Making SVG a Web Service in a Message-based MVC Architecture

Xiaohong Qiu[1, 2], Shrideep Pallickara[2], and Ahmet Uyar[1, 2]

[1]EECS Department, Syracuse University
[2]Community Grids Lab, Indiana University

Keywords
SVG, Web service, message, MVC, and publish/subscribe

Abstract
We reformulate Scalable Vector Graphics browser in a Web Service architecture separating the rendering from the W3C DOM processing of events. We describe this in a message-based Model-View-Controller (M-MVC) architecture and implement it with a powerful publish-subscribe messaging infrastructure. A Web Services oriented architecture with services loosely coupled by the exchange of messages is becoming an increasingly important feature in the deployment of Internet applications. The broad applicability of this approach includes enterprise software, e-Learning, e-Science and e-Business. Our work provides a general framework for integrating Desktop and Web Service applications. We summarize the performance results from detailed tests of our prototype. These measurements demonstrate the viability of our approach and identify some key issues influencing the performance of message-based Web and Desktop applications. We note how our architecture elegantly supports the major paradigms for collaboration.

1. Introduction

In this paper, we discuss the conversion of Batik SVG browser [BATIK], a desktop application, into a distributed system. Specifically, it includes decomposition of Batik SVG browser into separate "*View*" and "*Model*" components; modification of its architecture from traditional method-based MVC [MVC] into message-based MVC. The "*View*" including client interface components (Swing GUI and GVT rendering) is dynamically downloaded to client. The "*Model*" consisting of DOM and JavaScript modules (see fig. 4) naturally becomes a Web Service running on a Web server. Event-based messages, which communicate through our messaging infrastructure ─ NaradaBrokering [NaradaBrokering], play the role of the "*Controller*".

In the following sections, we first give a brief introduction to the state-of-the-art technology in areas of message-based (Web) service oriented technologies. Under the general background, we provide a roadmap towards our research effort of M-MVC as a generic solution in design space for building distributed applications. A summary of several variations of MVC applications follows by a detailed discussion in the SVG experiments. We give elaborate performance tests and analysis corresponding results to quantify the correlations between system behavior (semantic and performance), user interaction pattern (typical mouse event), and environmental factors (settings of clients and messaging broker, operating system, and network). Conclusions include the lessons we've learnt from these experiments.

2. Background

Rapid growth of network and Internet technologies has brought fundamental changes in the new generation of computer technology. Particularly, continuous improvement of computer CPU speed [MOORE] and network bandwidth [GILDER] enables design and implementation of new software architecture with satisfactory performance that allows development of many capabilities previously impossible. On the other hand, the latest deployment of Web Services [WEBSERVICE] and service oriented architecture (SOA) [SOA] with loosely coupled messages is expected to replace traditional client/server and distributed system models such as CORBA and provides a more general and dynamic framework that supports scalability and interoperability among distributed software assets.

From a technical perspective, there are some distinctive features. Traditional distributed object model employs exchanging coupled-messages through distributed version of method calls and returns, such as those in RPC-based and RMI-based platforms. Message-based approach produces lightweight loosely coupled services supporting asynchronous messages linkage (e.g. one-way transmission from sender to receiver). The messages are targeted but not directly coupled, which enable software level routing mechanism to provide platform independent communication paradigms (e.g. publish/subscribe) with excellent scalability. Further more, XML-based interface and specifications such as Simple Object Access Protocol (SOAP) [SOAP] and Web Services Description Language (WSDL) [WSDL] provide a generic interoperable platform among heterogeneous systems, which increase interoperability and reusability of existing software components.

In architectural view, a virtual distributed operating system is forming as an intermediary layer over the conventional bit-level Internet infrastructure (physical network and protocols such as IP, TCP UDP, HTTP, and SSH). One current effort focuses on building of messaging infrastructure that provides assurance of communication services (reliability, QoS, security, firewall tunneling, event notification, publish/subscribe, overlay, and peer-to-peer) tailoring for the support of diverse applications. The separation of top level application architecture from underlying messaging infrastructure simplifies the deployment overhead of applications and significantly increases application portability.

