Title
Collaborative Web services and the W3C Document Object Model

Authors
Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox

Postal Address
Community Grids Lab, Indiana University
501 N. Morton St, Suite 222
Bloomington, IN 47404-3730

Xiaohong Qiu
Email: xiqiu@syr.edu
Phone: 812-8560754
Fax: 812-8567972

Bryan Carpenter
Grids Computing Lab, Indiana University
501 N. Morton St, Suite 222
Bloomington, IN 47404-3730
Email: dbcarpen@indiana.edu
Phone: 812-8560762
Fax: 812-8567972

Geoffrey C. Fox
Grids Computing Lab, Indiana University
501 N. Morton St, Suite 222
Bloomington, IN 47404-3730
Email: gcf@indiana.edu
Phone: 812-8567977
Fax: 812-8567972

Abstract

The Internet makes it possible to share information (e.g. text, image, audio, video and other formats of data) across the globe. The W3C DOM sets up document objects representation standard of information. We are attempted to build more powerful collaborative distributed systems over the Internet that provide high-quality, interoperable and transportable Web Services. Can we meet our goal with the help of XML technology and an event-driven "message passing MVC" model? This article shows our approach and answer.

1 Introduction

Grids, peer-to-peer networking or more generally Internet systems (or Internet computing) are developing both new technologies and new approaches to large scale applications. These efforts are developing pervasive shared resources and capabilities managed to support dynamic or structured virtual organizations [1,2]. There are several key projects such as TeraGrid [3], the UK e-science program [4] and technologies such as Globus [5], Gnutella [6,7], JXTA [8] and JINI [9]. In the Community Grids laboratory at Indiana University, we have proposed peer-to-peer Grids [10,11] integrating many of these ideas, and developed some prototype technology components as well as some full systems. We have emphasized the special requirements of real-time collaboration [12,13], as needed in distance education [14] and support of distributed research collaboratories [15]. In this paper we focus on one part of this program – how should applications be built in peer-to-peer Grids so that they can be easily integrated and take advantage of other services. The answer to this is simple/clear – namely applications are just (software) objects and in today's Grids and P2P networks, distributed objects are built as Web services [16]. We have explained in an earlier paper [11] how one can in fact make general Web Services collaborative by sharing either the input or output resource-facing and user-facing ports (figure 1). We have also introduced two useful technology components to support this.

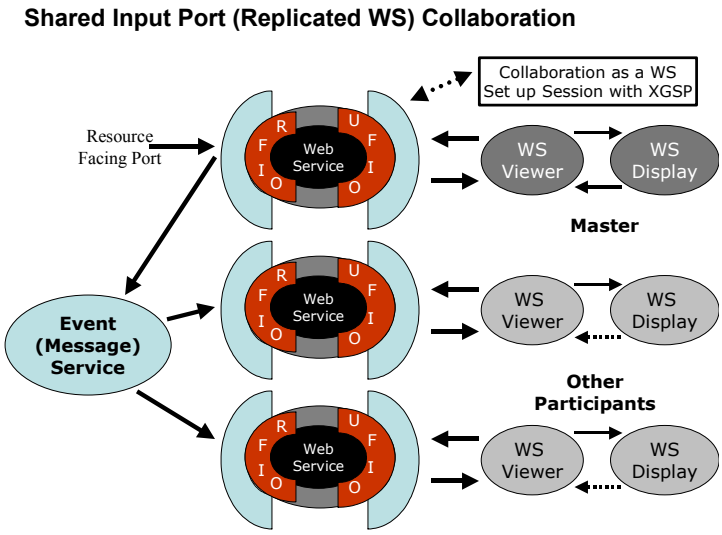

Figure 1 *One way of setting up Collaborative Web services involving replicating the application and using the system event service (in our case NaradaBrokering) to share state-defining messages. We show user-facing (UFIO) and Resource facing (RFIO) Web service ports*

1) A Web Service that supports collaboration by providing the Web service equivalent of H323, SIP and JXTA functions – these include establishing sessions, clients, profiles and a collection of shared resources. There is a new XML protocol XGSP introduced to capture the messaging needed to implement this which we term "Collaboration as a Web Service". [17,18]
2) An event and messaging infrastructure NaradaBrokering [18, 19] that can manage the unicast and multicast delivery of messages between the different clients. NaradaBrokering copes with multiple protocols (both TCP/IP and UDP based) and tunnels through firewalls and network bottlenecks determined by a performance module.

