Web Service Architecture for e-Learning

Xiaohong Qiu
EECS Department, Syracuse University
Community Grids Lab, Indiana University
501 Morton N. Street, Suite 224
Bloomington, IN 47404, USA

and

Anumit Jooloor
CS Department, Indiana University
Bloomington, IN 47405, USA

ABSTRACT

Message-based Web Service architecture provides a unified approach to applications and Web Services that incorporates the flexibility of messaging and distributed components. We propose SMMV and MMMV collaboration as the general architecture of collaboration based on a Web service model, which accommodates both instructor-led learning and participatory learning. This approach derives from our message-based Model-View-Controller (M-MVC) architecture of Web applications, comprises an event-driven Publish/Subscribe scheme, and provides effective collaboration with high interactivity of rich Web content for diverse clients over heterogeneous network environments.

Keywords: Web Service, Message-based MVC, messaging, Publish/Subscribe, SMMV, MMMV, e-education and collaboration

1. INTRODUCTION

The Internet provides a distributed infrastructure for sharing information globally with an estimate that the online population will reach 6,330 million users in 2004 [1]. The very large user market becomes a great motivation for new technologies enabling one to build the next generation of Web based applications. In particular, it is very attractive to develop collaborative applications linking of the growing number of diverse clients with rich media Web content.

This evolution brings fundamental changes to our society in communication and knowledge acquisition pattern ─ anytime, anywhere, people no longer have to meet face to face to communicate while all information is delivered to the client interface online and on demand. The new trend comprises innovative technological features: it offers a platform facilitating ubiquitous access of desktop, PDA and cellular phone (Windows, MacOS, UNIX, Linux, and PalmOS) clients; it supplies an interface with services for easier availability to global resources including data, text, 2D and 3D graphics, video/audio stream, and MP3 music; it promotes an interoperable synchronization mechanism that captures interaction between participants ─ teacher and student, trainer and trainee for real time experience.

Collaboration tools are revolutionizing the training industry. Particularly, Web-based e-Learning solutions are adopting collaboration environments that enhance accessibility to a full range of educational resources supporting rich interaction with participating parties in a synchronous or asynchronous fashion. Distance education has been successful using tools such as Audio/Video conferencing and shared curriculum using either shared display or shared event architectures. Here we explore a richer model with Single Model Multiple View (SMMV) and Multiple Model Multiple View (MMMV) collaboration as the general architecture of collaboration as Web Service [2] model, which can be applied to both instructor-led learning and participatory learning. The premise of this work is building forward-looking architecture of Web applications for scalability, reusability, interoperability, pervasive accessibility, and automatic collaboration.

The rest of the paper is organized as follows: in Section 2 we briefly review technical issues that cover general concept of building message-based Web Service applications, relationship of our message-based Model-View-Controller (MVC) [3] architecture of Web applications and derived SMMV and MMMV collaboration as a Web Service model for e-Learning, and methodology of our prototyping. Section 3 summarizes collaboration framework and presents SMMV and MMMV collaboration as our general architecture of collaboration as Web Service model. In section 4, we discuss performance issues based on the experiments of Scalable Vector Graphics (SVG) applications. Finally, we present our conclusions and propose future work.

2. TECHNICAL ISSUES

2.1 Message-based Web Service model and messaging infrastructure

The history of Internet and Web technology saw the evolution of Web applications with architectures dominated by centralized client-server system with traditional point-to-point (unicast) connection, decentralized self-organizing peer-to-peer (P2P) system that evolved to overlay network with application level multicast mechanism, and RPC-model (e.g. CORBA) derives from method-based system calls for tightly coupled single CPU system (e.g. desktop applications) but with remote procedure calls to support the distributed objects. Client-server and P2P models are suitable for solving problems with features applicable to their patterns but real world problems can be arbitrarily complicated. Examples can be seen in parallel applications with decomposition in high dimensionality. On the other hand, RPC-like model deals well with distributed objects or components for reusability but do not scale well. Message-based Web Service model provides a unified approach that incorporates messaging flexibility with components distribution. It accommodates to the diverse and scaling nature of the Internet and also promotes Web applications development with Web Services for reusability, interoperability, and scalability.

The messaging approach decomposes a Web system into three layers: physical networks or Internet, messaging infrastructure, and Web application. This separation can greatly improve applications' portability by reducing their dependency on underlying connection topologies and platforms. It also reduces deployment overhead of Web applications. However, it requires a powerful messaging infrastructure on TCP/IP network stack providing a variety of communication services that reconcile the differences between underlying connection topologies and deployment of high-level applications. In our lab, we have developed an open source messaging infrastructure NaradaBrokering [4] that supports a publish/subscribe paradigm. It provides Java Messaging Service (JMS) [5] compliance and JXTA [6] interaction and has been applied to a suite of collaboration tools (Audio/Video conferencing [7], Carousel [8] and Anabas e-Learning platform [9]). NaradaBrokering supports multiple protocols (including TCP/IP, UDP, HTTP, and multicast), firewall tunneling, security, and heterogeneous services (e.g. messaging services and grid services).