The overall innovation and advancement in computer technologies provides a great opportunity and foundation for deploying sophisticated distributed applications (e.g. Internet collaboration enabling virtual enterprises and large-scale distributed computing). Over the decade, the architecture of network-based applications keeps evolving ─ from earlier client/server, to multi-tier, middleware, peer-to-peer and overlay models. There're also many systems provide framework and standard APIs to address interoperable relationship between client user interface (GUI) and server side application behavior. Typical examples are JSP [JSP] for J2EE (or similarly ASP for .Net), JSR-168 [JSR168] and WSRP [WSRP], and REST [REST]. Each example addresses issues in targeted problem scope. However, one still needs a paradigm with a highly flexible architecture adapting to fast changes and requirements in real world. This motivates us to look into some intrinsic design concepts of client system (MVC [MVC]), parallel system (messaging [MPI]), distributed system (Web Services [WEBSERVICE]), and Internet collaboration (double-linked multiple-stage pipeline model [FOX03]). We pursue a generalization of the existing models aimed at simplicity of building applications with following properties:

- separation of application architecture from underlying messaging infrastructure for generality (independent of specific platform, programming language, and network protocols) and portability
- proposing message-based MVC (M-MVC) approach to address the problem of traditional tightly coupled *model*, *view*, and *controller* classes for scalability and universality
- extending M-MVC architecture to legacy desktop applications so as to have a uniform Web Services model with messaging linkage for reusability and interoperability
- providing a paradigm with automatic collaboration and universal access (including thin client interface such as PDA and cellular phone) capabilities

As our approach is based on investigation of MVC paradigm and message-based Web services, which are fundamental design models from desktop to distributed applications, deployment of a uniform architecture for desktop and distributed applications with automatic collaboration capability has general importance and we have detailed discussions of the design principles in another paper [QIU-09-07-04]. In this paper, we will provide our solutions to the following questions with focus on SVG implementation:

- Can MVC be implemented in a message-based fashion?
- What principles are there to govern the decomposition of a given application into MVC components?
- What is the performance of the message-based MVC and what factors influence it?
- How does it depend on the operating system, the application, machines and network?
- What is the relationship of collaboration and Web services with MVC paradigm?
- How easy is it to covert an existing application to message-based MVC?
- What are the architectural and implementation principles to be used in building applications from scratch in a message-based MVC paradigm?

3. Overview of MVC approaches

3.1 Variants of MVC

The concept of Model-View-Controller (MVC) [MVC] initially appeared openly in Smalltalk-80 [SMALLTALK]. It inherited from object-oriented programming idea of Simula 67 [SIMULA67] with integration of graphical user interfaces and interactive program execution. MVC proposed the logical separation of presentation from behavior and data structure in an interactive multiple windows programming environment with the triad of *Model*, *View*, and *Controller* components. Classic MVC paradigm is frequently used in almost all modern desktop architecture design and is popular in interactive applications. As a design paradigm, MVC is nothing new in the object-oriented programming world. However, it is the realization ─ the MVC pattern is particularly well-suited to addressing many of the fundamental problems inherent in building Web-based or distributed applications that rejuvenates the MVC concept.

There're many ways to classify MVC approaches according to the properties of decomposition strategies, interactive pattern, and communication mechanism. We present three examples to illustrate how interoperable relationship between the *model* and the *view* components impacts an application's architecture. The approaches depicted in fig. 1 are: a) classic method-based model; b) request/response model in method-based or message-based style; c) message-based publish/subscribe model. Note that the *controller* can be implemented as a separate class, combined with model and/or view components, or contained in messages in the scenarios. The sub-graphics of a, b, and c delineate a trend of system design from tight coupling to loose coupling, which fairly reflects the trace of evolution of standalone desktop application (single-user environment), client/server Web application, and distributed application with group communication enhancement. A more detailed summarization with each category and corresponding applications is shown in table 1.

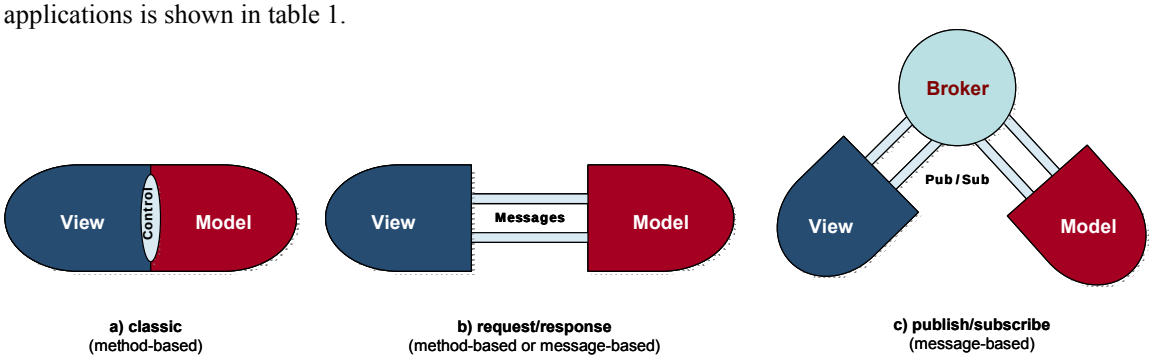

Figure 1 Three MVC approaches based on different communication mechanism and interactive pattern between *model* and *view*