Critical to the concept of collaborative web services illustrated in figure 1, is that Web services are built around messaging – their state is determined by control messages from the user or other services and their "meaning" (in particular their output display) is defined by messages sent from other Web services. This idea has prompted the development of WSRP (Web Services for Remote Portals) to specify the form of user-facing ports [20].

Although any application is "just an object", it is not like a distributed object or Web Service with message-based input and output. Rather one has integrated software that bundles user interface, the "core of application" and system interactions (to files and other programs) in a single package. Microsoft Word , used to prepare this paper, is such a classic or legacy application. In fact there are the equivalents of the Web services messages "hidden" inside the application where the message might appear as a method call with the message placed on the program stack. Recognizing this, a variant of WSRP, WSIA (Web Services for Interactive Applications) has been proposed for such cases [20].

Here we wish to investigate an approach that essentially builds all applications as Web services and correspondingly

1) Defines all system interactions with messaging on resource facing ports
2) Separates the application into a "user interface" portion and a functional "core"
3) Converts all user interaction (such as mouse and keyboard actions) in the "user interface" to messages sent for interpretation at the Web service

We suggest applying this design principle systematically will lead to many advantages including easier support of universal access [21], easy deployment on server-controlled network computers, and the natural support of collaboration. We are investigating this idea both in applications like Word but this is not trivial because the object model defining such applications is not freely available. So here we choose to look at an application – the Java SVG (Scalable Vector Graphics) application [22] – whose full source is available from Apache [23]. This application also has the important feature that it faithfully supports the W3C document object model DOM [24-26] which essentially defines all needed SVG state in terms of their event model. Further we can expect browsers, word processors and presentation programs to eventually adopt such an object model. Thus we believe our study will indicate how any W3C DOM based application can be built in the Web service fashion. In other publications we have explored the universal access implications of this idea by using it to support collaborative SVG between desktop and PDA devices [27]. The work illustrates that the user interface can have many different realizations – it could just be a viewer of a bitmap image as in other PDA work or can be the display of a vector graphics standard. There we did not explore the W3C DOM rich event model, which is the focus here.

In the following sections, we introduce the key concepts: Web services, W3C DOM, SVG and the well established MVC (Model View Controller) approach which is closely related to the Web service design pattern. We then describe our design of an "event-driven message passing" collaborative SVG viewer system, analysis of the different event types and current results. These are built on the NaradaBrokering and XGSP infrastructure already developed and tested in conventional web service case. Finally we present some conclusions.

2 Background

2.1 Web Services

With the using of Internet broadened and in depth, the web has already become the communication infrastructure not only for people to people but also for application to application. The programmatic interfaces made available are referred to as Web services [16]. They are aimed at using XML to build distributed information processing systems that work across the Internet. W3C Web Services Description Language (WSDL) [28] provides a model and an XML format for describing Web services (e.g. abstract functionality and context information for their executions). Legacy applications that implement Web Services interface are made services that are accessible through the web.

2.2 W3C DOM

The World Wide Consortium (W3C) Document Object Model (DOM) [24,25] is a platform- and language-neutral interface that defines how browsers and other software represent documents as objects to dynamically access and modify a document's contents, structure and style. In fact, this specification defines a standard set of fundamental interfaces (as in DOM Core Level 1 specification) that are capable of representing any structured document model, not just a programmatic interface for XML and HTML. In the broadest possible sense, any information can be represented by Document objects – text, graphics, audio and video, hyperlink and other formats of data.

