Building Desktop Application with Web Services in a Message-based MVC Paradigm

Xiaohong Qiu[1, 2], Geoffrey C. Fox[2]

[1]EECS Department, Syracuse University
[2]Community Grids Lab, Indiana University
501 Morton N. Street, Suite 224
Bloomington, IN 47404, U.S.A
xqiu@indiana.edu, gcf@indiana.edu

Abstract

Over the past decade, classic client side applications with Model-View-Controller (MVC) architecture haven't changed much but become more complex. In this paper, we present an approach of building desktop applications with Web Services in an explicit message-based MVC paradigm. By integrating with our publish/subscribe messaging middleware, it makes SVG browser (a Microsoft PowerPoint like client application) with Web Service style interfaces universally accessible from different client platforms ─Windows, Linux, MacOS, PalmOS and other customized ones. Performance data suggests that this scheme of building application around messages is a practical architecture for the next generation Web application client.

Keywords

Message, MVC, Web service, SVG and publish/subscribe

1. Introduction

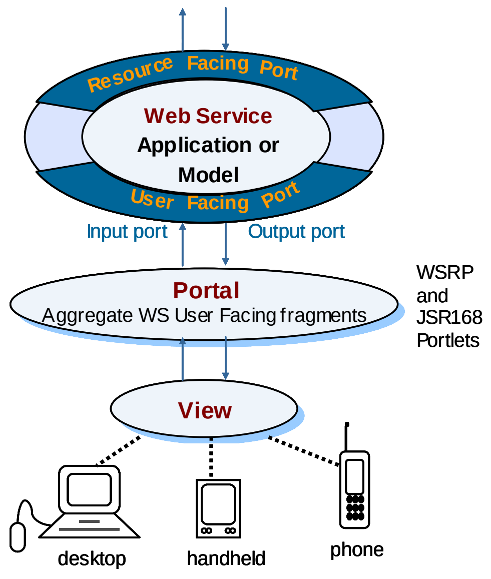


Figure1 Portlet Approach to Web Services and their user interfaces

Web Services are becoming an increasingly important feature of Internet and Grid systems. They support a loosely coupled service oriented architecture that builds on previous distributed object architectures like CORBA, Java RMI, and COM to provide scalable interoperable systems. The broad applicability of this approach includes enterprise software, e-Science and e-Business. Correspondingly there are a growing number of powerful tools that are available for building, maintaining and accessing Web Service-based systems. These tools include portals that allow user frontends to Web Services. This model for user interaction has new standards like portlets with WSRP (Web Services for Remote Portlets) [1] and the Java Specification Request JSR168 [2] supporting lightweight interfaces to the backend resources. This architecture shown in fig.1 implements the Model-View-Controller or MVC [3] [4] architecture with a clean message based interface partially specified by the portlet standard. The general MVC approach of fig. 2a) is a well established paradigm which has been used for many years. As we describe later in more detail, traditional MVC applications employ method-based interactions between the components with this approach giving the needed high performance for interactive applications. In this paper, we explore "*explicit message-based MVC*" [5] ─ a different approach with MVC being used systematically but with message based interactions (shown in fig. 2b), between the model and the view components. We suggest that modern computers and networks are fast enough that this approach will give adequate performance for desktop applications such as those in the Microsoft Office Suite. We embody this idea as the "*MVC rule of the Millisecond*" [6]. This asserts that message based interactions between "nearby" components have an intrinsic delay of less than a millisecond and so this linkage approach is possible whenever such a delay is acceptable. Simple non optimized Java messaging gives such a delay whenever the components are either on the same computer or on machines with a local area connection.