In the communication column, "method-based" and "message-based" mechanism defines the interaction interface: either through a coupled pair of method call and return or uncoupled messages. Accordingly, the degree of coupling is indicated by this feature. However, in terms of timing, a typical runtime method call (e.g. Java) in a standalone single processor environment is at microsecond level while millisecond and 100 milliseconds are typically found for the communication in intranet and internet scope [FOX04]. As these different timescales imply different fundamental building ground for application architecture and viability, we give in depth analysis of performance through our experiments with batik SVG browser [BATIK], which converts a desktop application to a distributed system.

The communication patterns refer to the three models depicted in fig. 1. Interactive pattern describes the interoperating relationship between the *model* and the *view*: one-to-one, one-to-many, and many-to-many. The power of MVC modularity is enabling component reusability and we propose Single Model Multiple View (SMMV) and Multiple Model Multiple View (MMVC) as the models corresponding to the latter two patterns. SMMV is widely used in client/server Web applications with multiple clients (through Web

browsers) accessing a server, as well as legacy interactive applications (e.g. Microsoft Windows and Office) with multiples window layouts sharing the same data structure. The problem of SMMV is that it does not provide direct support for universal access (for clients of heterogeneous platform or interface); rather it requires extra customization for the *view*. For example, to make a SVG browser accessible for both Windows and cellular phone clients with consistent semantics and visual output, an adaptor class is needed for the thin client rendering. MMMV is a generalization of SMMV, which enables ubiquity with the customization done from the *model* at server side. JavaServer Faces (JSF) [JSF], which extends JavaServer Pages (JSP) [JSP] and Java Servlet [SERVLET] technology, 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 but 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. For further discussion, we provided details of participatory learning vs. instructor-led learning as collaborative Web Service models based on MMVC vs. SMMV [QIU-09-07-04], which extend the idea of "shared input port" vs. "shared output port" collaboration framework [FOX03].

Web Services for Remote Portlets (WSRP) [WSRP] is a communication protocol between portal servers and backend portlet containers, while Java Specification Request (JSR) 168 [JSR168] is a Java API for portlets to work with WSRP portals. These two standards enable aggregation of portlets so that different portal products are available to an organization, typically through a Web browser at client tier. JSR-168 and WSRP are in orthogonal direction in architecture space and they can be implemented in either method-based or message-based manner. However, they define the nature of the messaging for message-based MVC, which produces an important technology in support of Web Service applications.

Representational State Transfer (REST) [REST] proposed a simplified version of message-based approach that extended from client/server Web application architecture. M-MVC and REST both are message-based architecture. The distinctions are: a) REST addresses scalability, reliability, tunneling through firewall and security (SSL) issues within the containing system; M-MVC assumes that application level architecture is separated from underlying messaging infrastructure and the latter provides various communication services (e.g. QoS, fault-tolerance, event notification, and publish/subscribe). b) REST is suitable for less time critical collaboration through sharing of application state over HTTP protocol; M-MVC support both asynchronous and synchronous collaboration through sharing of event (the change of application state) and allows dynamic binding to transportation protocols. For the timescales of synchronous collaboration, the affordable latency for an audio/video conferencing system (over UDP) is 200 milliseconds with client buffering and pre-fetching and 20 milliseconds for SVG Web Services experiment (over TCP) of this paper with vector events and combined rendering optimization. c) REST is designed for Web application; M-MVC is proposed as a uniform architecture for both client and distributed application. d) REST is a SMMV model that uses request/response interactive interface; M-MVC can be deployed in either SMMV or MMMV with publish/subscribe scheme.

Table 1 Variants of MVC applications

	Application type		Degree of coupling	Communication					Interactive pattern	
				mechanism		pattern				
	client/ desktop	distributed		method based	message based	method call	request/ response	publish/ subscribe	**SMMV**	**MMMV**
Microsoft Office	√		+ +	√		√			√	
JSP/JSF		√	+		√		√		√	√
JSR-168 & WSRP		√	n/a	√	√		√		n/a	n/a
REST		√	−		√		√		√	
M-MVC	√	√	−		√			√	√	√

In summary, table 1 shows different MVC application examples that decrease in degree of coupling between model and view components ─ from client to distributed domain with method-based to message-based interoperation. At meanwhile, loosely coupled messages facilitate the overall system design with a

more distributed, scalable and interoperable communication mechanism, which enables a general framework over heterogeneous platforms. M-MVC is a high-level application architecture that converges desktop application and distributed application with automatic collaboration and universal access support. Web Service is naturally fitting in to M-MVC and we elaborate the composition in subsequent section.