While a typical DOM structure model is a hierarchical tree, an entire HTML or XML document can be represented as a Document object. Conceptually, the document interface is the root of the document tree with Node objects and provides implementations to insert, remove and alter the nodes in the tree.

Furthermore, DOM Level 2 Event Model [26] defines a generic event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event. There are three types of events in the DOM; firstly *UI events* (user interface events that are generated by user interaction through an external device such as mouse and keyboard); secondly *UI logical events* (device independent user interface events such as focus change messages or element triggering notifications); and finally *Mutation events* (events caused by any action which modifies the structure of the document). In this paper, we combine the latter two types of DOM events as so called *Semantic events*. The latter also include other high-level actions defining state of system but not recording directly a user action. Semantic events are usually generated by UI events and we are careful not to double count when we replay events (the basis of collaboration).

Since most of DOM APIs are interfaces rather than classes, vendors can implement them as a thin veneer on top of their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs. This increases interoperability on the Web. The W3C DOM is the standard for modern browsers and expected to be the future of web development.

2.3 W3C SVG

Scalable Vector Graphics (SVG) is a new XML-based language for describing two-dimensional vector and mixed vector/raster graphics from the W3C [22]. Compared with raster graphics (pixel images such as JPEG, GIF, PNG and BMP), developing web pages or application GUI with SVG has the merit of being scaleable without loss of resolution.

Another great attraction to graphical Web developers is that SVG has the features of openness, interoperability and transportability. The information about an image is stored as plain text with XML rather than in a proprietary binary format like those of Adobe Postscript and Macromedia Flash files from graphic design systems. Sophisticated applications of SVG are possible by use of a supplemental scripting language (e.g. JavaScript) which manipulates the SVG Document Object Model (DOM). SVG provides complete access to all graphic objects (vector graphic shapes, images and text), attributes and properties interactively and dynamically. A rich set of event handlers such as *onmouseover* and *onclick* can be assigned to any SVG graphical object.

We may think of SVG as a particular XML-based application of the W3C DOM. Using SVG as client graphical User Interface for browsers and more general distributed applications is very promising ─ it makes it easier to develop GUIs that present documents in a unified layout over a variety of displaying devices (from PC monitor, PDA screen to Printer, etc.).

2.4 MVC Techniques

The well-known Model-View-Controller (MVC) framework [29] is the central concept behind the Smalltalk-80 user interface. MVC applications are split into several triads each of which comprises a relationship between a Model object, a View object and a Controller. The *view* manages the graphical and/or textual display. The *controller* interprets the mouse and keyboard GUI events, commanding the model and/or the view to change accordingly. The *model* implements core functions of the application (the state and behavior of application domain), responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). The picture below illustrates the basic Model-View-Controller relationship.

The MVC model has been the basis for most widely used graphical environments nowadays [30]. Currently this approach is typically implemented as an event-driven MVC model, where the *controller* becomes an

event handler that dispatches mouse events, keyboard events, and other system events, to the corresponding processing functions in the *model*. Microsoft Windows [31] and Java Swing UI components [32] are examples of event-driven MVC architecture.

3 Collaborative DOM Applications

3.1 Message passing MVC

In our collaborative DOM application architecture, we use the fundamental concepts of the MVC framework but adapt it to the more distributed event-driven "message passing MVC" design (where an event-driven message passing system takes the role of *Controller*) suggested by Web Services. Each client or collaborative component (a classic application like Word for example) is split into the *view* -- a basic user interface and presentation layer which is mainly concerned with accepting DOM UIEvents (mouse and keyboard events) and a functional *model* layer. The latter implements computation and modification of the DOM objects, and may also include graphical preparations for rendering of a DOM object although this part can be moved to the rendering part of the view. As the view and model are designed to be separate components running on different machines, the details of their hosts and connection network determine how much functionality one puts into each. The *Controller* becomes an internal event-handler within a stand alone client and is an event-driven message passing layer externally between collaborative clients. Namely, primitive event information (UIEvents) is packed into events at the *view* user interface layer, passed on as messages and interpreted at the *Model*. Finally the user interface will get a feedback response from the *Model*.

