But in some constrained computing environments, such as mobile computing and real-time computing, processing verbose XML-based messages becomes a performance bottleneck. These performance overheads consist of parsing to retrieve information from its structured representation, more transmission time with increased document size over a narrow-bandwidth mobile connection and conversion overheads from in-memory representation to textual format.

The high-performance XML encoding is an open research area [25] [26] [27]. Related work on solving these problems can be categorized as individual message optimization or message stream optimization. An individual message approach produces a simplified, efficient, and self-contained message that has different representation of XML content – XML Infoset information. XBIS [28] falls on to this category. The message stream approach optimizes a whole sequence of messages – a session. Participating nodes negotiate the characteristics of the session and the message format in the session. Fast Web Services [29] and Handheld Flexible Representation [30] [31] (described below) fall on this category.

To achieve high-performance SOAP message processing and exchange in constrained environments, we are designing Handheld Flexible Representation (HHFR) and have implemented a prototype. We focus initially on handheld applications but we see important extensions to both Web enabled sensor devices and to data-centric Web Services in general. The architecture provides the preferred representation for applications (target services, client services, or message receivers) by separating SOAP message contents from the XML syntax of the message. The architecture targets a message stream and negotiates characteristics of the stream that includes the structure and types of SOAP message, reliability and security issues, and a preferred representation. To achieve the goal, the architecture design should address several issues.

Replacement of XML Syntax with Optimized Representation: The HHFR architecture and its framework supports message exchanging in the preferred representations, which is an optimized (or binary) format in most cases. The architecture separates SOAP message contents from the XML syntax. It is responsible to build the message in the preferred format (or representation) using internal *DataStructure* object and the separated contents stored in the HHFR data model. The internal *DataStructure* is created by parsing the HHFR schema document, which is a surrogate of separated structure and types of the SOAP message. We restrict the XML Schema definition for the HHFR Schema definition. These restrictions, such as a single schema document, no facets like minInclusive and maxInclusive, no references, make parsing a HHFR Schema document produce a single structure.

Focus on Message Streams: The HHFR works best for the Web Services, where two participating nodes exchange *a stream of messages*. For applications using a specific service, messages in the stream share the same data structure and data type for information items in it. Most of message headers are not changing in the stream session. Therefore, the structure and type of SOAP message contents in HHFR schema format and unchanging-SOAP headers can be transmitted only once, and the rest of the messages in the stream has only payloads.

Context-store as a Repository: In the HHFR architecture, a *context-store* module holds a static data of the message stream including the unchanging-SOAP headers, HHFR schema as a data representation, and other stream characteristics. These characteristics are captured by a *negotiation stage*. WS-Context (see 3.2) is well suited for this purpose. Information is defined with a URI. For example, we derived the HHFR scheme itself as *URI-S*. The current representation of the message in the stream is *URI-R* and the choice of transport protocol is *URI-T*.

Negotiation of Characteristics: To use the preferred representation and set up characteristics of the stream, the HHFR uses a conventional SOAP message in the beginning of the stream like the negotiation in the WS-SecureConversation [32] specification. The SOAP message, which has a *Negotiation* Schema defined with the HHFR Schema definition, is sent to request to start a HHFR session. It contains a HHFR Schema document of the SOAP message and any available quality of service issues.

To demonstrate the effectiveness of HHFR architecture, we have implemented a prototype mobile Web Service framework based on the HHFR architecture design. The prototype implements core design features of the design, such as the representation conversion, a fast transport option for a message streaming, the negotiation stage, and a simple HHFR data model. We choose Java as a language platform for both mobile and conventional computing and we limit the mobile node as the service client. Since the HHFR design doesn't cover the SOAP Engine feature, we utilize existing efforts – Apache AXIS and kSOAP.