2.2 Web application and Internet collaboration

Web application deployment shows diversified directions but have common features ─ namely, user interfaces and services for the sharing of information and resources over Internet infrastructure. The "sharing" can be done asynchronously and synchronously at every possible stage along the deployment pipeline. The objects that need to be synchronized may range from Web contents (e.g. video, audio and raw data streams), user interactions (e.g. editing operations on shared whiteboard document), distributed programs (e.g. distributed large-scale simulation components), to team participants who involve in development or management. The "sharing" can be organized through unicast or multicast style of group communication. Therefore, in the most general sense, collaboration is the core problem and service of Web applications of "sharing" although people usually refer the terminology "collaboration" to real-time synchronous Web applications with compelling time issue or constraints. As key objective of our approach, design and implementation of a uniform architecture for Web applications with automatic collaboration capability has general importance.

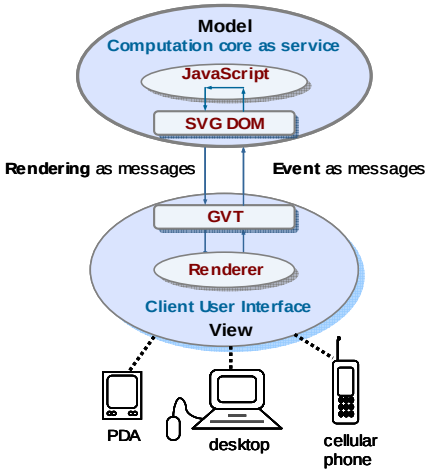

Figure1 SVG browser derived from message-based MVC

We have looked at several examples as part of systematic exploration of our design concepts: We proposed an "*explicit message-based MVC*" paradigm (M-MVC) [10] as the general architecture for Web applications. It is built around systematic use of Web services and an event driven message-based separation between *model* and *view* in the MVC pattern. We have carried out initial "*collaboration as a Web Service*" [11] experiments to test viability of our architecture in supporting of interoperable applications with rich graphical contents and tight time constraints through examples of teacher-student scenario and multi-player online game. As an extension to our research scope, we converted desktop application to distributed system at architectural level. This is done through replacing conventional method-based MVC with message-based MVC in Publish/Subscribe scheme [12] for maximum reusability of existing software assets. We have in-depth discussions of performance issues for Web-based applications [13] that help to investigate the process of message-based Web application deployment

and supply feedback for the construction of underlying messaging infrastructure, which is indicative especially when this area is still immature and one expects substantial evolution. Finally, as discussed in this paper, we define collaboration with SMMV and MMMV model derived from our uniform Web Service architecture of M-MVC that converge desktop and Web applications.

2.3 SVG and DOM

As key challenge of a new approach with systematic utilizing Web Service for building message-based applications, many subtle factors may not be addressed by general architectural consideration. So, we choose to build a prototype with a forward-looking architecture and

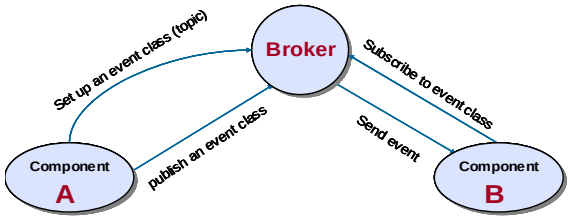

Figure2 Event-driven message-based Publish/Subscribe scheme

conduct systematic experiments to explore and identify general principles and key implementation issues associated with this approach. Multimedia with rich graphics and media stream composition forms an important feature of new general Web applications. We select a presentational style desktop application with two-dimensional Scalable Vector Graphics (SVG) [14] contents ─ Batik SVG browser [15] from Apache as our testing case for further experiments and evaluations. We have restructured the open source Batik software in the explicit message-based MVC model as illustrated in figure1. The SVG document is parsed as a Document Object Model (DOM) [16] tree with nodes representing document fragments and graphics objects. Graphical Vector Toolkit (GVT) tree reflects DOM structure and is used for rendering convenience. In this approach, NaradaBrokering supplies all messaging services (interaction between clients, events and rendering within a client) and provides Publish/Subscribe architecture.

Building collaborative tools on SVG has some important general features:

a) SVG is an open source standard for 2D vector graphics of World Wide Web Consortium (W3C). It is an important technology for visualization. XML content format and scalable vector graphics feature make it an ideal choice of intermediate transportation and high-resolution rendering. It has been applied to various applications including mapping services in Geological Information System (GIS) and authoring tools (e.g. SVG viewer plug-in from Adobe and Corel). The latter feature is especially important for universal access from small wireless devices.

b) Batik SVG browser has full support of SVG 1.0 specification. The openness of both standard and implementation provides us valuable experience of a complete analysis of system structure and components interaction, which is unavailable from similar commercial tools (such as Microsoft PowerPoint, Macromedia Flash, Adobe Photoshop and Illustrator, and Corel Draw) with proprietary implementation and data format.