The message passing mechanism seamlessly glues the architecture of a client (internally) and that of a collaborative component (externally) together in a unified interface ─ message. It makes the framework more distributed, dynamic and powerful, in addition of the merits (e.g. scalability, maintainability and reusability) that are gained from MVC.

3.2 Event Structure

Events play an important role in a collaborative DOM system as they contain all the information of collaboration. Specifically, events bridge the mapping from user interactions to corresponding core functions of respond. We have the following ways of marking events types:

1) We classify DOM events into two categories – UIEvents and semantic events.

The former comes from user input ─ mainly mouse and keyboard events; the latter higher level events are usually generated from UIEvents and represent functionality of the application or service. They includes UI Logic Events and Mutation Events of DOM. Examples of semantic events in a SVG viewer application are "Open a SVG document", "Open An New Window", "Open A Hyper Link", "Zoom in", "Zoom out" and "Rotation" in a SVG viewer.

2) Master events vs. non-master events

In our collaborative session, all participating clients subscribe to an event topic through NaradaBrokering system. Among them, only one client withholding "master" token generates master events that trigger collaborative behaviors in the communication group. Therefore, events come from other participating clients are non-master events.

3) Major events vs. minor events

To build a robust system, we have to take into consideration that the following scenarios will occur in the real world: clients join and leave a collaborative session asynchronously; a client system crashes and reboots; a service of replay (recording of the collaborative session so far) is requested, and so forth. For the purpose of synchronization and replay functions, we design a mechanism that marks the synchronization point with major events. Major events are selected semantic DOM events (such as load a SVG file and open a new window) which fully specify the application state. Minor events are events like "mouse move" specifying "small" system changes.

Collaboration involves sharing state between collaborating applications and we define state in terms of a stream of time-stamped change (minor) events applied to a given initial state which is a major event. We commit this sequence of changes "every now and then" to form new major events that fully specify the application but keep both major events and the minor events that led up to them. A change (minor) event based application specification is most powerful as one can dynamically choose which events to accept and which events to discard; further each collaborative client can inject their own events. A state (major) event is the most efficient way of specifying the instantaneous state of an application. By keeping both major and minor events we can trade off performance and flexibility. Note both the full state and change specifications are thought of as "just events".

4) Collaboration as a Web Service (XGSP) Events
All information in our approach is carried by events transported by NaradaBrokering. The nature of the collaboration (e.g. who is in the session and what applications are shared?) is specified by XGSP [11] and generated by the Collaboration Web Service. This service initiates collaborative applications such as SVG discussed here and for example generates the "master token". Thus the Controller event handler must process both events specialized to the application and such overall control events.

5) Structure of Events
An event contains information such as follows:

- An original UIEvent or selected semantic events of DOM
- Event types (e.g. master/non-master, major/minor)
- Context information of the collaboration (e.g. client ID, session/topic, windows name in a multi-SVG viewer application, event sequence number)
- Context information of the Web services specifying application and collaboration session.