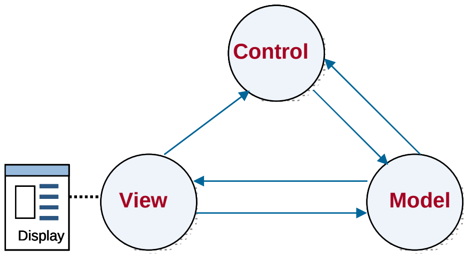
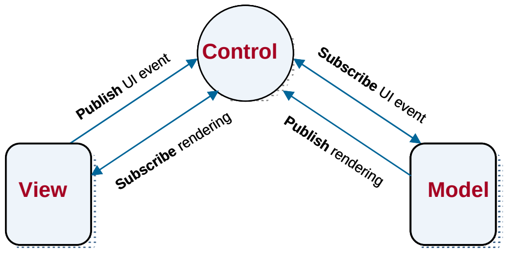


Figure 2a) Conventional MVC paradigm



2b) Explicit message-based Publish/Subscribe MVC model

In this paper, we explore this area and make two contributions. Firstly we look at existing method-based MVC application – the Batik SVG browser from Apache [7] – and convert into a message-based approach as contrasted in figures 2a) and 2b). We discuss some of the issues that came up in this conversion. Secondly we use this message-based version of SVG to explore the overhead in the message-based approach. We find it is between 0.1 and 1 millisecond for "*model*' and "*view*" distributed in different sites on Indiana University's Bloomington campus and the user does not distinguish the interactive experience in switching from method to message-based interactions.

In section 2, we describe some background on the World Wide Web Consortium (W3C) standard of document object model (DOM) [8] [9] [10] and Scalable Vector Graphics (SVG) [11]. One advantage of testing our approach with SVG is that it is fully compatible with the W3C DOM and we can expect the latter to be used in future desktop applications. Thus experience with SVG should extrapolate to "next generation desktop applications". The next section describes in detail our work while section 4 briefly discusses an application to collaboration described in more detail in earlier papers [5] [12]. Conclusions are in section 5.

2. Technology Background: W3C DOM and SVG

Due to the difference of document object models that implemented by earlier versions of major browsers, web developers access HTML document using JavaScript had to write wrapper code to reconcile the incompability between Netscape and Internet Explorer. In 1998, W3C proposed Document Object Model (DOM) level 1 specification that defines "*a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents*." [8] DOM level 2 standard further specifies a generic event model [10]. Version 5 browsers (Mozilla NGLayout engine (Gecko) [13] and Microsoft Internet Explorer 5 [14]) implemented the DOM specification.

The impact generated by the new standard goes beyond scripting community who build cross-browser dynamic pages. DOM carefully defines "just" the logical structure of "document" and an API (application programming interface). Such a standard can effectively specify any XML describable information, which means it can reflect structure of Meta data abstracted by XML schema. DOM also allows any language bindings; therefore it can be used by variety of applications. SVG is an example of DOM application.

SVG, as defined by W3C, is "*a language for describing two-dimensional graphics and graphical applications in XML*" [11]. Compared with HTML content, SVG has richer web graphics flavor. It has the following features (including those inherited from XML and DOM) and makes it a unique technology:

- Rich graphical content ─ SVG includes three types of graphical objects (vector shape, text and image) that can be nested, grouped, transformed and styled, in addition to graphical processing (clipping, masking and filtering).
- Vector graphics ─ SVG content can be dynamically updated (zoom, rotate and translate) without loss of rendering resolution.
- Interoperability and scripting ─ Java and JavaScript binding with SVG, enriches its capability of interactivity, hyper linking, scripting and animation as a rendering content
- XML feature ─ makes it an attractive portable intermediate format for exporting (e.g. from Illustrator and PowerPoint); transcoding between vector graphics (e.g. pdf and PowerPoint) and from vector to rasterized graphics (e.g. PNG); third party application (e.g. JDBC). Through supporting of XLINK [15], SVG provides strong capability for URI referencing of both internal document fragment and remote external document object.
- DOM feature ─ SVG DOM has a tree like structure with nodes of parsed graphics objects. It allows complete access and manipulate of the objects and their properties.

A growing number of SVG viewers are developed for rendering SVG format content. Among them, Adobe [16] and Corel [17] implemented SVG as a plug-in of a conventional browser; Apache Batik SVG browser is a stand-alone client application, which is written in Java apart from a few native classes; there are also customized SVG implementations for handheld devices.