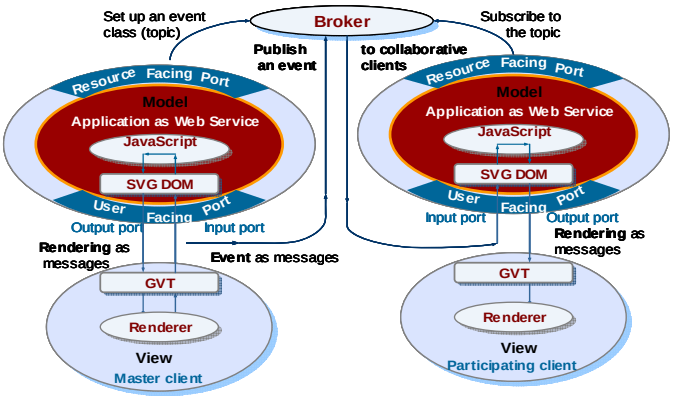

Figure3 Shared Input Port of Collaborative SVG

c) SVG is built on the W3C DOM [16], which is the natural description of all desktop documents used in Office applications of the type built by Microsoft, Macromedia and OpenOffice. Thus our work can naturally generalize to a new Web service architecture for a next generation of desktop applications including those used for curriculum authoring. The DOM events specification provides a generic event model that propagates changes in nodes (objects) of the tree structure. This allows this style of application to have a common web service core that drives a variety of client interfaces which can either be standalone or collaborative.

3. COLLABORATION FRAMEWORK

3.1 SMMV collaborative Web Service

In the message-based MVC framework, one can classify collaboration in a way very familiar from parallel computing. In this case we are very familiar with Flynn's taxonomy [17] which includes the two key architectures SIMD (Single Instruction Multiple Data) [18] and MIMD (Multiple Instruction Multiple Data) [19]. We show that the model of figure 5 can be thought of like SIMD and that of figure 6 as MIMD.

Single Model Multiple View (SMMV) shown in figure 5 corresponds to Flynn's SIMD parallel computing case with multiple clients sharing a single Model component. For the parallel computing analogy we find a single instruction stream shared by multiple data elements. The SMMV collaboration model can be used for lecturing in distance education and is common in client/server Web applications with multiple Web browsers sharing a Web Server.

We have explained how one can make message-based network applications collaborative in two modes – *shared input port and shared output port* [20]. In each case one multicasts the messages – either those arriving at a shared input port or those produced by shared output port. Note that in each case a client assigned with "master" token has "master role". Requests for switching between different roles (e.g. "master" vs. "nonmaster" and player vs. observer) can be done dynamically. Figure 3 and figure 5 illustrate SMMV collaborative Web Service architecture with the shared input port model.

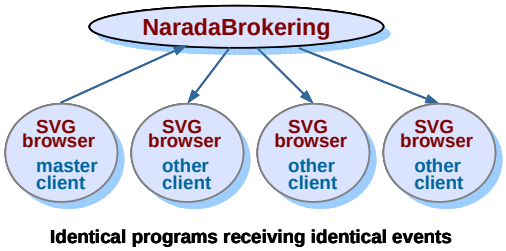

Figure 4 Monolithic collaboration

3.2 MMMV collaborative Web Service

MMMV is a generalization of SMMV, which enables ubiquity with the customization done from the Model at server side and is shown in figure 6. Now we have multiple models each driving its own separate view. This corresponds to Flynn's MIMD with multiple instruction units each driving its own data. We see model maps into CPU and view maps into data as we compare collaboration and parallel computing. Furthermore, we could have hybrid models which can mix SMMV and MMMV. Thus we can have replicated models as in MMMV but with some or all models driving in SMMV fashion more than one view.

A rather deeper issue comes from the many tiers present in most web service (distributed) applications. We can in fact have a general workflow (pipeline) with several *model* and *view* components as illustrated in figure 6. As one example consider JavaServer Faces (JSF) [21], which extends JavaServer Pages (JSP) [22] and Java Servlet [23] technology. This allows a multi-tier Model component with a JSP Web tier and backend business logic. This illustrates that our classification is incomplete as often the Web tier has multiple models even if there is only single business logic. One would classify these systems as SMMV or MMMV depending on the relative importance of Web tier and business logic. Of course there are also confusing cases with multiple services (resources) in the business logic.

Turning to education for examples where we noted that SMMV was the natural distance education paradigm, we see that MMVC is the natural architecture for developing applications such as participatory learning tools.