3.3 Interaction between DOM and Web Service

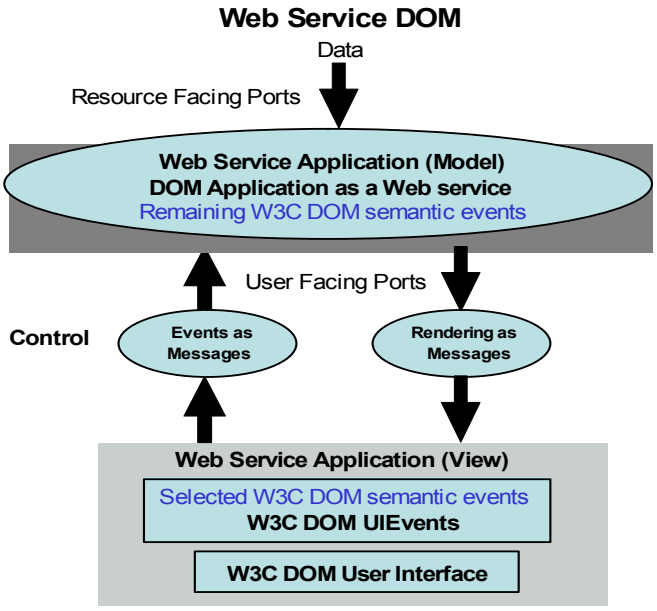

Figure 2 DOM Application as a Web Service

Figure 2 outlines the basic idea of a DOM application as a Web Service. We split the application into user interface layer and functional core layer. The separation may look simple, but the event-driven message passing layer (*controller*) makes the system distributed and very powerful. The *model* runs on a server as a Web Service and the *view* runs as a client interfacing with the user. The *controller* can be distributed between these components to suit capabilities of host machines and network. For a thin layer client system, as with network computers or small devices, one may move nearly all the computation to the back-end server and leave very simple interactive functions to the user interface. In a Web Service DOM application, the communication between client interface and Web Service server is elegantly done by passing as messages both the state-defining events and the rendering information defined in the Web service. This leaves just some relatively simple tasks, such as interpretation of UIevents and messages, implemented at the client side, together with the final rendering. Semantic events can be implemented either on the client or the backend Web service depending on convenience and performance issues.

We have *Resource Facing Ports* and *User Facing Ports* as the interfaces between the Web service and resources, and Web service and user interface, respectively. The former ports supply the information needed to define the content of the Web service; the latter carry control information from user or rendering information from *Model*. We exploit the fact that the state and results of a Web service is entirely defined by messages on these ports. Control information is included in two classes of messages ─ *events* (UIEvents and/or semantic events) that trigger changes of *Model* (e.g. DOM document modification) flow from the client user interface to Web Service server; *rendering* messages carrying display changes (e.g. buffered bitmap image) flow in the reverse direction.

3.4 Collaborative DOM

As we may think of any functional unit as an object of certain type (no matter whether it is a segment of text, a graph, a file, a software, a supercomputer, an endpoint of Grids, a peer in the networking or a Web service), DOM provides a standardized interface of universal access to those objects. Therefore, the approach of building collaborative DOM is a meaningful exploration and a building block of our architecture for general Internet systems (or Internet computing.)

Since SVG is essentially an application of DOM, as an experiment in building collaborative DOM, we have designed and implemented a collaborative SVG viewer system. A SVG viewer is just like an ordinary web browser except that all the contents are rich graphical contents from SVG files. Interactions like zoom in and zoom out will bring an enlarged or shrunken portion of the contents in appropriate ratio while keeping the same resolution. In a collaborative SVG viewer system multiple clients that join a collaboration session can view rendered graphics, texts and client interaction feedbacks in a synchronized fashion.

Architecturally, there are many styles and approaches to implement collaboration [10,11]. However, the key to a collaboration system is "sharing". To implement a collaborative SVG viewer, we need to map or convert a classic SVG viewer application to a MVC model and make it an event-driven message passing system (figure 2). Moreover, we need to define collaborative events that have sufficient information describing synchronized behavior among clients (subscribers in the same session). Note that these collaborative events are shared and the mechanism of control is implemented by passing events and rendering messages through *User Facing Ports.* The key structure of collaborative SVG (or DOM) is illustrated in figure 3. The lower part (with a classic application) shows how to abstract events of a classic application and make it collaborative. The right part of the graph displays a Web service application (with service functions and view separated) that are synchronizes with the master application through event-driven messages in the collaborative system.