We experiment benchmarks to compare the performance of the HHFR prototype with the performance of the conventional Web Service framework – AXIS. The details of the experiments and results can be found in [31] and we summarize them here. We develop two applications, a string array concatenation application and a floating number array addition application. Both applications use TCP transport for benchmarking. A string array concatenation service produces a single concatenated string of all string in a message. We measure a Round Trip Time (RTT) of the session, which includes multiple messages of given array length. The other application is a float number addition service that returns a summation of all float numbers of an array in a message. In this benchmark, RTT of the conventional SOAP application contains an OS level float-to-text conversion overhead, while RTT of the HHFR doesn't.

From the experiments, we observe a bigger performance savings on a longer session. The string concatenation benchmark and the float addition benchmark show that HHFR communication out-performs a conventional SOAP and the gap is fast-increasing as the number of messages in a session grows. These performance gaps are mainly caused by high latency of the cellular network, which uses the SprintPCS Vision service (speed up to 14.4kbps). The second observation we did is from the float number adding service and it is an efficient memory space usage of the HHFR prototype implementation by avoiding text conversions for building SOAP messages. During the benchmark, the runtime system of the prototype processes a larger size array in the message.

The high-performance XML can be achieved in many ways. For example, a binary data attachment in the SOAP message, such as MTOM [33] and XOP [34], is very popular solution to avoid a redundant encoding for already-encoded multimedia data or to preserve data integrity of the encrypted data. However the approach can be applied to only fixed data format and cannot cover user defined data structure such as the array. Another example is compressing SOAP documents. Compressing is reducing the size of the document for narrow bandwidth connected Web Service nodes. The XML specific XMill [35] can reduce the document size in half or more. But because of another layer of processing – a compression and a decompression, the compression approach is not saving overall performance overhead in many data domains.

5 Summary and Conclusion

We have described several research efforts to investigate problems in managing streaming data, with an emphasis on applications to earth science data and collaboration through our SERVOGrid project. As we have discussed, streaming approaches are capable of increasing Web Service performance for data-centric services such as the Web Feature Service. Similarly, static Web Map Services may be transformed into video streaming services that can be further integrated with shared collaboration tools for annotation and playback. Underlying these systems is the need to manage both the services and the brokers (or, messaging intermediaries) that constitute the system. We described our efforts here in developing HPSearch and WS-Context. Performance is one of the key problems facing Web Service applications. As we discussed, efficient and flexible XML representation is one possible solution. We have initially investigated this for hand-held collaboration devices but see obvious extensions web enabled sensor devices and to more conventional data services (particularly the Web Feature Service). Several open areas for additional research are described within Sections 2, 3, and 4.

This work is supported in part by a grant from the NASA Advanced Information Systems Technology program.

6 References

1. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. "Web Service Architecture." W3C Working Group Note, 11 February 2004. Available from http://www.w3c.org/TR/ws-arch
2. G. Fox, S. Pallickara and S. Parastatidis Towards Flexible Messaging for SOAP Based Services Proceedings of the IEEE/ACM Supercomputing Conference November 2004. Pittsburgh, PA.
3. M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework." W3C Recommendation 24 June 2003. Available from http://www.w3.org/TR/soap/.
4. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Service Descrption Language (WSDL) Version 2.0 Part 1: Core Language." W3C Working Draft 3 August 2005.
5. G. Fox, S. Pallickara, M. Pierce, H. Gadgil, Building Messaging Substrates for Web and Grid Applications to be published in special Issue on *Scientific Applications of Grid Computing* in Philosophical Transactions of the Royal Society of London 2005.
6. I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems." IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.
7. S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids.
   Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.