The utilization of messaging services provided by NaradaBrokering in our collaboration system comprises two cases: one is registration for Publish/Subscribe service in the monolithic, SMMV and MMMV (interfaces among SVG DOM Web Services); the other is the communication between separated *view* and *model*

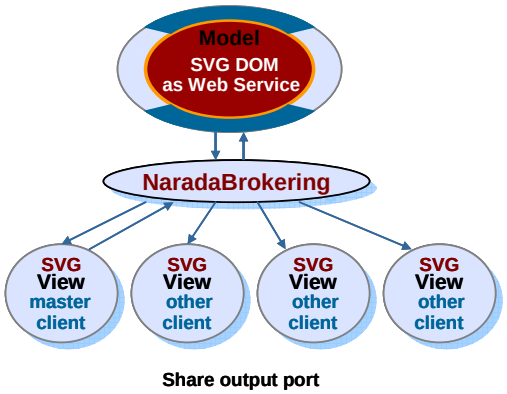

Figure 5 SMMV collaborative Web Service

within each application component. In the latter case, the "Brokers" shown in figure 6 run in NaradaBrokering's point-to-point mode supplying transport services such as security, firewall and NAT traversal, protocol choices and compression.

3.3 Event-based collaboration with Publish/Subscribe scheme

Event-based programming has become a widely used programming style that supports interrupt-handling mechanisms for user input-device interactions. Compared with previous interrupt processing procedures, it promotes system modularity and asynchronous response. Most of modern desktop systems, including Microsoft Windows and their applications use MVC paradigm. The system is decomposed into triad of *Model*, *View*, and *Controller*, which is comparable to computation core (including data structure), visual components, and the communication between those two within a separate class or inherited in either of them. In conventional event-driven method-based MVC, messaging is hidden at

system level, whose differences with our message-based MVC approach are discussed in depth in Ref. [12]. Method-based event approach is also extensively used in distributed systems including Java AWT, Swing and their applications. As a common mechanism of event-based programming, event listener components subscribe to event producer component and get notification of event occurrences. In the MVC cases of Swing and Batik SVG browser, visual components (*view*) form the observers of data structure (*model*) with rendering updating corresponding to model changes.

Our approach of event-driven message-based collaboration with Publish/Subscribe scheme (see figures 2 and 3) has following implications:

- An "event" defines the incremental change of system state. We have given a complete analysis of events and classify them as UI event, SVG/DOM event, and semantic event categories in our collaboration experiments with SVG [11]. Event-based collaboration system works through timely synchronization with updated event information communicated among participatory parties. Moreover, events can be queued and stored as record for retrieval and replay and we have these services in our messaging infrastructure for supporting system reliability, Quality-of-Service and functionality.
- The event workflow of a presentation style application can be illustrated by its propagation along a pipeline with stages consisting of objects (constituent system components). As shown in figure 1, the "U-turn" trip for Batik SVG browser starts from user interaction triggering a mouse event to the completion of update rendering in image buffer. Each stage forms the natural synchronization point for collaboration. In a SVG Web Service model (figure 3), "*Input port*" and "*Output port*" are refer to interfaces between *view* and user facing port of Web Service in input leg and output/rendering leg of the pipeline.
- Event-based collaboration can be implemented in method-based fashion such as those built on top of RPC-like system (e.g. CORBA). However, we adopt a different approach of event-driven message-based Web Service model with details of underlying platforms hidden in the implementation of the messaging infrastructure level. We have elaborated this in the context of our general approach of Web applications deployment in section 2.1. In our approach, communication among distributed components is conducted indirectly through messaging brokers.
- Publish/Subscribe schemes present the capability of handling complex topologies with multiple topics and multiple clients. Our messaging infrastructure provides topic management service and registration (for Publish/Subscribe) service so that the collaboration system can host virtual collaborative community activities (e.g. shared browsers, multiplayer online game, and share whiteboard) in dynamic and parallel fashion.

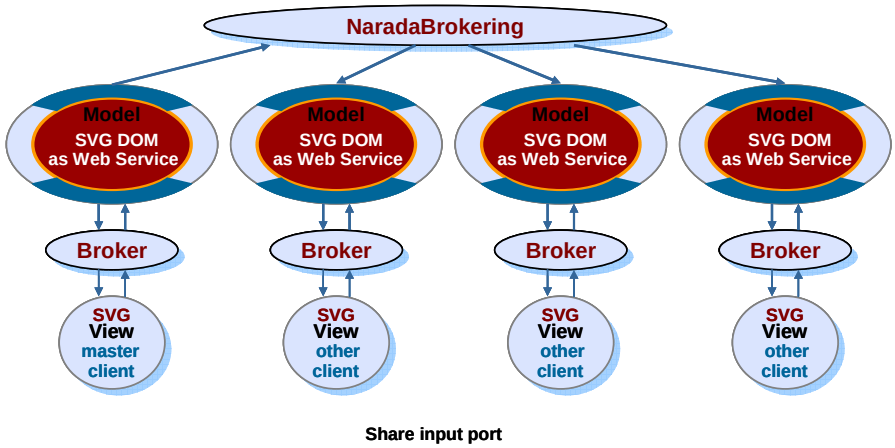

Figure 6 MMMV collaborative Web Service

- Building on top of the collaboration framework, one can develop SVG applications of instructor-led (SMMV) and participatory (MMMV) programming models with Java and JavaScript. One can expect this approach be applied to other presentation style application and programming languages, and we have in our laboratory other initiatives on OpenOffice and PowerPoint.