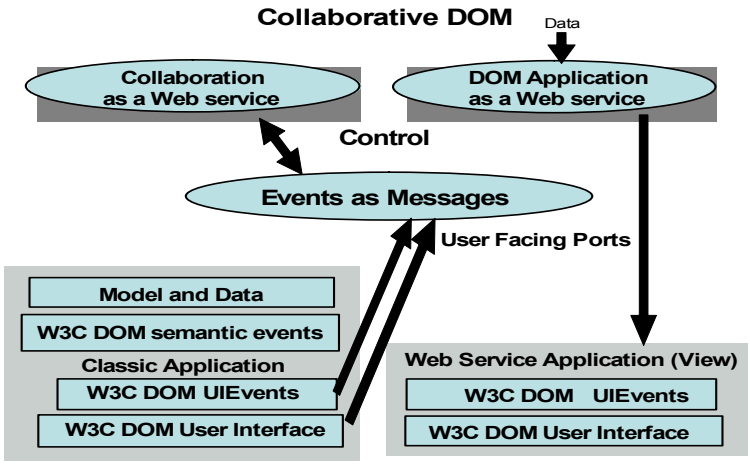

Figure 3 Collaborative DOM as a Web Service

We define a collaborative event as an object that wraps original SVG events (as in the classic application) with additional context information for collaboration and Web services. The context information helps to guide the events passage through the NaradaBrokering system [19] (events and messaging infrastructure) to reach other clients (subscribers in the same session). The receivers un-wrap the collaborative event and get an SVG event that defines detailed actions on SVG DOM. The *Model* part of Web service application analyses the SVG event based on its type and then execute actions as appropriate.

3.5 Collaborative SVG implementation

As in typical collaboration systems [11,12,13,17] , we assume there is a single "master" client that "controls" the Web service, and all other participating clients update their states according to messages reflecting the view from the "master" client. In the so-called shared event collaboration model (equivalent to Shared Input Port model of figure 1), we have control information flowing from the "master" client to the NaradaBrokering system (which manages event queues that support the publish/subscribe model) and then reach "participating" clients – the other members of a collaboration session. As show in figure 4, the "master" client creates collaborative events ─ one copy flows within its own system and another copy is sent out to other clients in collaboration. Although one can have much more complicated scenarios with interchangeable control, only one "master" is typically instantaneously chosen among the clients. The participating clients only respond to the event message coming from the "master" client and implement the same behavior as the "master". Thus, all clients are synchronized in a collaborative session.