3.3 Message-based MVC and Web Services

Web Services provide interfaces for service oriented architecture (SOA). Ultimately, the services would offer GUI to end users for access. Nevertheless, Web Services (or SOA) do not address system decomposition issue and application developers have to determine which component should reside in the service vs. client interface. Instead of making general remarks based on component functions (e.g. business logic and query for database belong to service), here we illustrate a systematic approach with a Message-based MVC architecture.

- M-MVC is a SOA that decomposes a system into the *model* ("computation core") and the *View* (visual component) with messaging linkage. The model component naturally becomes the "service" while the view component represents client interface.
- M-MVC employs a double-linked multiple-stage pipeline model that refines MVC partition into small grained stages with messages exchanging between the neighbor stages in both directions. This structure has following advantages: a) the uniform stages and pipeline communication behavior with input and output interfaces forms a regular modularized structure. Theoretically, this pattern can be applied to decomposition at any part within the system embracing natural event linkages and produces multiple coordinated objects in a single application. Each stage or object, a primary distributed component, forms the core of a Web Service. b) This modular multiple-stage approach facilitates the system process being controlled in a fine grained fashion for distribution, which is impossible in a canonical two-tier client-server model for Web applications and MVC model for desktop applications. c) Each stage along the pipeline forms a synchronization point for collaboration. d) Bi-directional traversal between adjoining stages enables invertible changes of system state, which is an effective method for participatory components to reach a common stage. e) The messages, which contain event or rendering information, provide a uniformed format for flexible dissemination over diverse communication protocols and patterns (e.g. unicast and multicast)

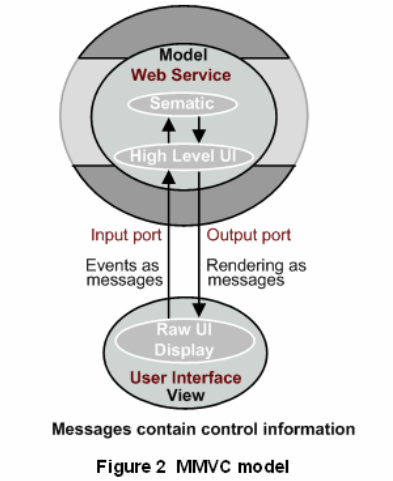

Figure 2 MMVC model

Among many decomposition possibilities, fig. 2 delineates M-MVC architecture being deployed in a three-stage pipeline model. A complete pass of an interactive process starts with an input event initiated by user input (e.g. a mouse click or a key stroke), interpretation and computation along the multiple stages, and ends with an output mostly consisting of text or graphics for re-display of the updated image buffer, each stage effectively is passed by twice during the procedure ─ one is along event propagation path; the other is on rendering approach. Mapping to the SVG browser experiment (see fig. 4), Raw UI events represent mouse (or key) events while High Level UI and Semantic events imply DOM and application events. Note

that decoupled messages are exchanged via event brokers of our underlying messaging infrastructure, NaradaBorkering [NARADABROKERING], in a publish/subscribe scheme. We've elaborated this mechanism in another paper for two collaborative patterns based on SMMV and MMMV model [QIU-23-07-04].

The Web Services composition of M-MVC is further depicted in fig. 3, which embracing three elements: NaradaBrokering (NB) that provides communication services (e.g. HTTP, UDP, and TCP transportation protocols); SOAP (header, body, and encoding rules); and application (event messages). Normally, SOAP messages use text encoding (XML format) and are carried with HTTP protocol through port 80. However, the overhead of replicated information in each envelope and header, XML parsing, and HTTP protocol etc. added up can make this approach very inefficient. We use a high performance approach ─ namely, only keeping initial negotiation message (e.g. message 1) with XML format whilst encoding subsequent messages (message ***i***) with agreed "mapped SOAP" format (e.g. native format for serialized event object) through NB transports in a changed port. This can be achieved by special encoding rules with proper settings in SOAP header [HPSTREAMING]. Next release of NaradaBorkering will include implementation of this algorithm in support of high performance streaming for Web Services. Apart from performance gains, which particularly important for time critical applications, it allows a uniform interface for native transportation and Web Services conformation. Our performance testing with SVG experiments generate consistent results for both scenarios.