3.4 Monolithic collaboration and Web Service collaboration

In this paper, we discuss two ways of building an event-based collaboration system: monolithic and Web service.

Monolithic collaboration (see figure 4), is obtained when all participating components are formed as replications of an existing application without explicit break up into a separate *model* and *view* component as required by the Web service architecture. This approach works through interception of the events on a master application and allows messaging broker to multicast them to the collaborating clients. It is a common strategy for collaboration systems built on top of vendor's APIs with event exposure with either proprietary or open source implementations. We have demonstrated this in our laboratory with PowerPoint and OpenOffice [24].

We have already described the idea of "*Collaboration as a Web Service*" although at that time we demonstrated collaboration features through monolithic SVG experiments [11]. We presented already preliminary result on a collaborative SVG browser and JavaScript multiplayer chess game as a prototype to explore a general approach of collaborative Web Services. We then separated SVG into *model* and *view* components [12] and convert the desktop SVG application into a distributed system. In this paper, we are rebuilding the collaboration environment with explicit Web Service models and demonstrate both SMMV and MMMV models.

4. PERFORMANCE MEASUREMENTS

4.1 Test Scenarios

We have presented a unified collaborative architecture M-MVC, which supports the above variety of SVG applications. The advantages of this modular design imply performance overheads coming from the replacement of method calls by explicit messages. We have performed extensive performance tests [13] [25] with encouraging results but we still need further optimization to minimize thread scheduling and transport overheads.

To identify key factors that influence the performance of M-MVC in particular and message-based model in general of building distributed applications, we adopt the decomposition strategy of the Model and the View of Batik SVG browser as delineated in figure 1. The collaboration interactions between decoupled MMMV Model-Model or SMMV Model-View components are done through intermediary event brokers with publish/subscribe or point-to-point interface that is provided by NaradaBrokering [4] as messaging services (see figures 5 and 6). The "*View*" including client interface components (Swing GUI and GVT rendering) is dynamically downloaded to client. The "*Model*" consisting of DOM and JavaScript modules naturally becomes a service which could run standalone or on a Web server. Event-oriented messages, which are transported through our messaging infrastructure - NaradaBrokering, play the role of the "*Controller*".

There are many variables that we can vary in our tests including the locations 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. To simplify the issues, here we present some investigations with Broker, Model and View in the single Model and View case as displayed in figure 1.

We list scenarios for a set of performance tests: environment settings in table 1 and system configurations in table 2. Each test case presents a choice of combinations based on network, operating system, and CPU configurations. The coupling of the Model and the View components varies when Broker distance changed from direct switch connection to a remote site in campus or inter-city area. In test 1 to 6, the Broker either shared the same runtime environment or ran in a distinct operating system platform in communication with Model and View, where both of these were run with Windows on desktop computers. Hosting computers of Broker and View were varied to delineate the influence of CPU processing power.

Table 1 Specification of Test Scenarios

Test scenarios		**Environment Settings**					
No	**Description**	**Event Broker** (NB0.97 Server)	**View** (Client)	**Model** (Service)	**Network connection type**	**Broker distance**	
						area	hop
1	Switch connection	desktop2	desktop1	desktop2	direct switch	10 meters	1
2	Switch connection	desktop3 (High-end desktop)	desktop3	desktop2	direct switch	10 meters	1
3	Office area	linux (gridfarm1)	desktop1	desktop2	hub	10 meters	1
4	Within-City (Campus area)	linux HPC cluster node	desktop1	desktop2	routers	40 miles	n/a
5	Inter-City	Solaris (ripvanwinkle) (light loaded)	desktop1	desktop2	routers	100 miles	n/a
6	Inter-City	Solaris (complexity) (heavy loaded)	desktop1	desktop2	routers	100 miles	n/a

Table 2 System configurations used in Table 1

Computer		Hardware				Software
No.	**Type**	**Brand**	**Processor**	**CPU (MHz)**	**RAM**	**OS**
1	desktop	Dell Dimension 8100	Intel Pentium 4	1500	523,344KB	Windows 2000
2	desktop	Dell Dimension 8100	Intel Pentium 4	1500	512MB	Windows XP
3	desktop (highend)	Dell Dimension XPS	Intel Pentium 4	2990	1GB	Windows XP
4	Solaris (grids/community)	SUN Ultra-60	UltraSPARC II	450	1GB	Solaris 5.8
5	Solaris (ripvanwinkle/complexity)	SUNW, Sun-Fire-880	UltraSPARC III	900	16GB	Solaris 5.9
6	Linux (gridfarm1)	Angstrom, Phython	Intel Xeon	2400	2GB	Linux 2.4
7	Linux cluster (supercomputer node)	IBM	470 processors	1.1 Teraflops	0.5 TB	Linux 2.4 SMP

4.2 Timing model

A Graphics User Interface (GUI) provides the conventional computer-based interactive style applications for visual evoked responses. A complete pass of system behavior is started with user input in the *View*, event interpretation and process in the *Model*, and ended with re-display in the *View* corresponding to the system state change. At a high level, three parts contribute to the major cost of performance — computation at *View* and *Model*, and interaction between them. Performance is sensitive both to the nature of the application and the coding style and system architecture used. Further even with the same application, one will often find different results reflecting background loads in system and the nature of the user interaction.