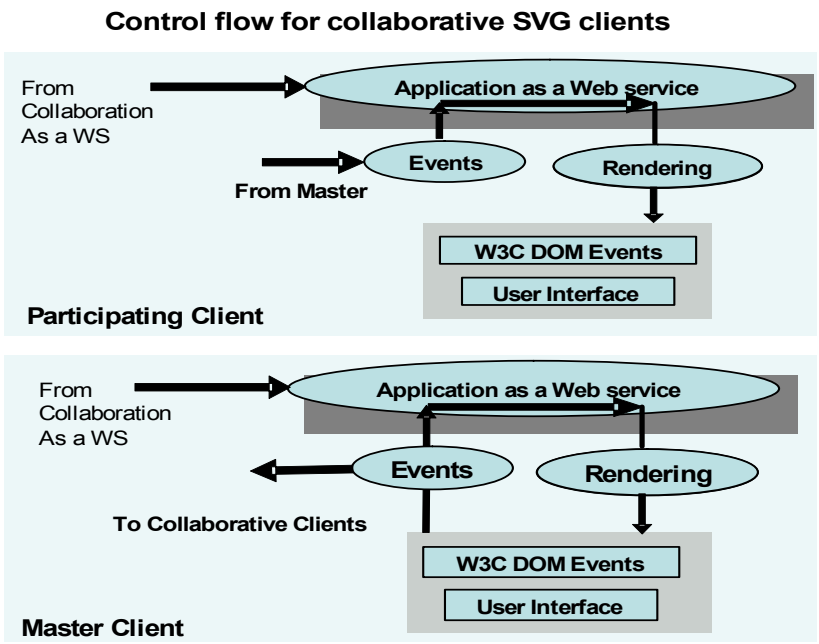

Figure 4 Control flow for collaborative SVG clients

While there is only one SVG viewer application running at each client side in a collaborative session, each SVG viewer may load multiple SVG files with each in a separate window. This is in fact implemented by a top-level instance of the application window as the root window. It pawns multiple threads that generate multiple sub-windows. The root window usually saves context information (e.g. user interface CSS and font settings) for windows lower in the hierarchy. To make sure that an event will be invoked in the correct window, we design a CollaborativeEventsProcessor object that loops through an internal event queue (of collaborative event message objects) and handles incoming event messages ─ identifying and dispatching messages through the EventsRepository (an internal event queue of collaborative event message objects) and handles incoming event messages ─ identifying and dispatching messages through the program to the appropriate window.

4. Conclusion

As discussed in earlier sections, we have designed and prototyped an approach to building DOM applications as a Web service and then making them collaborative. We have reached the following conclusions from the work reported in this paper ─

1) To share "legacy applications" like Microsoft Word and make them as shared Web services, it is important to convert the systems to an event-driven "message passing MVC" model (figure 2). Namely

separating preliminary user interface interactions from core computation or processing functions while using message passing For this strategy, systems with object-oriented design are particularly suitable as illustrated by the Java Batik SVG

2) A collaborative event object should be well defined and contains sufficient information for collaboration (including context information of collaboration, Web service and original SVG events information). It reflects the essence of control in an event-driven message passing model.
3) With the successful experience of building collaborative SVG DOM, we build up confidence for continuing the approach of building other applications as Web services and using a similar collaboration strategy. OpenOffice and Microsoft Office are natural applications to consider next.

References

1) *Grid Computing: Making the Global Infrastructure a Reality* edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, Chichester, England, ISBN 0-470-85319-0, February 2003
2) The Grid Forum http://www.gridforum.org
3) TeraGrid Project http://www.teragrid.org/
4) United Kingdom e-Science Activity http://www.escience-grid.org.uk/
5) Globus Grid Project http://www.globus.org
6) *Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology*, edited by Andy Oram, O'Reilly Press March 2001.
7) Gnutella P2P System. http://gnutella.wego.com
8) Sun Microsystems JXTA Peer to Peer technology. http://www.jxta.org.
9) Sun Microsystems Jini Java service technology http://www.sun.com/jini.
10) Geoffrey Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Xi Rao, Shrideep Pallickara, Quinlin Pei, Marlon Pierce, Ahmet Uyar, Wenjun Wu, Choonhan Youn, Dennis Gannon, and Aleksander Slominski, "An Architecture for e-Science and its Implications" in *Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunications Systems*, edited by Mohammed S.Obaidat, Franco Davoli, Ibrahim Onyuksel and Raffaele Bolla, Society for Modeling and Simulation International, pp 14-24 (2002). http://grids.ucs.indiana.edu/ptliupages/publications/spectsescience.pdf
11) Geoffrey Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, Minjun Wang, Wenjun Wu *Collaborative Web Services and Peer-to-Peer Grids* presented at 2003 Collaborative Technologies Symposium Orlando January 20 2003 http://grids.ucs.indiana.edu/ptliupages/publications/foxwmc03keynote.pdf
12) WebEx Collaboration Environment. http://www.webex.com
13) Placeware Collaboration Environment. http://www.placeware.com
14) Collection of Resources on distance education by Community Grids Laboratory http://grids.ucs.indiana.edu/ptliupages/publications/disted/.
15) Geoffrey Fox, Sung-Hoon Ko, Marlon Pierce, Ozgur Balsoy, Jake Kim, Sangmi Lee, Kangseok Kim, Sangyoon Oh, Xi Rao, Mustafa Varank, Hasan Bulut, Gurhan Gunduz, Xiaohong Qiu, Shrideep Pallickara, Ahmet Uyar, Choonhan Youn, *Grid Services for Earthquake Science*, Concurrency and Computation: Practice and Experience in ACES Special Issue, 14, 371-393, 2002. http://aspen.ucs.indiana.edu/gemmauisummer2001/resources/gemandit7.doc
16) W3C Web Services at http://www.w3.org/2002/ws/.
17) Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut "A Web Services Framework for Collaboration and Audio/Videoconferencing"; proceedings of *2002 International Conference on Internet Computing IC'02*: Las Vegas, USA, June 24-27, 2002. http://grids.ucs.indiana.edu/ptliupages/publications/intl-sub03.pdf
18) Hasan Bulut, Geoffrey Fox, Shrideep Pallickara,Ahmet Uyar and Wenjun Wu, *Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service* IASTED International Conference on Communications, Internet, and Information Technology, November 18 to November 20, 2002, in St.Thomas, US Virgin Islands. http://grids.ucs.indiana.edu/ptliupages/publications/AVOverNaradaBrokering.pdf
19) Geoffrey Fox, Shrideep Pallickara, and Xi Rao, "*A Scaleable Event Infrastructure for Peer to Peer Grids*", proceedings of 2002 Java Grande/ISCOPE Conference, Seattle, November 2002, ACM Press,