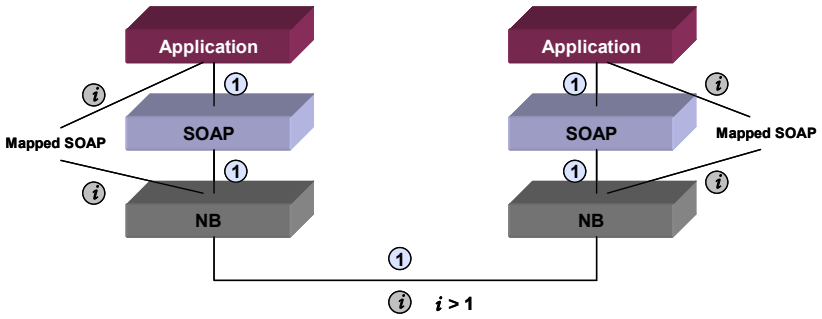

Figure 3 Web Services composition of M−MVC application, SOAP, and NB

3.4 Message-based MVC and SVG

Traditionally, desktop applications employ MVC paradigm in method-based interactions between the components to achieve high performance for interactive applications. Publish/subscribe scheme enables event-based programming to link event source component and event listeners' components asynchronously through callback methods while event messages are hidden at system level. This approach is widely used in object-oriented systems including Java AWT, Swing, and applications built on top of them such as Batik SVG browser. We propose a different approach of "*explicit message-based MVC*" paradigm (M-MVC) for applications deployment [QIU-23-06-03], which replaces hidden method-based events at Java run-time level by exposed messages. By doing so, the tightly coupled connections between different parts of an application are replaced by a loosely coupled messaging linkage service model with flexibility, distribution, and scaling advantages.

We have a complete analysis of constituent components and their interactive relationship for the Batik SVG browser. The logistic components can be decomposed into a three-stage pipeline, as illustrated in fig. 4. Theoretically, any parts with natural event linkage between client user interface and computation core can produce web services coordinated in a single application. We performed substantial experimentations to find the best decomposition point, which preserved system functionalities while avoiding excessive re-engineering of the software. We chose to split the SVG browser between the DOM and GVT tree, which allows generalization to other DOM applications.

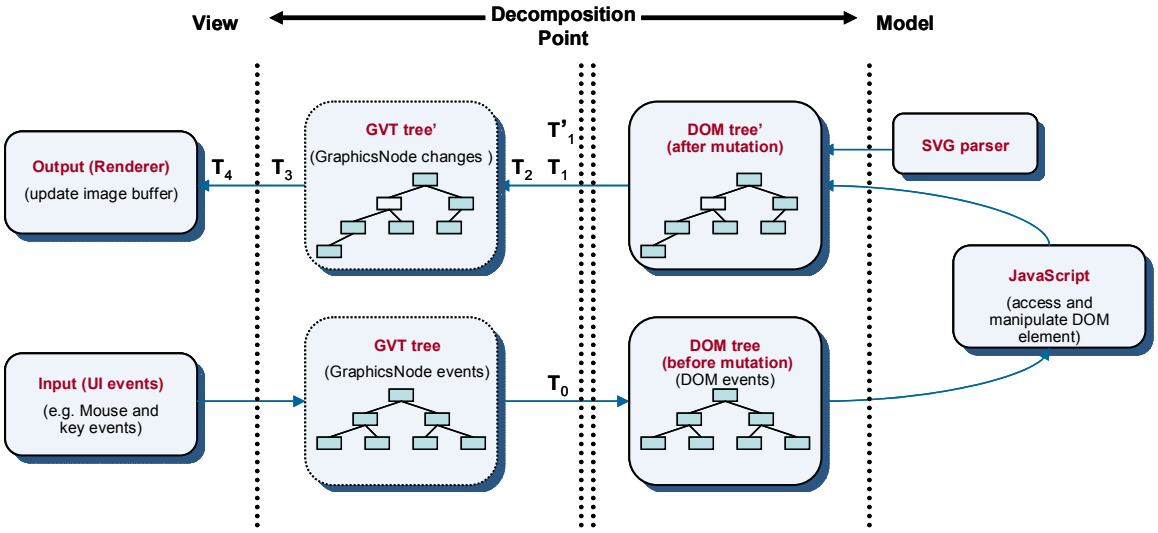

Figure 4 Decomposition of SVG browser in stages of pipeline

4. Performance

We have performed a series of performance measurements to test the effectiveness 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 list scenarios for a set of performance tests in this paper: system configurations in table 2 and testing environment settings in table 3. Tables 4 to 6 contain a selection of measured data while more extensive information including histograms can be found in [M-MVC].