The procedure of interactions between user and SVG applications is illustrated in figure 1. This shows the "U" turn trip along the pipeline delineates two legs of event propagation: one from input device to Broker and then the *Model*; the other from the updated DOM model via Broker to GVT tree and output of image rendering. Each stage is comprised of a component with different runtime states based on its function during the event process. Note that the communication between the *View* and the *Model* is routing of event-based messages via Broker over the network while the inter-stage interaction within *View* or *Model* component is done by runtime method call.

We found that the system performance is mainly composed of the latency at client (GUI and GVT for locating event and graphical rendering), service (application JavaScript code manipulates DOM elements), and messaging (event processing, buffering, and routing). We add a timer along the pipeline (ref. figure 1) at each of the marked timing points T0, T1, T2, T3, and T4 to scrutinize the cost and characteristics of the modules delineated by these timers. The times T0, T1, T2, T3, and T4 are all measured in the *View* and defined as follows:

T0: start time

T1: A given user event such as a mouse click can generate multiple associated DOM change events transmitted from the Model to the View. T1 is the arrival time at the *View* of the first of these.

T2: 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

T3: This is the start of the rendering stage

T4: This is the end of the rendering stage

As performance measurement is directly related to the choice of event for testing, we choose to track down system interactions within the testing model to the smallest atomic unit — mouse event — that is triggered by detection of each pixel change for performance measurement purpose.

4.3 Performance measurement and analysis

Tables 3 contain a selection of measured data, which records times between the processing markers T0, T1, and T4. More extensive results including a discussion of timing points T2 and T3 will be found in [25]. Figure 7 gives a detailed histogram extending the results of table 3. Each row of the table corresponds to averages over many event processing sequences i.e. to averages over processing of mouse events with understanding that for efficiency strings of mouse move events (generated by the system as each pixel is passed) are passed as single vector events. Note from the figure that events start on the *View* as a User Interface Mouse action and the pipeline sends them through the *Model* and back to the *View*.

We used the same JavaScript chess program described in earlier papers [11] in tables 3. All events are W3C DOM compliant as required by the SVG application. T0 represents the time that messages are transmitted from View to Model after initial processing in View of mouse event. T1, recorded in the View, represents the time that the associated events are returned from the Model to the View. A given user interface event generates several Model events which are sent back to the View as separate messages and we record in tables 3 the times of the first and last messages in this returned sequence. The final time recorded T4 corresponds to the end of the rendering

update in the View component. All times are recorded relative to the processing marker T0. We record mean, statistical error in the mean and standard deviation of the distribution. Essentially all plots show broad distributions with large standard deviations.

Table 3 Average performance

Test	Mousedown events		Average of all mouse events (mousedown, mousemove, and mouseup)					
	First return – Send time: T_1-T_0 (milliseconds)		**First return – Send time:** T_1-T_0 (milliseconds)		**Last return – Send time:** T'_1-T_0 (milliseconds)		**End Rendering** T_4-T_0 (microseconds)	
No	**mean ± error**	**Stddev**	**mean ± error**	**stddev**	**mean ± error**	**stddev**	**mean ± error**	**stddev**
1	33.6 ± 3.0	14.8	37.9 ± 2.1	18.7	48.9± 2.7	23.7	294.0± 20.0	173.0
2	18.0 ± 0.57	2.8	18.9 ± 0.89	9.07	31.0 ± 1.7	17.6	123.0 ± 8.9	91.2
3	14.9 ± 0.65	2.8	21.0 ± 1.3	10.2	43.9 ± 2.6	20.5	414.0 ± 24.0	185.0
4	20.0 ± 1.1	4.8	29.7 ± 1.5	13.6	49.5 ± 3.0	26.3	334.0 ± 22.0	194.0
5	17.0 ± 0.91	4.3	24.8 ± 1.6	12.8	48.4 ± 3.0	23.3	404.0 ± 20.0	160.0
6	20.0 ± 1.3	6.4	29.6 ± 1.7	15.3	50.5 ± 3.4	26.0	337.0 ± 22.0	189.0

In table 3, we record the difference between types of mouse events by recording both all mouse down processing sequences and the results averaged over mouse move, mouse down and mouse up. The measurements in the first two columns are an upper limit on the overhead due to the decomposition and this varies from 20-40 ms with most measurements at the lower end of this range. This holds for all broker positions from collocation in the desktop to remote location (in Indianapolis with the Clients in Bloomington). We call this an upper limit as it is processed concurrently with essential computation (the thread scheduling issue) and we get some improvement in M-MVC due to concurrent processing between Model and View for operations sequentialized in the conventional version.