ISBN 1-58113-599-8, pages 66-75.
http://grids.ucs.indiana.edu/ptliupages/publications/ScaleableEventArchForP2P.doc
20) OASIS Web Services for Remote Portals (WSRP) and Web Services for Interactive Applications (WSIA) http://www.oasis-open.org/committees/
21) Sangmi Lee, Geoffrey Fox, Sunghoon Ko, Minjun Wang, Xiaohong Qiu, *Ubiquitous Access for Collaborative Information System using SVG*, Proceedings of SVGopen conference July 2002, Zurich, Switzerland. http://grids.ucs.indiana.edu/ptliupages/projects/carousel/papers/draft.pdf
22) W3C Scalable Vector Graphics (SVG) version 1.0 Specification http://www.w3.org/TR/SVG/.
23) Batik project at: http://xml.apache.org/batik.
24) W3C Document Object Model (DOM) Level 2 Core Specification http://www.w3.org/TR/DOM-Level-2-Core/
25) W3C Document Object Model (DOM) Level 1 Specification at http://www.w3.org/TR/REC-DOM-Level-1/.
26) W3C Document Object Model (DOM) Level 2 Events Specification at http://www.w3.org/TR/DOM-Level-2-Events/.
27) Geoffrey Fox, Sung-Hoon Ko, Kangseok Kim, Sangyoon Oh, Sangmi Lee on Integration of Hand-Held Devices into Collaborative Environments at http://grids.ucs.indiana.edu/ptliupages/projects/carousel/papers/PDA_IC2002.pdf . Proceedings of the *2002 International Conference on Internet Computing* (IC-02) Volume 2 pp. 231-238.
28) W3C WSDL version 1.2 at http://www.w3.org/TR/2003/WD-wsdl12-20030124/.
29) A Goldberg. "*Smalltalk-80: The Interactive Programming Environment*". Addison Wesley, 1984.
30) G. Lee, "*Object oriented GUI application development*". Prentice Hall, 1994. ISBN: 0-13-363086-2.
31) The MVC framework and Microsoft Windows at http://infolab.kub.nl/pub/theses/w3thesis/Prototype/mvc.html
32) Explore the underpinnings of the JFC's Swing components at http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html