3 Message-based MVC and SVG

3.1 Introduction

We choose an existing system ─ Batik SVG browser [7], and modify its architecture from a method-based desktop application to a message-based one. This has several implications.

The message-based architecture allows one to build desktop applications as web services and so unify traditional desktop and web service plus portal approaches. This unification makes collaborative applications straightforward to build as described in section 4. Further the separation of *model* and *view* makes it easier to support diverse client devices and operating systems. This could be significant with the growing interest in PDA and Linux clients. Note our strategy allows "long distance" linkage between the "*model*" (business logic of application) and view as well as their cooperation on local networks as within a campus. However transcontinental latencies are hundreds of milliseconds and so this cannot be used for interactive experience. As we describe later, we will use the same messaging infrastructure – NaradaBrokering [18] operating in Java Message Service emulation – as has been used to support large Grid applications. This unification of Grid and client applications into a single message-based architecture is key to our paper. We can use our approach for interactive applications when model and view are nearby and allow collaboration and traditional Web portal use for remote access.

Our first goal is a complete analysis of the structure and interaction between components of a real client application. Batik SVG browser is an Apache open source project that implements Scalable Vector Graphics (SVG) specification version 1.0 [11], a recommendation of W3C. Such experience has general significance as it helps us in understanding of similar commercial tools such as Microsoft PowerPoint, Adobe Illustrator and PhotoShop, Corel Draw, and Macromedia Flash which have proprietary implementations.

Secondly, our approach allows building collaborative SVG as a special case of our general Collaboration as a Web Service architecture [19]. This work has been discussed in our earlier papers in Internet Computing 2003 [5] and SVG Open 2003 [12]. We presented a multiplayer chess game as a test case of our collaborative SVG infrastructure without decomposition of the client application. Note in collaboration, one user is typically in charge and this case requires interactive delays of less than a millisecond. As shown in section 4 and fig. 7, the other users receive change events multicast by the messaging system. These events can be delayed as all applications in a session are similarly treated and the pipelined events give a satisfactory real-time experience.

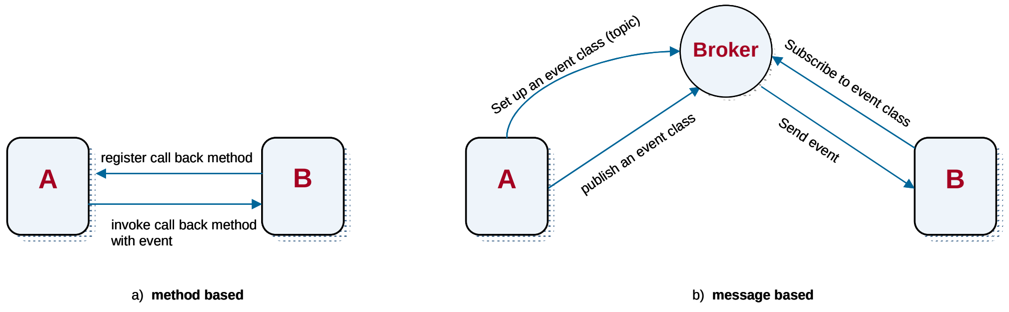


Figure 3 Message or method based Publish/Subscribe

Thirdly, we demonstrate a way of modification in architecture level which enables the conversion of a client application into a distributed system and identify the difference in design principles. MVC is a frequently used paradigm in modern architecture design (e.g. Microsoft Windows). In a "conventional" MVC, "*controller*" executes its tasks through method calls since messages are hidden in system level. We make a critical observation, namely "conventional" MVC has to be replaced by an "*explicit message-based*" MVC in order to enable components of the application to be distributed. In our approach, we use "explicit control messages" to abstract the semantic meanings of "*controller*" so that messages of the original system are exposed and pull into application level. Such abstraction generates structural changes as the following:

a) Original client application is physically split into client user interface ("*view*") and core functional component ("*model*"). The latter naturally becomes a Web Service on server side.
b) Method calls, which play the role as "*controller*" in a client application, are taken over by "explicit control messages" that communicate between client interface and Web Service server through network.
c) Our approach requires us to support our model view linkage with a high performance messaging middleware infrastructure. We use NaradaBrokering [18] which has been separately developed and provides a variety of publish subscribe models including peer-to-peer and Java Message Service (JMS) [20] emulation. Our results are not sensitive to the details of NaradaBrokering and do not currently exploit its ability to traverse firewalls and support multiple protocols. Our use for collaborative SVG would exploit these latter Grid messaging capabilities of NaradaBrokering.