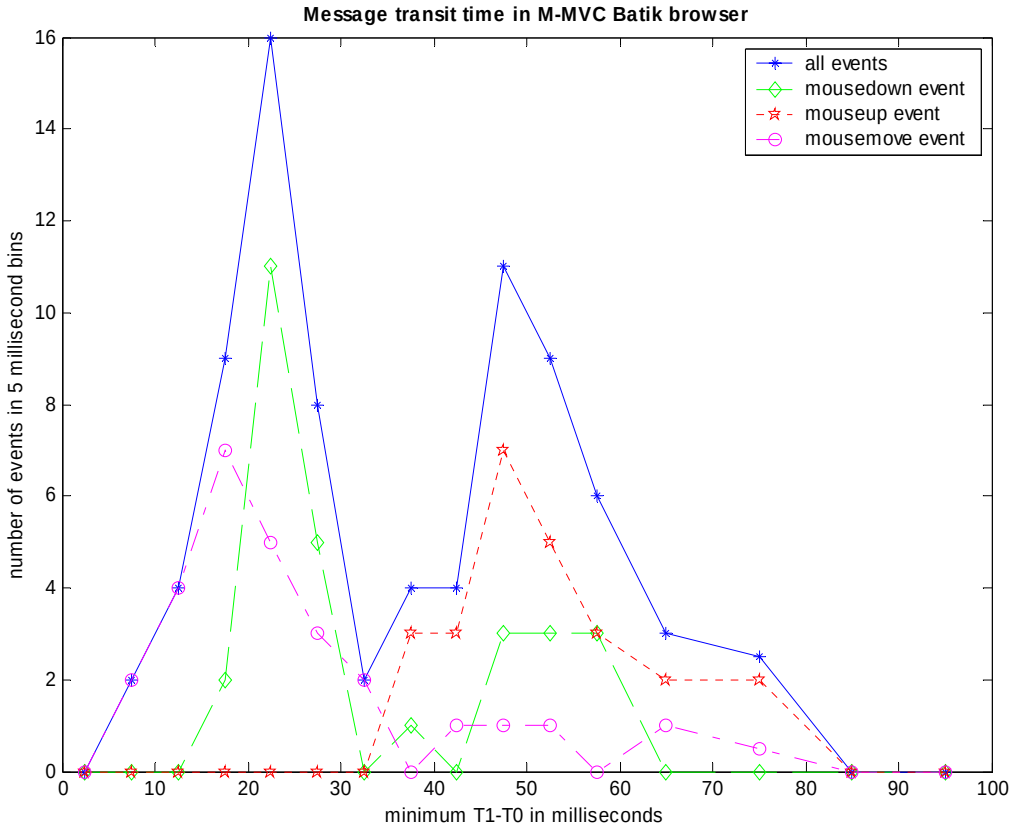

Figure 7 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 1 of Table 1 and the row labeled 1 in table 3. The configuration is in detail: NB on Model; Model and View on two desktop PCs; local switch network connection. A few events with timing greater than 100 milliseconds are not shown on the plot

Figure 7 shows for the three major mouse event types (up, down, move) rather clear peaks with widths at half height of about 10-15 milliseconds. Mouse move shows the lowest and mouse up the largest means but the shapes are comparable.

It is known that human visual system (retina and brain) retains an image for a fraction of a second after it views the image, which is essential to all visual display technologies. For computer-based GUI, it implies that time delay of each system change including visual analysis and graphical feedback must be lower than the 30 millisecond time frame to achieve coherent view — with prompt image update and no flickering. Of course often a complex model change can take longer than this to process and render. It is especially challenging for the design of distributed media rich applications. This is due to compute intensive graphics image processing and rendering plus extra network latency overhead. However, messages that containing representation data and control instructions must be delivered and processed before rendering can begin and this can lead to possible bottlenecks of overall system performance. To achieve proper functioning and real time experience, interactive style applications usually employ optimizations for improved system performance. The Batik SVG browser itself buffers changes to exploit visual persistence and it only updates the rendering every 20 milliseconds.

We have discussed network performance in this subsection and shown that it impacts the M-MVC application significantly as it can add 100's of milliseconds to the user interaction. However we intend M-MVC to be used in the local environment where our results show good performance even when components are separated by about 40 miles corresponding to the connection between the Bloomington and Indianapolis campuses of Indiana University with a very good network link. Note collaboration applications MMMV and SMMV are not so sensitive to network latency as the events are pipelined and non-masters follow master events. Here one is sensitive to the acceptable delay in round-trip audio for interactive conversations. As discussed in Uyar's thesis [26], this is an order of magnitude longer than the delay associated with visual persistence.

If the Web Service *model* and *view* are placed on nearby machines with the message broker on one of these computers, we get an overhead of about 10 milliseconds in the transport from *view* (user input) to broker to *model* and back again for rendering at the *view*. This overhead is the same for both the collaborative and standalone cases.