8. The Open Geospatial Consortium, Inc. see: http://www.opengeospatial.org/.
9. Vretanos, P (ed.) (2002), Web Feature Service Implementation Specification, OpenGIS project document: OGC 02-058, version 1.0.0.
10. Cox, S., Daisey, P., Lake, R., Portele, C., and Whiteside, A. (eds) (03), OGC Geography Markup Language (GML) Implementation Specification. OpenGIS project document reference number OGC 02-023r4, Version 3.0.
11. G. Aydin, M. S. Aktas, G. C. Fox, .H. Gadgil, M. Pierce, and A. Sayar SERVOGrid Complexity Computational Environments (CCE) Integrated Performance Analysis." Technical report June 2005 accepted as poster and short paper in Grid2005 Workshop, 2005.
12. Southern California Earthquake Data Center see: http://www.data.scec.org.
13. de La Beaujardière, J. editor, 2002. Web Map Service Implementation Specification, Version 1.1.1, OGC 01-068r3. http://www.opengis.org/techno/specs/01-068r3.pdf.
14. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force Request for Comments 3550 (2003). http://www.ietf.org/rfc/rfc3550.txt
15. Access Grid, http://www.accessgrid.org
16. W. Wu, G. Fox, H. Bulut, A. Uyar, and H. Altay "Design and Implementation of A Collaboration Web-services system", Journal of Neural, Parallel & Scientific Computations (NPSC), Vol. 12, 04. http://www.globalmmcs.org.
17. H. Gadgil, G. Fox, S. Pallickara, M. Pierce, and R. Granat, "A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications." In proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference, CCGrid 2005, Cardiff, UK
18. S. Pallickara, G. Fox, and H. Gadgil , "On the Discovery of Topics in Distributed Publish/Subscribe Systems." (To appear) Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing Grid 2005 Conference, Seattle, WA.
19. R. McCollum (ed), "Web Services for Management (WS-Management June 2005)." see: http://msdn.microsoft.com/library/en-us/dnglobspec/html/-ws-management.pdf.
20. W. Vambenepe (ed), "Web Service Distributed Management: Management Using Web Services (MUWS 1.0) Part 1." OASIS Standard, 9 March 2005. see: http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf
21. M. S. Aktas, G. Fox, M. Pierce, "Managing Dynamic Metadata as Context." In proceedings of Istanbul International Computational Science and Engineering Conference (ICCSE2005) June 2005, see: http://www.opengrids.org
22. B. Plale, P. Dinda, and G. Von Laszewski., Key Concepts and Services of a Grid Information Service. In Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002), 2002
23. Bunting, B., Chapman, M., Hurlery, O., Little M., Mischinkinky, J., Newcomer, E., Webber J., and Swenson, K., Web Services Context (WS-Context), see: http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf
24. M. S. Aktas, G. C. Fox, and M. Pierce, "An Architecture for Supporting Information in Dynamically Assembled Semantic Grids", 1st International Conference on Semantics, Knowledge and Grid, November 2005

25. O. Goldman, "XML Binary Characterization", W3C Working Group Note, Mar. 2005, http://www.w3.org/TR/xbc-characterization/
26. K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", Proc. Of 11^{th} IEEE Int. Symposium on High Performance Distributed Computing HPDC-11 2002, July 2002, pp. 256.
27. Fast Infoset, http://asn1.elibel.tm.fr/xml/finf.htm
28. XBIS XML Information Set Encoding, http://xbis.sourceforge.net/
29. P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E. Pelegri-Llopart, "Fast Web Services", Aug. 2003, http://java.sun.com/developer/technicalArticles/Web Services/FastWS/
30. S. Oh, H. Bulut, A. Uyar, W. Wu, and G. C. Fox, " Optimized Communication using the SOAP Infoset For Mobile Multimedia Collaboration Applications", Proc. Of the IEEE 2005 International Symposium on Collaborative Technologies and Systems (CTS 2005), St. Louis, Missouri, USA, May. 2005.
31. S. Oh and G. C. Fox, "HHFR: A new architecture for Mobile Web Services Principles and Implementations", Comm. Grids Technical Paper, Sep. 2005.
32. S. Anderson, at al, "Web Services Secure Conversation Language (WS-SecureConversation), May. 2004, ftp://www6.software.ibm.com/software/-developer/library/ws-secureconversation052004.pdf
33. M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan, "SOAP Message Transmission Optimization Mechanism", W3C Proposed Recommendation, Nov. 2004, http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/
34. M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan, "XML-binary Optimized Packaging", W3C Recommendation, Jan. 2005, http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/
35. H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data", Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, May. 2000.