Table 2 System configurations

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 (ripvanwinkle/complexity)	SUNW, Sun-Fire-880	UltraSPARC III	900	16GB	Solaris 5.9
5	Linux (gridfarm1)	Angstrom, Phython	Intel Xeon	2400	2GB	Linux 2.4
6	Linux cluster (supercomputer node)	IBM	470 processors	1.1 Teraflops	0.5 TB	Linux 2.4 SMP

Table 3 Testing environment settings

Test scenarios		Environment Settings					
No	**Description**	**Event Broker** (NB0.97 Server)	**View** (Client)	**Model** (Web Service)	**Network connection**	**Broker distance**	
						area	hop
1	Switch connects Desktop server	desktop2	desktop1	desktop2	switch	10 meters	1
2	Switch connects High-end Desktop server	desktop3	desktop3	desktop2	switch	10 meters	1
3	Office area Solaris server	solaris	desktop1	desktop2	hub	100 meters	1
4	Office area Linux server	linux	desktop1	desktop2	hub	100 meters	1
5	Campus area Linux cluster node server	linux cluster	desktop1	desktop2	routers	5 miles	n/a

6	Inter-city area Solaris server	solaris	desktop1	desktop2	routers	50 miles	n/a

The results tables 4 to 5 record times between the processing markers T_0, T_1, and T_4 shown in fig. 4 (times for other markers are given in [M-MVC]). 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*. In tables 4 and 5, we used the same JavaScript chess program described in earlier papers [SVGOPEN03]. All events are W3C DOM compliant as required by the SVG application. T_0 represents the time that messages are transmitted from *View* to *Model* after initial processing in View of mouse event. T_1, 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 4 and 5 the times of the first and last messages in this returned sequence. The final time recorded T_4 corresponds to the end of the rendering update in the *View* component. All times are recorded relative to the processing marker T_0. We record mean, statistical error in the mean and standard deviation of the distribution. Essentially all plots show broad distributions with large standard deviations.

In table 4, 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. Table 5 records times for a special bounce back event generated automatically for these runs by the *Model* component as soon as it receives a message from the *View*. Table 6 does not concern Batik and SVG at all. It records times for the *View* sending a message to NaradaBrokering and recording its return (2 hop events in table 6); the 4 hop events correspond to messages going from *View* location to NaradaBrokering to Model location and back. In all cases for table 6, a simple Java program generating events of the same structure as used in SVG was used. However this program did no further work on the message – only its communication. So this table 6 records the natural overhead from NaradaBrokering. This is about 2 milliseconds per event but is increased in some entries in table 6 and in the bounce-back event of table 5 by interference between communication and other active threads on the *Model* and *View* computers. This interference probably accounts for the broad distribution seen in essentially all results. We have studies of clean unloaded Linux and Windows machines documenting the 2 millisecond per hop NaradaBrokering natural overhead. Note configuration 2 includes the fastest client – desktop3 – and this impact is very clear in all the tables. It is worth noting that Moore's law helps M-MVC for increasing client performance will reduce the M-MVC overhead and the better results on desktop3 highlight this.

Table 4 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	48.90± 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	17.0 ± 0.91	4.3	24.8 ± 1.6	12.8	48.4 ± 3.0	23.3	404.0 ± 20.0	160.0
4	14.9 ± 0.65	2.8	21.0 ± 1.3	10.2	43.9 ± 2.6	20.5	414.0 ± 23.6	185.0
5	20.0 ± 1.1	4.8	29.7 ± 1.5	13.6	49.5 ± 3.0	26.3	333.8 ± 22.0	194.0
6	20.0 ± 1.3	6.4	29.6 ± 1.7	15.3	50.5 ± 3.4	25998.0	336.7 ± 22.0	189.0

Table 5 Immediate bouncing back event

Test	**Boucing back event**		**Average of all mouse events (mousedown, mousemove, and mouseup)**					
	Bounce back – Send time: (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** (milliseconds)	
No	**mean ± error**	**stddev**	**mean ± error**	**stddev**	**mean ± error**	**stddev**	**mean ± error**	**stddev**
1	36.8 ± 2.7	19.0	52.1 ± 2.8	19.4	68.0 ± 3.7	25.9	405.0 ± 23.0	159.0
2	20.6 ± 1.3	12.3	29.5 ± 1.5	13.8	49.5 ± 3.1	29.4	158.0 ± 12.0	109.0
3	24.3 ± 1.5	11.0	36.3 ± 1.9	14.2	54.2 ± 2.9	21.9	364.0 ± 22.0	166.0