5. CONCLUSIONS

The SMMV collaboration model can be used for lecturing in distance education; the MMMV collaboration model would support participatory learning. Education requires different mode of interaction ranging from rather passive fashion lecturing to highly interactive and collaborative in participatory learning such as joint projects. A whiteboard represents a good example for interactive project-based learning. It allows multiple people to participate interactively together. Joint modeling projects have the same structure as the whiteboard although using different detailed tools.

We have proposed M-MVC, a universal modular design with messaging linkage service model that unifies support of desktop applications, Web applications, and Internet collaboration. This approach allows maximum reusability of existing components; use of a flexible messaging scheme with high scalability; automatic and effective collaboration with interactivity of rich Web content for diverse clients over heterogeneous network environments; finally it suggests a uniform interface for the next generation Web client with ubiquitous accessibility. Applied to education, our architecture enables new participatory education tools and a richer distance education environment.

The architecture presented here is being used in our laboratory in several related projects looking at collaborative desktop and visualization tools and together these could enhance the e-education environment.

6. REFERENCES

[1] Global Internet Statistic http://www.glreach.com/globstats/index.php3

[2] W3C Web Service Description Language at http://www.w3.org/TR/wsdl

[3] G. Lee, **Object oriented GUI application development,** Prentice Hall, 1994. ISBN: 0-13-363086-2.

[4] Community Grids Lab NaradaBrokering system at http://www.naradabrokering.org

[5] Sun Microsystems Java Message Service at http://www.hostj2ee.com/specs/jms1_0_2-spec.pdf

[6] Sun Microsystems JXTA at http://www.jxta.org/

[7] Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara, "*A Web Services Framework for Collaboration and Videoconferencing"*, WACE 2003 Workshop on Advanced Collaborative Environments Seattle June 22 2003 http://grids.ucs.indiana.edu/ptliupages/publications/wace-submissionjune-03.pdf

[8] Community Grids Lab Carousel project at http://grids.ucs.indiana.edu/ptliupages/projects/carousel/

[9] Anabas Conferencing system http://www.anabas.com

[10] Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox, "*Internet Collaboration using the W3C*

Document Object Model", 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

[11] Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox, *"Collaborative SVG as A Web Service"*, Proceedings of SVG Open, Vancouver, Canada, July 2003 http://www.svgopen.org/2003/papers/CollaborativeSVGasAWebService/#S.Bibliography (requires SVG viewer plug-in http://www.adobe.com/svg/viewer/install/main.html for displaying figures)

[12] Xiaohong Qiu, "*Building Desktop Applications with Web Service in a Message-based MVC Paradigm"*, IEEE 2nd International Conference on Web Services (ICWS 2004) pages 765-769, San Diego July 2004 http://grids.ucs.indiana.edu/ptliupages/publications/ICWS04_BuildingDesktopApplicaitonwithWebServicesinaMessageBasedMVCParadigm.pdf or http://dx.doi.org/10.1109/ICWS.2004.1314812

[13] Xiaohong Qiu, Shrideep Pallickara, and Ahmet Uyar, "*Making SVG a Web Service in a Message-based MVC Architecture",* in Proceedings of SVG Open Conference, September 2004, Tokyo, Japan. http://www.svgopen.org/2004/papers/MakingSVGaWebServiceinaMessageBasedMVCArchitecture/

[14] W3C Scalable Vector Graphics (SVG) version 1.0 Specification http://www.w3.org/TR/SVG/.

[15] Apache Batik SVG Toolkit http://xml.apache.org/batik/

[16] W3C Document Object Model (DOM) level 1 specification http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

[17] Flynn's taxonomy is a classification of computer architectures based on the number of streams of instructions and data. It is proposed by Flynn in 1972. http://en.wikipedia.org/wiki/Flynn%27s_taxonomy

[18] Single Instruction Multiple Data (SIMD) at http://en.wikipedia.org/wiki/SIMD

[19] Multiple Instruction Multiple Data (MIMD) at http://en.wikipedia.org/wiki/MIMD

[20] 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

[21] Sun Microsystems, Java Server Faces Technology, Sun Microsystems. http://java.sun.com/j2ee/javaserverfaces/overview.html

[22] Java Server Pages JSP for producing Java based dynamic web content http://java.sun.com/products/jsp/

[23] Sun Microsystems, Java Servlet Technology, Sun Microsystems, http://java.sun.com/products/servlet/index.jsp. Servlets allow dynamic updating of Java Servers.

[24] 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

[25] Xiaohong Qiu. *Message-based MVC Architecture for Distributed and Desktop Applications*. Ph.D. thesis. EECS Department of Syracuse University. Spring 2005. http://grids.ucs.indiana.edu/~xqiu/thesis.html

[26] Ahmet Uyar. *Scalable Grid Architecture for Video/Audio Conferencing*. Ph.D. thesis. EECS Department of Syracuse University. Spring 2005.