Internet Collaboration using the W3C Document Object Model

By Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox

Key words
Collaborative, MVC, DOM, SVG, event and Web services

Abstract

The Internet makes it possible to share information (e.g. text, image, audio, video and other formats of data) across the globe. In this paper we look at collaborative Internet environments for applications whose user interface is described by the W3C DOM – this can be expected to be a standard for browsers and other office tools and so of general importance. We demonstrate a powerful general approach, which first uses the MVC (Model View Controller) paradigm to restructure applications as Web Services and then applies a general approach to making Web services collaborative. We demonstrate the essential ideas with the Java open-source SVG (Scalable Vector Graphics) browser and describe the key general features of this way of building interactive Web-based applications.

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 well understood – 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.

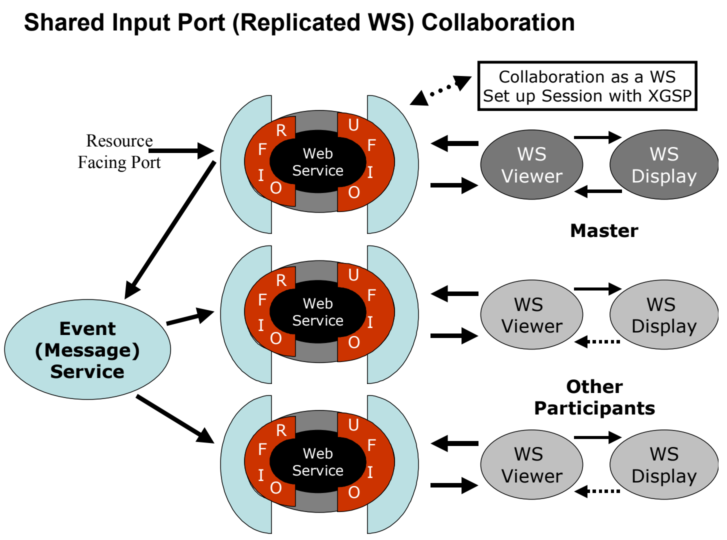

Figure 1 *One way of setting up Collaborative Web services involving replicating the application and using the system event service (in our case NaradaBrokering) to share state-defining messages. We show user-facing (UFIO) and Resource facing (RFIO) Web service ports*

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]. This standard is layered on top of the basic WSDL specification of Web Service ports [21].

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 [22], 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 [23] – whose full source is available from Apache [24]. This application also has the important feature that it faithfully supports the W3C document object model DOM [25-27] that 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 [28]. 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 section, we briefly review the MVC (Model View Controller) approach, which is closely related to the Web service design pattern. More details can be found in [29]. 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. MVC Paradigm

The well-known Model-View-Controller (MVC) framework [30] 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 MVC model has been the basis for most widely used graphical environments nowadays [31]. 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 [32] and Java Swing UI components [33] are examples of event-driven MVC architecture.

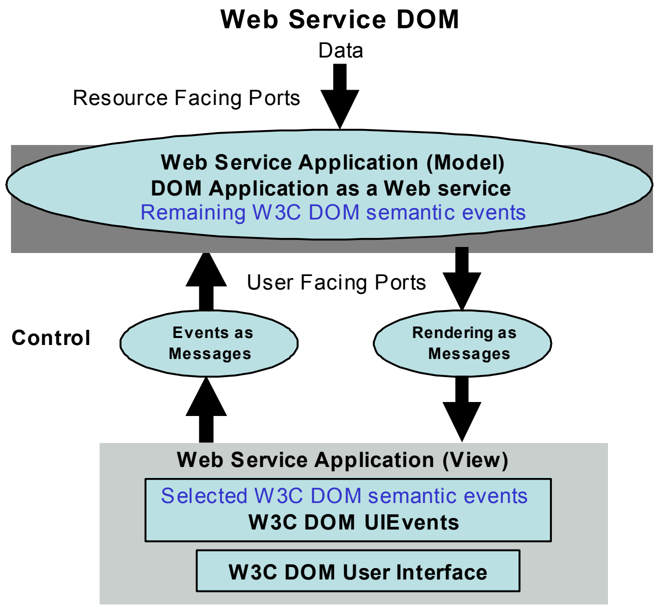
Web Service DOM
Data
Resource Facing Ports
Web Service Application (Model)
DOM Application as a Web service
Remaining W3C DOM semantic events
User Facing Ports
Control
Events as Messages
Rendering as Messages
Web Service Application (View)
Selected W3C DOM semantic events
W3C DOM UIEvents
W3C DOM User Interface