4	15.4 ± 1.1	7.6	26.9 ± 1.6	11.6	46.7 ± 2.9	20.6	329.0 ± 25.0	179.0
5	18.1 ± 1.3	8.8	31.8 ± 2.2	14.5	54.6 ± 4.9	32.8	351.0 ± 27.0	179.0
6	21.7 ± 1.4	9.8	37.8 ± 2.7	19.3	55.6 ± 3.4	23.6	364.0 ± 25.0	176.0

Table 6 Basic NB performance in 2 hops and 4 hops

Test	**2 hops (View – Broker – View)**		**4 hops (View – Broker – Model – Broker – View)**	
	milliseconds		milliseconds	
No	**mean ± error**	**stddev**	**mean ± error**	**stddev**
1	7.65 ± 0.61	3.78	13.4 ± 0.98	6.07
2	4.46 ± 0.41	2.53	11.4 ± 0.66	4.09
3	9.16 ± 0.60	3.69	16.9 ± 0.79	4.85
4	7.89 ± 0.61	3.76	14.1 ± 1.1	6.95
5	7.96 ± 0.60	3.68	14.0 ± 0.74	4.54
6	7.96 ± 0.60	3.67	16.8 ± 0.72	4.47

Note that much of the time delay from *Model* to *View* comes from waiting for a CPU that has been scheduled to a different (from the communication) Batik thread. For example comparing the first two rows of tables 5 and 6 (Bounce back time versus 4 hops), the two tables are measuring the same computation and communication time but table 5 is 10-20 milliseconds longer than table 6. This can be explained by the large (extraneous to message passing) computations on the *Model* and *View* in table 5 which delay the processing of messages which increases both the mean and the standard deviation – as this delay in scheduling the communication thread has a large variability.

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. The difference between column 1 and column 3 of table 5 measures the 30 ms typically spent on *Model* processing; this is an underestimate as it does not include the scheduling delay discussed above – an overestimate is gotten by replacing column 1 numbers from table 5 with the 4 hop measurements of table 6. Comparing columns 1 and 2 of table 4 shows that mouse down events are processed quicker than average – that is because most of chess application processing used in the *Model* occurs for Mouse up events. Comparing columns 1 and 2 of table 5 shows the 10-15 ms processing needed on the *Model* before any events are generated in response to a given mouse event received from the *View*.

In summary, these early results show the main issues to be the algorithmic effect of breaking the code into two, the network and broker overhead, and thread scheduling interference of operating system between interfaces of SVG application and messaging brokers. Our initial tests show the client to server and back transit time is only 20% of the total processing time in the scenarios where the message broker is local. Note that the Batik SVG Browser already uses a 20 ms buffer in its rendering engine to collect all updates occurring in time windows of this size; M-MVC adds a similar overhead. Little optimization has been attempted as the current results indicate that the processing overheads to be already acceptable. We will in the near future use Linux clients and study the large thread scheduling effects in more detail.

Conclusions

We've presented a uniform architecture with message-based MVC service model which unifies desktop and Web applications. Our experiments with SVG suggest that the structural change from traditional method-based MVC to message-based MVC is a viable approach that converts a tightly coupled system to a distributed system of SOA with loose messaging linkage. However, building applications centered on messages should provide consistent functionalities and acceptable performance for both design spaces. This approach requires the procedure of deployment to follow some principles: it is essential for the system to

have a strict modularized structure for the split (or distribution), an effective interactive model with sequential event processing design for synchronization, and serialization capability of streaming event messages for communication. The performance testing results show that the overall system performance is influenced by factors inherent from the logistics of the split of the code into two parts, messaging cost (e.g. network latency and overhead of event broker) and environment fluctuation (e.g. operating system and thread scheduling). Other research is undergoing in our laboratory in extension of these ideas to other presentation style applications including OpenOffice and PowerPoint using vendor APIs.