The changes bring up issues that cause a challenge to the system:

- timing becomes a compelling issue —with the separation of client and Web Service server, original assumption and design principle break since time scope drastically increases from microsecond level (e.g. a Java method call) to millisecond level (network latency plus system overhead).
- Object serialization is a must have toolkit — messages, as a linkage vehicle, contains component information from both sides and keep context same. Synchronization is a factor to consider for context consistency.

3.2 Message-based Event model

The basic idea is very simple and illustrated in fig. 3. Traditional event-based programming is used extensively in the Batik SVG browser and most modern applications. Different parts of a program are linked asynchronously with one part producing events that are passed to listeners whose call-back method has been passed to the producer. As shown in fig. 3b) this can also be implemented with explicit messages where listeners subscribe to an event class (topic) and events producers publish them to this topic. Our strategy is to replace the listener model of fig. 3a) by the publish/subscribe broker model of fig. 3b). Note that either approach can use explicit queues (maintained on a broker in the message case) or alternatively integrate the broker into the producer as in most simple method-based event models.

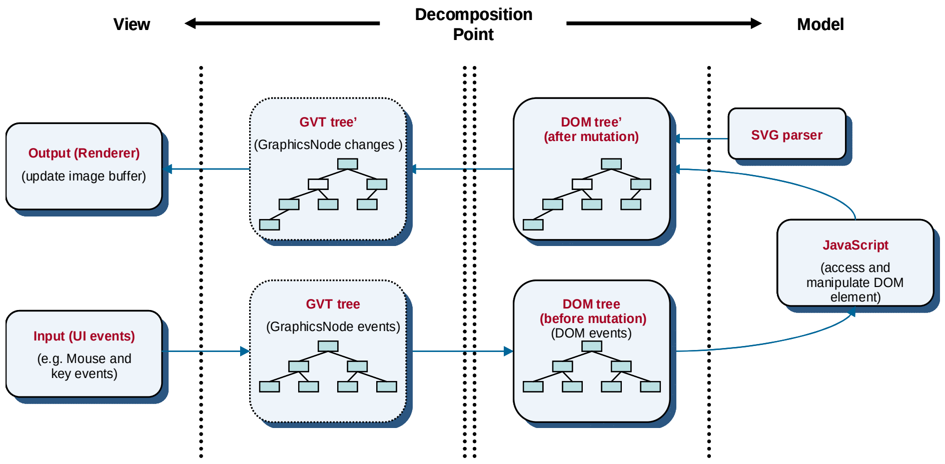


Figure 4 Decomposition of SVG browser into stages of pipeline

As shown in figure 4, one can use this strategy in several parts of the SVG browser and in doing so produce multiple web services coordinated into a single application; there are natural event linkages between the client user interface and the GVT (or Graphic Vector Toolkit, an internal module to represent graphical view of DOM) tree used in Batik; another between GVT and the DOM tree and finally that between the DOM and the Java or JavaScript application. After substantial experimentation, we chose to split the SVG browser between the DOM and GVT tree. The resultant Web Service architecture is shown in figure 5. This choice has the advantage that it naturally generalizes to other DOM applications. However we made for more pragmatic reasons as other choices appeared to require major restructuring of the existing software. Our search for appropriate places to split applications into message separated services illustrated two important principles.

- Firstly one should split at points where the original method based linkage involved serializable Java objects. Serialization is needed before the method arguments can be transported and this is familiar from Java RMI.
- More seriously we found that the Batik often involved large classes that implemented many different interfaces. These interfaces often came from different parts of the program and crossed the possible stages mentioned above. Such "spaghetti" classes as in fig. 6a) implied that additional state information would need to be transmitted if we split at points where classes spanned interfaces from different modules. Of course the message based paradigm (fig. 6b) tends to force a more restrictive programming model where all data is shared explicitly and not implicitly via interfaces crossing splitting lines.

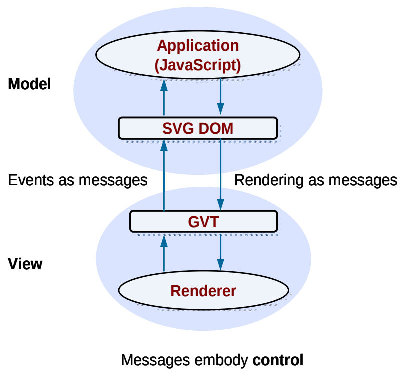


Figure 5 SVG browser derived from message-based MVC

4 Collaborative SVG Applications

We have explained how one can make message-based network applications collaborative in two modes – *shared input port and shared output port* [21] [5]. In each case one multicasts the messages – either those arriving at a Web Service (shared input port) or those produced by the Web service (shared output port).

NaradaBrokering was explicitly designed to support this model and has been used for a variety of cases from audio-video conferencing, text chats, white boards, and shared display as part of the Anabas [22] collaboration environment.

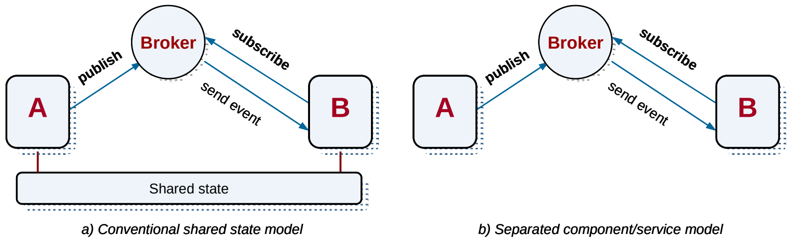


Figure 6 Implicit and explicit state

We have already described this idea for SVG [5] although at the time we did not explicitly break up SVG into a separate model and view component as required by the Web service architecture. Rather as shown in figure 7, we intercepted the events on a master application and allowed NaradaBrokering to multicast them to the collaborating clients. This corresponds to the shared input port model in our architecture. We will rebuild the collaboration environment with explicit web service models and demonstrate both shared input and shared output port models in the next two months.

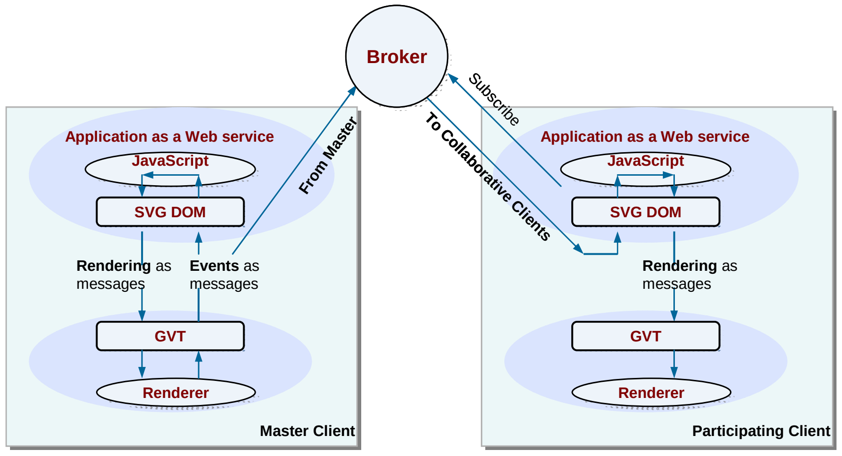


Figure 7 Shared Input port of collaborative SVG

5. Performance Testing

We have started an extensive series of performance measurements to demonstrate the viability of our approach. There are many variables including position of Model, View, and Event Broker (NaradaBrokering) and the choice of type of host computer and network connection. One can also vary the application running in the Model Web service. One can investigate either the single Model and View or the collaborative models. We expect the main impact to be the algorithmic effect of breaking the code into two and the network and broker overhead. We know already the latter will be large so that for a broker in the UK and a Model and View in Indiana, the overhead would be a horrendous 500 milliseconds. Here we present some initial investigation with Brokers, Model and View "nearby". In each of the three tests, the Model and View were run on middle aged Dell Windows 2000 PC's with 1.5 GHz Pentium 4 processors. In the first test the two PC's are in the same office and served by an aged Sun Solaris UltraSparc 2 laboratory server. In the next two tests, the Broker and View are on one PC and the View Web service on the second one. In test 2, the PC's are directly connected by Ethernet and in test 3 they use a 802.11 low-end wireless link.

We present here four timings for each of the test scenarios with the timing positions shown in fig. 8 which is a simplified version of the pipeline shown in fig. 4. The results in table 1 give mean the error in its determination and the standard deviation. The times T_0 T_1 T_2 T_3 T_4 and T_5 are all measured in the View and defined as follows

- T_1: A given user event such as a mouse click can generate multiple associated DOM change events transmitted from the Model to the View. T_1 is the arrival time at the View of the first of these.
- T_2: This is the arrival of the last of these events from the Model and the start of the processing of the set of events in the GVT tree
- T_3: This is the start of the rendering stage
- T_4: This is the end of the rendering stage

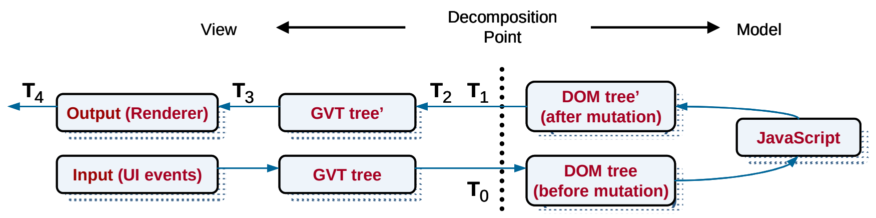


Figure 8 Timing points

Table 1 gives all times with reference to T_0. The overhead lies between our measured values of T_2-T_0 and T_1-T_0 i.e. between 100 and 200 microseconds. Each timing is an average of several hundred non-trivial mouse events with the SVG application running the JavaScript chess game presented in ref. 5. In later publications we will present a more refined analysis which studies the structure of the different types of mouse events and compares with the unmodified program. However, we can already conclude that message passing is a negligible overhead being less than 200 microseconds which is small and less than the 300 microseconds used on the average for GVT processing and rendering T_5-T_2. We note that we were careful not to flood the system with irrelevant mouse-over and mouse-move events; the latter are generated each time there is a one pixel change in the mouse position. Processing of mouse event messages is dominated by "start-up time" or latency. Thus we can buffer multiple mouse-move events into one "vector" event and get the small overheads reported here.

Table 1 Timing of Stages in microseconds

	First arrival from Model: T_1-T_0		Start Process DOM T_2-T_0		Start Rendering T_3-T_0		End Rendering T_4-T_0	
	mean ± error	stddev	mean ± error	stddev	mean ± error	stddev	mean ± error	stddev
Test 1 (Solaris server)	110±5.0	95.0	180±10.0	184.0	243±11.0	204.0	478±13.0	238.0
Test 2 (direct connect. Desktop server)	108±5.0	132.0	180±7.0	170.0	234±8.0	194.0	485±12.0	272.0
Test 3 (wireless connect. Desktop server)	113±3.0	54.0	212±5.0	77.0	225±5.0	78.0	510±5.0	78.0

6. Conclusions

We believe our prototype shows how a message-based MVC (three-stage pipelines) model can generate a powerful application paradigm suitable for SVG and other presentation style applications. As SVG is an application of the W3C DOM, we can generalize the approach for other W3C or similar DOM based applications. Our approach suggests that one need not develop special "collaborative" applications. Rather any application developed as a Web service can be made collaborative using the tools and architectural principles discussed in this paper. Note that Moore's law implies that computer performance will continue to improve while networks will also continue to increase in bandwidth with however latency for long distance linkage remaining higher than that needed for interactive use. Thus inevitable infrastructure improvements will tend to make our approach more attractive in the future.

These ideas can also suggest a uniform approach to user interface design with desktop and web applications sharing a common portlet (WSRP, JSR168)-based architecture. This could motivate the development of new desktop applications with many capabilities not present in today's systems such as Openoffice and Microsoft Office. We are currently looking at extending our ideas to Openoffice while a limited implementation is possible using the rather crude event interface exposed for PowerPoint [Wang04]. These ideas can unify PDA and desktop, as well as Linux, MacOS, Windows and PalmOS applications.

In our final paper we will present the unified architecture from desktop to internet collaboration with much more extensive performance measurements.

References

1) OASIS *Web Services for Remote Portlets Specification*, http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf
2) Java Specification Request *JSR168 Specification* http://www.jcp.org/aboutJava/communityprocess/final/jsr168/
3) A Goldberg. *Smalltalk-80: The Interactive Programming Environmen*. Addison Wesley, 1984.
4) G. Lee, *Object oriented GUI application development*. Prentice Hall, 1994. ISBN: 0-13-363086-2.
5) Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox *Collaborative SVG as A Web Service* in Proceedings of SVG Open, Vancouver, Canada, July 2003 http://www.svgopen.org/2003/papers/CollaborativeSVGasAWebService/#S.Bibliography or Word version at
6) Geoffrey C. Fox, *Software Development around a Millisecond*. http://grids.ucs.indiana.edu/ptliupages/publications/cisejano4.pdf in CISE Magazine
7) Apache Batik SVG Toolkit http://xml.apache.org/batik/
8) W3C Document Object Model (DOM) level 1 specification http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/
9) W3C Document Object Model (DOM) Level 2 Core Specification http://www.w3.org/TR/DOM-Level-2-Core/.
10) W3C Document Object Model (DOM) Level 2 Events Specification at http://www.w3.org/TR/DOM-Level-2-Events/.
11) W3C Scalable Vector Graphics (SVG) version 1.0 Specification http://www.w3.org/TR/SVG/.
12) Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox *Internet Collaboration using the W3C Document Object Model* in Proceedings of the 2003 International Conference on Internet Computing, Las Vegas June 2003 http://grids.ucs.indiana.edu/ptliupages/publications/collaborative_dom_conference_2003_Int_IC_font10_without_title_page.pdf
13) Mozilla Layout Engine http://www.mozilla.org/newlayout/
14) Microsoft Internet Explorer http://www.microsoft.com/windows/ie/default.asp
15) W3C XML Linking Language (XLINK) version 1.0 http://www.w3.org/TR/xlink/
16) Adobe SVG Zone http://www.adobe.com/svg/main.html
17) Corel SVG viewer http://www.smartgraphics.com/Viewer_prod_info.shtml
18) Community Grids Lab NaradaBrokering system at http://www.naradabrokering.org
19) Geoffrey C. 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
20) Sun Microsystems Java Message Service at http://www.hostj2ee.com/specs/jms1_0_2-spec.pdf
21) Geoffrey Fox, Dennis Gannon, Sung-Hoon Ko, Sangmi Lee, Shrideep Pallickara, Marlon Pierce, Xiaohong Qiu, Xi Rao, Ahmet Uyar, Minjun Wang, Wenjun Wu, *Peer-to-Peer Grids*, Chapter 18 of [*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, March 2003. http://www.grid2002.org]
22) Anabas Conferencing system http://www.anabas.com

23) Minjun Wang, Geoffrey Fox and Shrideep Pallickara *A Demonstration of Collaborative Web Services and Peer-to-Peer Grids* to appear in proceedings of IEEE ITCC2004 International, Las Vegas April 5-7 2004. http://grids.ucs.indiana.edu/ptliupages/publications/wangm_collaborative.pdf