Figure 2 DOM Application as a Web Service

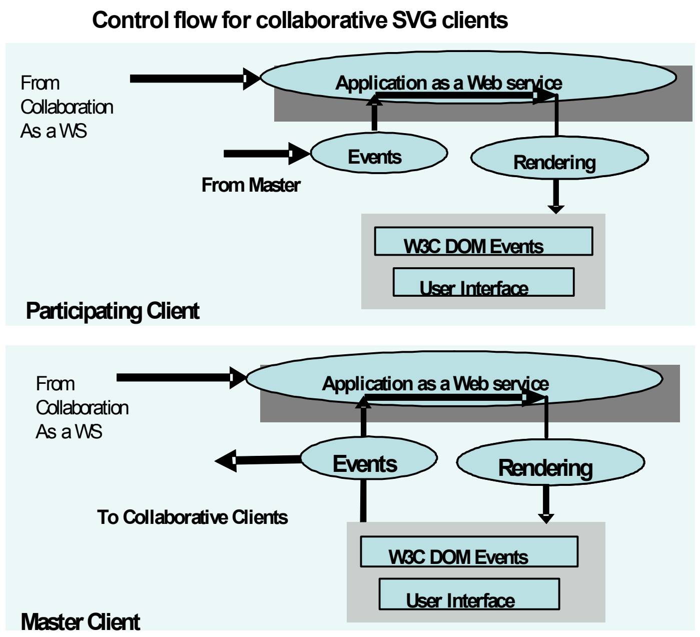

Figure 3 Control flow for collaborative SVG clients

3. Structure of SVG Web Service

The essential decomposition of SVG and related applications can be derived from the MVC paradigm. We take the *Model* component and this essentially becomes the Web Service as shown in fig. 2, while the *View* becomes the user interface. They are linked by the NaradaBrokering publish/subscribe messaging system; the combination of this with the preparation and interpretation of messages corresponds to the *Controller* MVC component. We analyze all possible events and divide them into DOM UIEvents (mouse and keyboard events) and semantic events (such as zooming). UIEvents are generated in the *View* and are converted into messages for the *Model*. One can design different *View* modules (with trade-offs in complexity and performance) through choice of which semantic events to process in the *Model* and which in the *View* component.

We support collaboration in two extremes; firstly the shared input port model where one replicates Web services and delivers events generated on a master *View* client to all instances of the *Model*. These service their associated *View* component. This has maximal flexibility for customization of each collaborative client while in the shared output port of service collaboration, a single *Model* instance uses NaradaBrokering to multicast rendering information to all collaborating *View* modules. We show the resultant message passing in fig. 3 separately for the master client defining the application state and other clients participating in the session. We show the two types of messages – those defining overall context (Collaboration as a WS) and those corresponding to this SVG application. Both are routed appropriately by NaradaBrokering.

4. Structure of Collaborative W3C DOM/SVG Events

We define a collaborative event as an object that wraps original SVG events with additional context information for collaboration and Web service model. The context information helps to guide the events through the NaradaBrokering system 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 the SVG DOM. The *Model* part of Web service application analyses the SVG event based on its type and then delivers the resultant rendering information to the associated *View*(s). Below we summarize key types of event and their structure.
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 the W3C 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 holds the "master" token and generates master events that trigger collaborative behavior in the communication group. We term events that come from other participating clients are non-master events. The master token can be changed dynamically. Further as discussed below, non-master clients can – as in all such collaborative architectures – choose whether or not to follow precisely the master's state.

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 will join and leave a collaborative session asynchronously; a client system will crash and reboot; the replay service (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. Note NaradaBrokering can save all published events (simply by subscribing a persistent store to the session) and so always replay can be supported.

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 the 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 as generated by the DOM
- Event types (e.g. master/non-master, major/minor type)
- 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.

5. 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, we have demonstrated a general approach involving conversion of the applications to an event-driven "message passing MVC" model (figure 2). One separates the user interface interactions from core computation or processing functions using message passing ─ systems with object-oriented design as illustrated by the Java Batik SVG are particularly suitable for this strategy. Traditionally shared event collaborative applications were generated by identifying state specifying actions as messages. We follow this idea but by going through the Web Service route, create a cleaner more powerful architecture, which has great value even when collaboration is not needed.
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.
4) In the final paper we will give more detail of the core events and study the performance of the Web Service when compared to the conventional packaging.

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) W3C WSDL version 1.2 at http://www.w3.org/TR/2003/WD-wsdl12-20030124/.
22) 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.
23) W3C Scalable Vector Graphics (SVG) version 1.0 Specification http://www.w3.org/TR/SVG/.
24) Batik project at: http://xml.apache.org/batik.
25) W3C Document Object Model (DOM) Level 2 Core Specification http://www.w3.org/TR/DOM-Level-2-Core/.
26) W3C Document Object Model (DOM) Level 1 Specification at http://www.w3.org/TR/REC-DOM-Level-1/.
27) W3C Document Object Model (DOM) Level 2 Events Specification at http://www.w3.org/TR/DOM-Level-2-Events/.
28) 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.
29) Xiaohong Qiu, Bryan Carpenter and Geoffrey Fox, Collaborative Web services and the W3C Document Object Model, http://grids.ucs.indiana.edu/ptliupages/publications/collaborative_domfeb28-03.pdf.
30) A Goldberg. "*Smalltalk-80: The Interactive Programming Environment*". Addison Wesley, 1984.
31) G. Lee, "*Object oriented GUI application development*". Prentice Hall, 1994. ISBN: 0-13-363086-2.
32) The MVC framework and Microsoft Windows at http://infolab.kub.nl/pub/theses/w3thesis/Prototype/mvc.html.
33) Explore the underpinnings of the JFC's Swing components at http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html.