References

1) [BATIK] Apache Batik project of Scalable Vector Graphics at http://xml.apache.org/batik
2) [FOX03] Geoffrey Fox, Dennis Gannon, Sung-Hoon Ko, Sangmi Lee, Shrideep Pallickara, Marlon Pierce, Xiaohong Qiu, Xi Rao, Ahmet Uyar, Minjun Wang, and 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, Chicester, England, ISBN 0-470-85319-0, March 2003.
3) [FOX04] Geoffrey Fox, *The rule of the millisecond*, CISE magazine (http://www.computer.org/cise/), March/April 2004. Available at http://grids.ucs.indiana.edu/ptliupages/publications/cisejano4.pdf
4) [GILDER] Gilder's law is an assertion by George Gilder, visionary author of the book *Telecosm: The World After Bandwidth Abundance*, Free Press, 07 May, 2002, which states that "network bandwidth grows at least three times faster than computer power." http://www.netlingo.com/pocketdictionary.cfm?term=Gilder's%20Law
5) [HPSTREAMING] Geoffrey Fox, Harshawardhan Gadgil, Shrideep Pallickara, Marlon Pierce, Robert L. Grossman, Yunhong Gu, David Hanley, Xinwei Hong, *High Performance Data Streaming in Service Architecture*, technical report, July 2004. http://grids.ucs.indiana.edu/ptliupages/publications/HighPerfDataStreaming.pdf
6) [JSF] Sun Microsystems, *JavaServer Faces (JSF) Technology*, Sun Microsystems, http://java.sun.com/j2ee/javaserverfaces/overview.html
7) [JSP] Sun Microsystems, *JavaServer Paces (JSP) Technology*, Sun Microsystems, http://java.sun.com/products/jsp/overview.html
8) [JSPWHITEPAPER] Sun Microsystems, *JavaServer Page Technolgoy – White Paper*, Sun Microsystems, http://java.sun.com/products/jsp/whitepaper.html
9) [JSR168] Sun Microsystems, *Java Specification Requests: Portlet Specification*, Sun Microsystems, http://www.jcp.org/en/jsr/detail?id=168
10) [M-MVC] Xiaohong Qiu, *Message-based MVC Architecture for Distributed and Desktop Applications*, Ph.D. dissertation, Syracuse University, 2004.
11) [MOORE] Gordon E. Moore, co-founder of Intel Corporation, Moore's law on computer performance increasing exponentially in time is available at http://en.wikipedia.org/wiki/Moore's_Law. According to Moore's law, computer processing powers can double every 18 months.
12) [MPI] MPIF, *MPI: A Message Passing Interface Standard*, http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html
13) [MVC] G. Lee, *Object oriented GUI application development*. Prentice Hall, 1994. ISBN: 0-13-363086-2.
14) [NARADABROKERING] Open Source Messaging Internet System from the Community Grids Laboratory at http://www.naradabrokering.org
15) [QIU-23-06-03] 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
16) [QIU-09-07-04] Xiaohong Qiu, *Building Desktop Applications with Web Service in a Message-based MVC Paradigm*, IEEE International Conference on Web Services, San Diego, California, July 2004
17) [QIU-23-07-04] Xiaohong Qiu and Anumit Jooloor, *Web Services Architecture for e-Learning*, International Conference on Education and Information Systems: Technologies and Applications, Orlando, Florida, July 2004
18) [REST] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. dissertation, 2000. Representational State Transfer (REST) is the name of the

architectural style for distributed hypermedia systems of the thesis. Available at http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
19) [SERVLET] Sun Microsystems, *Java Servlet Technology*, Sun Microsystems, http://java.sun.com/products/servlet/index.jsp
20) [SIMULA67] Graham M. Birtwistle, Ole-Johan Dahl, Bjoern Myhrhaug, and Kristen Nygaard, *SIMULA BEGIN*, Studentlitteratur, Lund, Sweden, 1973. ISBN 91-44-06211-7. Simula67 was the first object-oriented programming language and a predecessor of Smalltalk and C++.
21) [SMALLTALK] A Goldberg. *Smalltalk-80: The Interactive Programming Environmen*. Addison Wesley, 1984.
22) [SOA] Service Oriented Architecture Service Oriented Architecture or SOA is a loosely coupled linkage of distributed software. See D. DeRoure, A. Dunlop, G. Fox, P. Henderson, A. Hey, N. Paton, S. Newhouse, S. Parastatidis, A. Tefethen, and P. Watson, "*Web Service Grids: an Evolutionary Approach*", UK e-Science Core Programme Directorate Position Paper, July 2004.
23) [SOAP] W3C, Simple Object Access Protocol (SOAP), W3C, http://www.w3.org/TR/soap/
24) [SVGOPEN03] 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
25) [WEBSERVICE] W3C, *Web Service Architecture*, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. According to W3C, Web services provide "a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks."
26) [WSDL] W3C, *Web Services Description Language (WSDL) 1.1*, W3C, March, 2001. http://www.w3.org/TR/wsdl
27) [WSRP] OASIS, *Web Services for Remote Portlets (WSRP)* is an OASIS standard for Portals to access and display portlets that are hosted on a remote server. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp