Message-based MVC Architecture for Distributed and Desktop Applications

By

Xiaohong Qiu

B.S. Beihang University, 1991

M.S. Syracuse University, 2000

DISSERTATION

Submitted in partial fulfillment of the requirements for the
degree of Doctoral of Philosophy in Computer Science
in the Graduate School of Syracuse University
May 2005

Approved _____

Professor Kishan Mehrotra

Date _____

Abstract

Services oriented architectures with loosely coupled messages are becoming an increasingly important feature in the deployment of Internet and Grid systems. Compared with traditional client/server and distributed system models such as CORBA, it provides a more general, dynamic, and flexible framework that defines distributed components and their interoperating relationship. The broad applicability of this approach includes enterprise software, e-Learning, e-Science, e-Business, and e-Entertainment. The key challenge is to exploit concept in design and implementation to provide scalable interoperable systems.

The design goal of this dissertation is developing of a paradigm for next generation of software applications with a clear and unified architecture that unifies desktop and Internet applications. It is aimed at addressing the issues of leveraging existing software assets and incorporating advanced capabilities including collaboration and universal access. As the overall Web systems design on top of the Internet is extremely complex, we divide the task into two separate layers: message-based distributed application architecture and underlying messaging infrastructure linking services together as part of distributed operating system.

This dissertation presents a new approach to building applications as Web Services in a message-based Model-View-Controller (M-MVC) architecture. The premise of this research is that distributed and Web applications which provide services and interface to

end users ought to be centered on message exchange. This encourages good design and is an embodiment of the fundamental communication pattern of human interactions. The research investigates a universal modular design with publish/subscribe messaging linkage service model that converge desktop applications, distributed applications, and Internet collaboration. This approach allows maximum reusability of existing components; flexible messaging scheme with high scalability; automatic and effective collaboration with interactivity of rich media Web content for diverse clients over heterogeneous network environments, and suggests a uniform interface for the next generation Web client with ubiquitous accessibility. We apply this architecture to the quite complex example of an SVG browser and give detailed performance measurements to demonstrate the viability of our approach.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

"*Everything should be made as simple as possible, but not simpler*."

— Albert Einstein [A. EINSTEIN]

Software architecture has always been a focal point of research for building computer-based systems. Architectural design decisions are commonly made based on a comprehensive evaluation and understanding of existing technologies and evolution, social requirements, application functionality and behavior, and vision of the future. Einstein's advice is still enlightening today and works well for building software systems. A good architecture not only comes from supplying a viable solution that accommodates to the latest developments and adapts to the needs of fast changing world, but also abides by the fundamental principles such as simplicity, reusability, scalability, portability, performance, and reliability.

Simple is never simplistic. Systems that are made too simple can not promise sufficient functionalities while those made too complicated tend to dramatically increase development and maintenance cost. Due to CPU and network bandwidth constraints, traditional software systems were built with closely coupled structure. As lacking of interoperability and reusability, individual system becomes increasingly complex.

As asserted by Alfred Chuang [A.CHUANG], "Application development represents the future … Integration is about the past." Bridging the gap of development and integration requires for a new method of linking today's idea to legacy software so that it

won't become "a roadblock to innovation, instead of a building block on which the future can be built."

While it's clear that Service Oriented Architecture (SOA) [SOA] and its current implementation ─ Web services [WEBSERVICE] technology will have profound impact on next generation of distributed applications by providing the interoperable platform that maximizes the reusability of existing software assets, many aspects of this platform and how to deploy service-enabled applications within this framework still require significant research and development.

At its core of the new trend, interoperability and convergence allow development of diverse loosely coupled applications that can be distributed and accessed in a synchronous or asynchronous manner, from any client, across the network. Taken individually, the imperatives such as reusability, interoperability, scalability, and ubiquity are not new. What is new, however, is the need to execute all of them at once with real time collaborative access experience and in aligns with the latest (Web) service oriented framework.

My work is motivated by understanding the nature of application design in terms of system composition, interoperation, and communication; evaluating its impact on performance; how differently these factors are deployed in corresponding application domains embracing desktop application, distributed application, and Internet collaboration; ultimately seeking a unified solution that closes the technology gap.

This dissertation proposes a message-based Model-View-Controller (M-MVC) architecture to building of service-enabled applications. This work looks into some intrinsic design concepts of desktop system (MVC [MVC]), event-based messaging

system (NARADABROKERING [NARADABROKERING]), distributed system (Web Services [WEBSERVICE]), and Internet collaboration (Web Service pipeline model [Fox03]). We pursue a generalization of the existing models targeted at simplicity of building message-based applications and offer a systematic approach that seamlessly unify distributed and desktop applications.

As a relatively novel approach, it supplies a universal modular design for next generation of software applications with an underlying messaging architecture that links services together. It emphasizes a message centric approach of building distributed applications that enables both interoperability and scalability, which reflects our perspective of the Internet and Web pertaining to Internet being the network core, messaging infrastructure as a layer of distributed operating system, and Web applications supplying diverse services to heterogeneous end users.

Our exploration is conducted in design space of system composition (between service and heterogeneous client graphics user interface), communication (with messaging infrastructure in publish/subscribe scheme), interoperation (among constituent components enabling general collaborative patterns), and reusability (integration with legacy desktop applications). This approach allows the deployment of M-MVC as a paradigm of distributed applications with above essential system features that fulfill our design goal.

In support of the primary design purpose, we have carried out experiments with Batik Scalable Vector Graphics application [BATIK] for prototyping and performed a series of performance measurements to test the effectiveness of our approach. Since graphics user interface (GUI) plays an important role in enhancing human-computer

interaction and dominates interactive style applications, a graphics enriched open source system like Batik provides us with an excellent environment for the investigation of critical M-MVC properties. Especially, low-latency visual responses and compute intensive rendering mechanism are highly demanding for the architecture of distributed applications over the network. The aim is, in a coherent and quantitative manner, to analyze the relationship of visual interactivity and system behavior and identifies factors that affect overall performance of message centered approach.

Note that we conceive distributed application has broader semantics than Web application with the former deploying on general network systems that may or may not be based on the Internet protocols. However, since Internet-based Web applications are dominant in the development of distributed applications, this thesis uses these two terminologies literally without strict distinction unless it is important to point out the difference.

The remainder of the chapter outlines a complete picture of the thesis research. Section 1.1 through 1.4 cover the general context and important issues for Web technology evolution; the thesis approach to messaging centric design applied to the M-MVC model of Web application development and message-based collaborative paradigms (SMMV and MMMV). Section 1.5 summarizes the contributions of this thesis. Section 1.6 contains a summary of the research scope to highlight core technology constituents, research questions, and experimental methodology. It poses a set of research questions that guided our work; our answers to these questions are given in chapter 9. Section 1.6 finishes with a roadmap of the rest of thesis which contains a detailed

technology analysis, system design and prototyping, performance evaluation, and conclusions.

1.1 Motivation

Since the first electronic computer ENIAC [ENIAC1945] was built, many innovative technologies have emerged as imposing forces in facilitating the revolution of advanced computation and information processing in an accelerated speed. While Von Neumann's stored-program architecture [VONNEUMANN] still greatly influences the building of most modern digital computers, the development of network and Internet technologies has revolutionarily promoted the computational power through interconnecting these scattered resources into a worldwide repository. Today, Internet and Web technologies have evolved into a unique global information infrastructure that reshapes our society pervasively throughout every branch. Traditional Computing based applications in science and engineering fields has developed into an Information Technology (IT) industry with extended services embracing computing, information management and processing, and communication. The services, as we defined in this dissertation, are "computation core" which comprise the core computational components or functionalities of the software systems.

Moore's law [MOORE] implies that computer performance will continue to improve while networks will also continue to increase in bandwidth [GILDER] with however latency for long distance linkage remaining higher than that needed for interactive use. Thus inevitable infrastructure improvements will tend to make it a unique opportunity but a great challenge for the deployment of new application architecture that better utilizing CPUs power and the existing physical networks infrastructure in

providing sophisticated services (e.g. Internet collaboration enabling virtual enterprises and large-scale distributed computing) and multi-media rich interface to end users from heterogeneous environments with properties such as scalability, reusability, interoperability, ubiquity, reliability, high performance and cost effectiveness.

Recently, individual technologies and systems such as J2EE [J2EE], .NET [DOTNET] and XML [XML] have contributed diverse solutions to Internet applications. Technology innovations promise a better future with continuous newly added features including ubiquity and collaboration. In reality, applications become increasingly complex and a single application or platform solution can not meet all needs. The developments and technology gap has brought the industry into a crossroad for changes because it is no longer affordable to develop software infrastructure in a conventional isolated path. The benefits of productivity, efficiency and adaptability demand for overall system design enabling software interoperability and convergence.

Web Services [WEBSERVICE] defines standard interoperable interfaces for different software assets to communication with each other through exchanging XML-based messages over a network. Vendors provide accommodating software implementations with programmable platforms enabling applications being defined, published, and used as Web Services. Web Services are a particular realization of a Service oriented architecture (SOA) [SOA] which proposes applications being built with decoupling of core business logic and database from the presentation layer, and using Web Services technology to interoperate between the application logic and the presentation layer. In essence, SOA and Web Services provide a generic and dynamic distributed framework. Interoperability entails reusability of existing software assets.

Convergence closes the technology gap between different environments. Historically, object oriented technologies like CORBA [CORBA] attempted but it is considered to be too closely coupled to provide a scalable distributed platform.

It is expected that substantial increase in migration from traditional approach to SOA in the design of future web-based distributed application. While Web Services technology allows development of loosely coupled applications that can be distributed and accessed as a collection of services, from any client, across the network; it does not define the composition of a service enabled application metaphor. Neither does it define possible state and transition of a service (i.e. the nature of the input messages inducing state change or the output messages reflecting state change), nor how these factors affect GUI framework (although Portlets and WSRP are important developments discussed later). There still needs tremendous exploratory work to find a unified and viable design for building service-enabled applications.

1.2 Statement of problems and Insights

Deployment of software applications is greatly influenced by supporting functionalities of operating systems, which are in turn driven by the advance of underlying hardware subsystems in architecture and capabilities. The impact has many dimensions that range from software architecture design, development and adoption of programming language, optimization of system performance with advanced algorithms.

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 and network bandwidth enables design and

implementation of new software architecture with satisfactory performance that allows development of many capabilities previously impossible.

In this section we offer some assessments about recent developments, current problems, and forward-looking features that we think will have a significant impact on software design and engineering. Among many connected technical issues, we choose five to summarize the situation:

Firstly, the Internet provides a global information infrastructure for the sharing of resources over heterogeneous environments. The communication subsystem of the Internet has evolved into stability with the UDP [UDP] and TCP/IP [TCP/IP] network stacks dominating the communication protocol domain [S. Shi] and forming the low-level sphere surrounding hardware network core.

Secondly, the work of constructing distributed operating system over the Internet has not completed and keeps adding new functionalities to the general purpose platform. One current effort focuses on building of messaging infrastructure tailored for supporting disparate applications. It involves software level routing intelligence, which goes beyond the best-effort services (e.g. transmission of bit streams) provided by stateless hardware network core, and addresses issues including notification services, reliability, Quality-of-Service, security, adaptability for multiple topologies (e.g. unicast [UNICAST] and muticast [MULTICAST]) and scalability over heterogeneous platforms (wired, wireless, and virtual private network (VPN) [VPN] subsystems), and customized communication services for specific applications.

Thirdly, software systems become increasingly distributed, complex, media rich, interoperable, integrated, collaborative, and heterogeneous. This trend demands system

designs enabling flexibility and adaptability for fast change, expansion and incorporation of existing assets in the operational systems. However, the devising of architecture for Web applications is still an unfinished challenge. Particularly, lacking of an agreed paradigm accommodating for increasing complex problems caused large amount of isolated applications being repetitively constructed. Over the decades, the Internet operating environment exhibits an architectural evolvement based on models including client/server, multi-tier, peer-to-peer, a variety of distributed systems (e.g. RMI [RMI], CORBA [CORBA], DCOM [DCOM], J2EE [J2EE] and .NET [DOTNET]), and Grids [GRIDS] system. One latest development is service-oriented architecture (SOA) [SOA] linked with loosely coupled messages for scalability and interoperability among existing computer systems. Web Services [WEBSERVICE] supply software platforms for building applications as services. Application developers need to meet the challenge ─ exploiting this concept in design and implementation to provide scalable interoperable systems.

Fourthly, the deployment of Web applications show diverse directions but have common features ─ namely, user interfaces and services for the sharing of information and resources over Internet infrastructure (see table 3.1 and 3.2). The "sharing" can be done asynchronously and synchronously at every possible stage along the deployment pipeline. The original World Wide Web proposed sharing of HTML document using HTTP protocol over the Internet (ref. Appendix D). 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 to form ad hoc communities (e.g. P2P network sharing of MP3 files) and virtual organizations or enterprises (e.g. Grids supporting of structure and unstructured societies). Therefore, in the most general sense, collaboration is the core problem and service of Web applications although people often use the terminology “collaboration” to only apply to real-time synchronous Web applications with compelling time issue or constraints.

Finally, next generation of Web client should enable pervasive accessibility, which has two implications: its ubiquitous availability to clients from heterogeneous platforms (e.g. Windows, Linux, UNIX, Macintosh, PalmOS, and Symbian [SYMBIAN]) that accommodate to thin client demands; its uniform Web interface that provides a platform with integration of multiple services.

1.3 Why message-based MVC?

The primary purpose of Message-based MVC (M-MVC) is to provide a high-level application architecture that converges desktop application and distributed application with automatic collaboration and universal access support, so as to simplify the development of possible new generation of interactive applications.

M-MVC can be viewed as a distributed MVC paradigm, although it is not directly extended from single user model to multiple users heterogeneous platforms. Rather, it is rooted from a loose coupling (message centric) distributed architecture with extension to unify legacy MVC metaphor.

Emerging (Web) service oriented architecture naturally fits into M-MVC paradigm, as Web Services (ref. Appendix D) facilitate interoperability of applications as services.

We identify that the Model (or "computation core") and the View of M-MVC correspond to the service and presentation in SOA model respectively, and their linkage with loose coupled messages can be achieved by publish/subscribe interface from underlying messaging middleware, which plays an increasing central role in the development of application as services with interoperability and scalability.

There is much confusion between the service, resource, and object. W3C [W3C] first popularized the term "resource" and used it for any electronic entity. The term object tends to refer to a software resource accessed by RPC and whose internals (class structure) are effectively exposed by the RPC (RMI, CORBA IDL) mechanism. "Services"are intrinsically loosely coupled; they can be associated with particular resources as in the WSRF [WSRF] framework; they can just opaque capabilities which might be accessed via RPC but with no expectation as to the internal class structure. In this thesis, we use the word "resource" in W3C manner to represent any electronic entity. For instance, one can use resource to refer to a small data set for transaction, a computation process, or a large database component. In a broad sense, a legacy desktop (or client) application is also viewed a resource. In fact, a principle goal of the thesis is to integrate such resources as valuable services into generic distributed system model.

We have a blueprint of the Internet and Web. As illustrated in figure 1.1, we take the view that everything is a resource; no matter it is in form of information (e.g. raw data, text file, bitmap image, MP3 music, video/audio stream, and program), physical device (e.g. computer, printer, fax machine and sensor), and even human resource. All of the distributed resources are linked together through local area network (LAN) [LAN] and further interconnected to other networks via wide area network (WAN) [WAN], which

forms the largest internetworking infrastructure ─ Internet (see fig. C.4). Web, which is built on top of the Internet ─ physical network core, provides communication channels for the interactions with message streams. Web applications add sophisticated services and interface for the "sharing" of services, objects, or resources for end users. Fig. 1.1 displays a layered view of the network system that includes the above constituents.

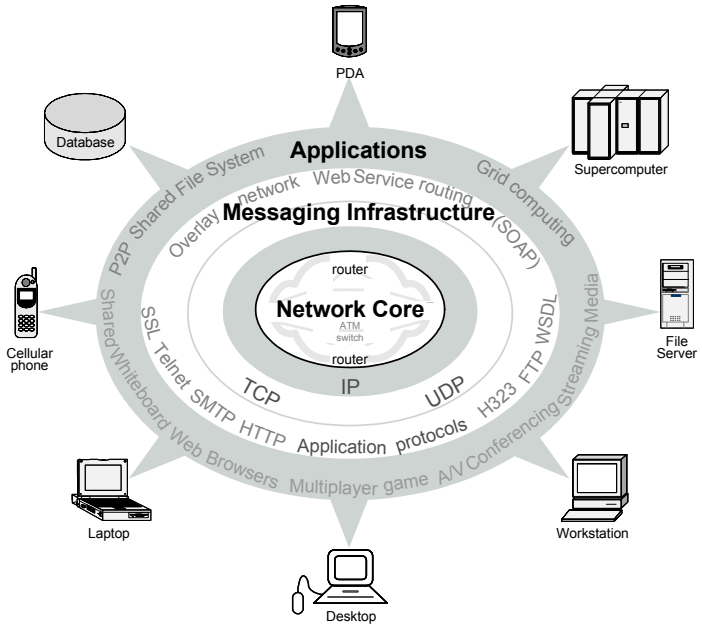


Figure 1.1 Architecture of network system

The growing demand for intensive interoperation and communication highlights the increasing importance of messaging ─ a ubiquitous solution embraces all forms of communication. A message may contain information of diverse formats (e.g. text, media, and raw data). A messaging service, as we discuss here, is a generic communication mechanism that facilitates the exchange of loose-coupled messages among the distributed objects or applications. Distributed resources or software components are wrapped with (Web) service interface and immersed in a sea of messages.

From a technical perspective, there are some distinctive features. Traditional distributed object model employs exchanging coupled-messages with explicit or implicit shared context, which may be implemented by 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 contracts rather than direct coupling, 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], Web Services Description Language (WSDL) [WSDL], and Universal Description, Discovery, and Integration (UDDI) [UDDI] that provide a generic interoperable platform among heterogeneous systems, which increase interoperability, reusability, and discovery of existing software components.

From an architectural view, a virtual distributed operating system is formed as an intermediary layer over the conventional bit-level Internet infrastructure (physical network and protocols such as IP, TCP UDP, HTTP, and SSH). Community Grids Lab's current effort NaradaBrokering [NARADABROKERING] focuses on building of messaging infrastructure over IP and 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 graphics user interface (GUI) and server side application behavior. Typical examples are JSP [JSP] and Java Server Faces (JSF) [JSF] for J2EE (or similarly ASP for .Net), JSR-168 [JSR168] and WSRP [WSRP], and REST [R. Fielding].

Each example addresses issues in targeted problem scope. However, one still needs a paradigm with a highly flexible architecture that is across the spectrum of platforms, programming languages, applications, and communication protocols so as to accommodate to rapid changes of individual technologies and adapting to sophistication, and cost-effective requirements in real world.

This motivates us to look into some intrinsic design concepts of client system (MVC [MVC]), event-based messaging system ([NARADABROKERING]), 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 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)
- employing publish/subscribe scheme, which is provided by the messaging infrastructure, for the exchange of messages among system components to enhance group collaboration capability

As in any new approach, there will be many subtle factors may not be addressed by general architectural consideration. So, we choose to build a prototype with a forward-looking architecture and conduct systematic experiments to explore and identify general principles and key implementation issues associated with this approach. Here, we list the major observations and analysis of the state-of-art in the evolving areas of software design, which form the starting point of our work. We build up our key idea in four stages:

We believe that Web applications ought to be built on messages to achieve important features such as scalability and interoperability. Method-based linkage of program components is obviously important and often the best approach. However such linkage implies tight coupling which handicaps both modularity and distribution. One can build distributed systems with RPC [RPC] like method-based models such as RMI,

CORBA, and COM. However although these platforms can be used in closely coupled and self-contained applications, they do not perform well on Internet scale distributed systems.

Messages resemble information dissemination from people to people. Messaging provides a mechanism facilitating the fundamental communication pattern of human interactions. The messaging approach is consistent with important principles in network design. Namely, scalability is more pressing than optimality in large network systems [COMPUTERNETWORK]. Especially, it complies with the Internet structure, which is built on diverse interconnected autonomous subsystems and subjects to flexible expansion. Further more, interoperability for systems from different platforms is an equally important design trait to pursue. Messages offer the abstraction accommodating to diverse system data format, which convey critical data and information for exchanging.

Historically, messages passing mechanism has been successful in parallel computing [PARALLELCOMPUTING], a tightly coupled distributed model, with satisfactory system performance for applications of sophisticated connectivity and synchronization issues. Moore's Law says that computer processing speeds double every 18 months [MOORE] but Gilder's Law implies that network bandwidth rises even faster than this [GILDER]. People’s ability to interact remains roughly constant. These trends continue and the difference of performance improvements continuous to widen, implying that explicit messaging gets more attractive especially for user interfaces (model-view interaction) where one gains from both computer and communication performance increases.

The message-based approach is an indispensable part of the big picture of Web system design. It requires a clear abstraction or insertion of an intermediate messaging layer so as to hide the complexity and diversity of services that reconcile the differences between underlying platforms and top level applications. Specifically, it decomposes a Web system into physical networks or Internet, messaging infrastructure, and Web application, as illustrated in figure 1.1. The system architecture includes hardware network core, whose primary function is reliable transmission of binary bits over distance, with routers and switches emerging at the edge and providing intelligent linkage; the ring of IP [IP] that offers critical routing protocols for the communication of large heterogeneous inter-networks; the messaging infrastructure sphere encompasses intelligence over IP, which includes a suite of traditional layered network protocols such as TCP [TCP], UDP [UDP], and a variety of application protocols; the outer sphere that provides application services to end users. We will cover related constituent components in detail in section 2. Abstraction of the messaging infrastructure as a separate layer of Web system has significant meanings. As discussed in the earlier section, the trend of Web application development shows increasingly complexity in functionalities. It demands for service aggregation with scalability, interoperability, reliability and pervasive accessibility. Messaging and IP layers essentially form the core of distributed operating system with rich communication services. This separation can greatly reduce the deployment overhead of Web applications. More importantly, it reduces their dependency on the details of underlying connectivity topologies and platforms, thus helps for application portability.

Service oriented architecture with loosely coupled messages linkage can maximize its capability by using advanced messaging services of the underlying infrastructure. We expect this architecture will have a continuing important role in Web applications deployment by preserving flexibility, scalability, interoperability, and reusability over the Internet model of diversity and arbitrary complexity. 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.

Message-based Model-View-Controller (M-MVC) provides a paradigm for next generation of software applications. It emphasizes a universal modularized service model with messaging linkage converging desktop application, Web application, and

Internet collaboration; it suggests a uniform platform for next generation Web client. Model-View-Controller (MVC) [MVC] is a fundamental architecture of Graphical User Interface (GUI) [Krasner+Pope] with system decomposition into triad of Model, View, and Controller for modularity. As a design paradigm, it is nothing new in the object-oriented programming world. What has served to rejuvenate the MVC concept again, however, is the realization that the pattern is particularly well-suited to addressing many of the fundamental problems inherent in building Web or distributed applications (e.g. design of user interface).

An aspect of distributed applications that requires research attention is the user interface. A well designed user interface is required if an effective use of the service is to be achieved [Abdullah+Gay]. A user friendly interface provides visual cues that facilitate navigation and effective access to available services. In client/server applications, Web browsers have provided relatively simple but effective human/computer interfaces for HTML content. In new generation of Web applications, GUI is expected to supply more intensive human/computer interactions both in richness of multimedia contents and versatility of communication. This puts high demands for a coherent architecture from user interface to autonomous Web Services. Although SOA or Web Services provide universal APIs for applications as services, however, they themselves do not address system decomposition and user interface issues. Thereby, application developers have to determine which component should reside in the service versus client interface. Further more, next generation client interface promise ubiquitous accessibility, but common client interfaces such as IE and Netscape Web browsers are not sufficient to deal with the variety of client profiles (e.g. thin client interfaces for mobile devices). MVC, as a

paradigm for interactive applications, has a crude system division of computation and presentation into two components: the Model and the View. This model makes it impossible for the system process being controlled in a fine grained fashion for distribution, especially for complex systems. For example, recent research shows increasing demand for multi-tier architecture with separate business logic at back end [LWLH]. To address the issues of the integration of legacy applications, M-MVC proposes a generalization of the classic MVC model (ref. section 2.2) and the Web Service pipeline model (3.2), which enables a unified service model with event-based messages linkage. In this architecture, system is decomposed of multiple stages and each stage forms a modular component with input/output linked to event messages for communication. As such, the decomposition of the Model and the View becomes a flexible and systematic approach based on different combinations of multiple pipeline stages, and messaging linkage facilitates for system division and distribution.

1.4 Design features of M-MVC and collaborative paradigms

As key to this dissertation, we propose "explicit message-based MVC" as an approach that systematic utilizes MVC in a message-based fashion with replacement of conventional method-based model and exploiting it with Web Services architecture in provision of a unified general approach of message-based service model for Web applications.

One prominent feature of the architecture with M-MVC is that the deployment is centered on distributed applications but devised for seamless integration of legacy

desktop applications and automation of emerging important features including Internet collaboration and pervasive accessibility.

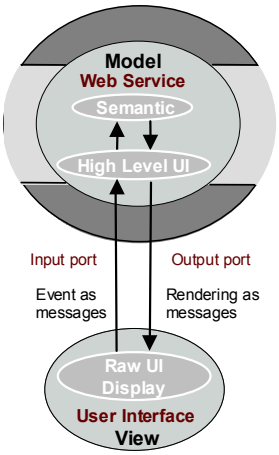


Figure 1.2 M-MVC model

M-MVC decomposes an interactive Web application using a flexible and fine-grained double-linked multistage pipeline model (ref. section 3.2) with natural event interactions. Theoretically, any part of an application with natural event interactions may form an object or stage with messaging linkage along the pipeline. The events, which represent the change of state, propagate along the path as messages that interconnect a dynamic graph of a finite state machine. In M-MVC architecture, we exploit the model-view compositions based on a variety of flexible combination strategies of these stages. This scheme provides many possibilities for building of Web applications with thin client interface facilitating universal access. We select a simple example ─ the three-stage pipeline (see fig. 1.2) to illustrate our design concept with raw UI events forming the View component; high level UI events and semantic events comprising the Model component (a Web Service); message-based events that play the role of the Controller linking the Model and the View components.

The major distinctions between MVC model and M-MVC model reflect different vision of system composition. A canonical MVC structure encompasses Model, View, and Controller ─ three independently existed and tightly coupled component classes at application level. However, modern architecture emphasizes scalability and interoperability. A refined Web service pipeline structure (ref. section 3.2) provides building blocks that accommodate dynamic, diverse, distribution and coordinative nature at Internet scale. M-MVC remedies the gap and argues that a basic system interaction is accomplished by a series of transformation along pipeline stages between the two ends of View and Model. M-MVC regards that Model (service) and View (user interface) are distributed components of an application that surrounded by messages in the background infrastructure as part of distributed operating system (see fig. 1.4). There's no single Controller class, rather its semantics are contained within control messages and processing is combined with various services along the pipeline stages.

From the beginning of our research on a generic model for building Web applications, Internet collaboration and ubiquity have been considered as important features to be integrated into the system design. We separately propose two interactive

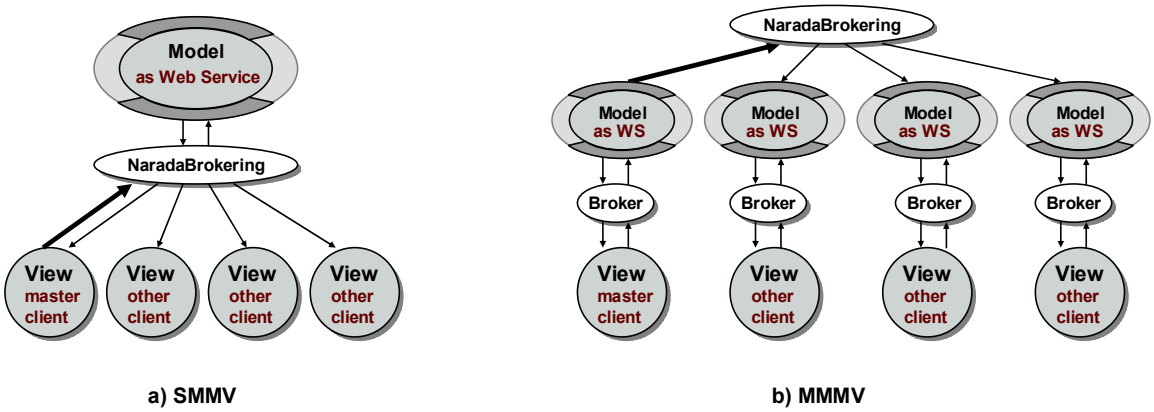


Figure 1.3 Collaboration paradigms deployed with M-MVC model

patterns ─ Multiple Model Multiple View (MMMV) and Single Model Multiple View (SMMV) for the general architecture of collaboration as Web Service model. The following two graphics in fig. 1.3 illustrate how SMMV and MMMV are deployed with M-MVC architecture, which facilitates assembly of either visual components (Views) or aggregation of Web Services (Models) through messaging services (e.g. NaradaBrokering [NARADABROKERING]).

SMMV and MMMV patterns extend the concepts of corresponding shared output port and shared input port models of the event-based collaboration framework [Fox03] and provide uniformed collaborative paradigms in publish/subscribe scheme for both desktop and distributed applications. While multiple clients join in a collaborative session, it is devised to assign "master role" to one client at each time. By sharing events from the "master client", all participants behave in a consistent and coordinated manner. SMMV is comprised of different client interfaces that share of the same model component. Shared display and Instructor led learning applications have SMMV structure. MMMV allows each client interface driven its own model, which is adaptable for more sophisticated applications and various user interfaces. SMMV and MMMV support both asynchronous and synchronous scenarios.

Figure 1.4 shows conceptual architecture for M-MVC applications. There are two key elements: the structure of M-MVC itself and its relationship with the messaging infrastructure.

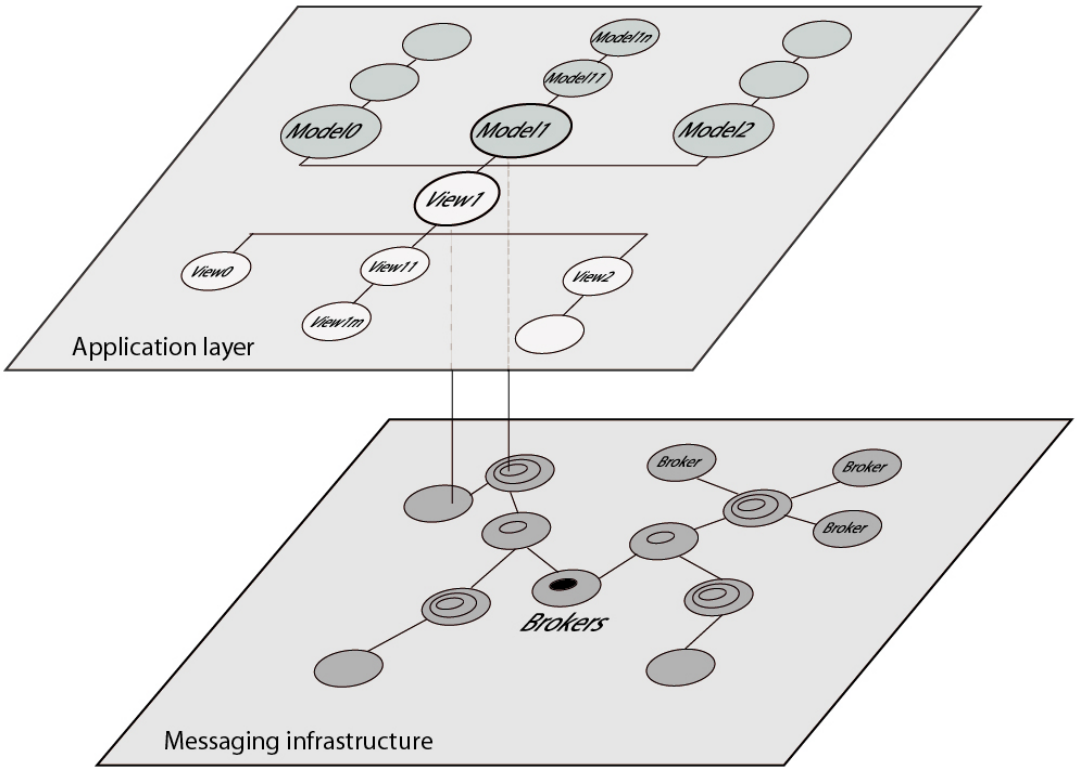


Figure 1.4 Message-based MVC and messaging infrastructure

Firstly, application architecture constitutes a separate layer from underlying communication infrastructure. (Web) Model (service) and View components are distributed in the application layer. Communication services (e.g. publish/subscribe and reliable messaging), which are provided by messaging brokers, form the infrastructure layer.

Secondly, in opposed to the tight coupling MVC triad (see fig. 2.5), M-MVC demonstrates a framework with three dimensional expansions. Each Model and View component can extend along the pipeline to form different stages of transformation (e.g. layout styling filter in View and multiple tier component such as separate business logic in Model) for system interaction and synchronization; and expand horizontally at each stage to represent disparate profiles. Vertically, connections between scattered Model and

View components are conducted via interfacing with messaging layer brokers that route event messages (e.g. control information) over the networks.

The heart of problems that modern architecture faces is increasing complexity of applications in versatile environment. As a case, fig 1.4 shows how M-MVC can be used to model Portal [PORTAL] applications. “View1” is equivalent to Portal UI such as JetSpeed [JETSPEED] that aggregates portlets user interfaces (“View0”, “View1m”, and “View2”). A filter component for Web page layout styling might be need as depicted by “View11”. Portlets such as “Model0”, “Model1”, and “Model2” are consists of different services that ultimately access various computing and database storage resources that distributed at the edge of the Internet (ref. fig. 1.1).

Note that distributed components, which include (Web) Services (Model) and User interface (View) modules, are linked by messages through messaging brokers with a variety of communication services. By registering with Publish/Subscribe services, different clients dynamically join and leave the system while services are integrated on demand. The approach also facilitates an automatic collaboration framework that can run either in a client/server mode or in a peer-to-peer mode, depending on the run time binding of messaging service with JMS or JXTA profile.

In this dissertation, we provide practice and experience that help to expedite 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. In summary, M-MVC emphasizes a combination of messaging flexibility as well as component modularity for the modeling of real world complex problems. We have included the incentives of our

approach of building distributed applications around messages, and briefly described some design features of M-MVC in support of Web Service composition and major collaboration patterns ─ SMMV and MMMV. A complete set of discussions on M-MVC that embraces its design, implementation, and performance evaluation are provided in subsequent chapters 2 to 7.

1.5 Contributions

1.5.1 Research achievements

The main contribution of this dissertation is to offer a comprehensive solution to building applications centered on messages. It is the first research as far as we know that systematically utilize Model-View-Controller (MVC) [MVC] paradigm for distributed application deployment in a message-based fashion. It enables the provision of a universal paradigm with a service model converging desktop applications, distributed applications and Internet collaboration. This work has following implications:

- Proposing an "e*xplicit Message-based MVC*" paradigm (M-MVC) as the general architecture of Web applications [QCF-06-03].
- Demonstrating an approach of building "*collaboration as a Web service*" through making decomposition of M-MVC collaborative [QCF-07-03]. As an example, we present architecture for three types of collaboration ─ monolithic, thin client, and interactive client.
- Bridging the gap between desktop and Web application by leveraging the existing desktop application with a Web service interface through "*M-MVC in a publish/subscribe scheme*" [X.Qiu]. As an experiment, we convert a desktop

application into a distributed system by modifying the architecture from method-based MVC into message-based MVC.

- Proposing Multiple Model Multiple View MMMV and Single Model Multiple View collaboration SMMV as the general architecture of "*collaboration as a Web service*" model [Qiu+Jooloor].
- Identifying some of the key factors that influence the performance of message-based Web applications especially those with rich Web content and high client interactivity and complex rendering issues [QPU].
- Future work includes extending our ideas, tools and architectural principles to other Web applications such as collaborative whiteboard and data visualization.

1.5.2 Significance of research

We propose a different approach of "explicit message-based MVC" (M-MVC) paradigm for application deployment, which delineates our design concept of building Web applications centered on messages. It encompasses our investigation of the interoperating relationship among constituent components of applications ─ from tightly coupled desktop application to loosely coupled distributed system. The most challenging part of the research work is a unified solution that reconciles the different architectural principles derived from the disparate system objectives of integrating MVC, messaging, Web Services, and collaboration models. M-MVC replaces opaque method-based events at application (Java) run-time level with exposed messages, and changes the tight connections of conventional method-based MVC model to a loosely coupled messaging for distribution.

We now provide further definitions of system features: message-based *Controller*, thin client as the *View*, and Web Service as the *Model*. M-MVC offers a framework with a double-linked multiple-stage pipeline architecture to refine MVC decomposition; we define MMMV and SMMV collaboration models using a publish/subscribe scheme supplied by the underling messaging infrastructure. As a general architecture, it emphasizes a modular service model with messaging linkage for reusability, scalability, interoperability, and automatic collaboration with universal accessibility. We can identify four generally important aspects:

Firstly, it provides a mechanism to make a desktop application as a Web service to allow maximum reusability.

This is done through converting a desktop application into a distributed system with modification of architecture from traditional method-based MVC to M-MVC in a publish/subscribe scheme, where computation core or *Model* naturally becomes a Web service. Conventionally, Web and desktop applications are developed with different architecture. Nevertheless, integration of legacy client side systems or existing components into up-to-date Web development is one of the crucial aspects of the technology evolution. This approach suggests that desktop applications with well modularized design can be modified to Web services, which greatly maximizes the reusability of existing components for Web application development. The architectural changes bring up issues that cause a challenge to the system. The experience also helps in finding the principles of building Web applications from scratch.

Secondly, it allows us to have a different view of building distributed applications around messages to achieve scalability and interoperability.

The architecture of traditional Web applications deployment is built on individual platforms evolving underlying point-to-point model (e.g. client/server and multi-tier), multicast model (e.g. peer-to-peer overlay network), and RPC-like distributed system (e.g. RMI, CORBA, DCOM, J2EE and .NET). This dissertation has adopted a different approach ─ a loosely-coupled message centric design with separation of Web systems into application architecture and messaging infrastructure layer. A message-based Web application with M-MVC architecture comprises three functional modules: thin client interface represents "view"; computation core stands for "model" which becomes a Web Service; messages, which convey abstracted context information from both components and facilitate synchronization between them, play the role as "Controller". This approach enables a universal paradigm of Web applications that applies to any of the above platform architectures, which is also adaptable to wireless network, VPN [VPN], and future development of other possible connecting topologies. This follows from our layered stack with a messaging middleware providing a communication channel that handles underlying network protocols and services over heterogeneous topologies. The higher level Web applications focus on the deployment of presentation and interface services to end users with high scalability and interoperability.

Thirdly, this paradigm provides a uniform architecture that bridges the gap between desktop and distributed applications seamlessly to support universal access.

Theoretically, any part of an application with nature events linkage can become splitting points of the View from the Model, which forms a multi-stage pipeline. However, traditional deployment of Web clients commonly mix presentation with content, which made it hard to build a general architecture of uniform client interface with rich Web content for service-oriented model. The M-MVC model exploits on further separation of the rendering from the logic components of traditional Web client, which enables a thin client structure. A thin client design allows a desktop application to form a local service part of the Web application, which also provides remote services to other Web services. Therefore, it facilitates the seamless unification of desktop and Web application. This scheme is also critical for universal and pervasive access. Particularly, it suggests a uniform approach to build next generation Web client with desktop and web applications sharing a common portlet (WSRP [WSRP], JSR168 [JSR168])-based architecture. These ideas can unify PDA and desktop, as well as Linux, MacOS, Windows and PalmOS applications to achieve pervasive accessibility.

Fourthly, this approach makes automatic collaboration ─ MMMV and SMMV collaboration pattern provides the general model of collaboration as a Web Service.

Multiple Model Multiple View (MMMV) and Single Model Multiple View (SMMV) present a generic collaboration approach that derived from our message-based MVC architecture of Web applications. MMMV and SMMV accommodate respectively instructor-led learning and participatory learning models. The trend of Web applications deployment demands for the design of architecture in provision of capabilities that allows assembly of diverse clients and aggregation of different

services. From a client's point of view, it requests for accessing to a variety of services (e.g. email, search engine, and web browser); from the perspective of a service, it holds commitment to supporting of heterogeneous client interface. As a response to the trend, we demonstrate that M-MVC, as a paradigm of message-based service model, has features of both messaging flexibility and component distribution and is a suitable architecture for complex problems with many-to-many dimensionality. More importantly, our work suggests that one need not develop special "collaborative" applications. Rather any application developed as a Web service with M-MVC model can be made collaborative using the tools and architectural principles discussed in this thesis. This could motivate the development of new desktop applications that preserve interoperability and sophisticated rendering effect while gaining many capabilities (e.g. collaboration) not present in today's systems such as Openoffice and Microsoft Office.

An additional contribution of this thesis is taxonomy of event-driven message-based collaboration framework. It covers a feature summarization of event-based collaboration applications, discussions of main design concepts embracing system composition (service, client interface, and session control components), event structure and interactive pattern, interfacing with messaging infrastructure in explicit publish/subscribe scheme, and classification of monolithic and Web Service collaboration model. The key concepts has applied in our system with scalable vector graphics content as well as other collaboration systems like shared display, shared data visualization, audio/video conferencing, and online game.

1.6 Summary of the dissertation

1.6.1 Scope of Research

This dissertation builds on design concepts from different areas of computer technology: desktop system, parallel system, distributed system, Internet collaboration, and Web system. Our investigation comprises the following aspects:

- Model-View-Controller (MVC) [MVC], a fundamental paradigm that separate system into triad of *Model*, *View*, and *Controller*, which is originated from desktop system and becomes a design pattern of object-oriented programming
- Messaging [NARADABROKERING] [JMS] [WSNOTIFICATION] [WSEVENT], a flexible and scalable message-based communication mechanism adaptable for complex problems with high dimensionality
- Document Object Model [DOM], a component-based structure for distributed system with a generic event mode; Scalable Vector Graphics (SVG) [SVG], an application of DOM specification
- The Web Service pipeline model of Internet collaboration [Fox03], an event-driven message-based collaboration framework that can be deployed in a clear publish/subscribe scheme with the messaging infrastructure [NaradaBrokering] support.
- Web services [WEBSERVICE], an implementation of emerging service oriented architecture for Web applications

The detailed technical points and their utilization in the implementations are presented in Chapter 3 to Chapter 6.

1.6.2 Research Questions

In this dissertation, we consider two major classes of applications: one is personal computer based, so called desktop application; the other is network or Internet based, referred to as distributed or Web application. Deployment of the latter shows multiple dimensions that targeted for solving different problems. Among them, there is content based hypermedia Web that supports desktop like high profile client interface with a mix of text, audio, video, two dimensional and even three dimensional graphics; there is Internet collaboration, which provides an interactive mechanism of sharing online information and computing resources in a synchronous (e.g. video/audio conferencing, multiplayer online game, shared whiteboard, Instant Messenger, portal for large-scale distributed computing) and asynchronous (e.g. email, news group, shared file system, Internet search engines) fashion; there is thin client platform (e.g. portlet aggregator such as JetSpeed [JETSPEED] or uPortal [UPORTAL]) that offers a unified service interface for variety of clients (e.g. PC, workstation, PDA and cellular phone) over heterogeneous operating systems (e.g. Windows, MacOS, Linux, Unix, PalmOS and Symbian [SYMBIAN]). All of these efforts motivate us to think about fundamental research of software design and engineering issues from the desktop to the Web.

Desktop and Web applications are built on totally different methodologies ─ the former is programmed on top of operating system optimized for using local CPU and storage resources; the latter is developed on distributed operating system over physical network and Internet infrastructure and takes advantage of online resources. Therefore, they are normally viewed as on two parallel tracks. However, deployment of legacy desktop operating system and applications have much longer history and have already formed a huge software industry with relatively matured technologies and a rich

collection of complex tools that range from text editing, graphics design, simulation, computation, data visualization, to database management. On the other hand, Internet and Web have seen tremendous growth over the last decade. Technologies of Web applications and underlying distributed operating system over Internet are under extensive development but still immature and keep evolving. This situation suggests that we may need to take another look at our methodology of Web development by reviewing through a considerable body of knowledge regarding work that already exists and asking following questions:

- Should we build every Web application from scratch or can we maximize reuse by leveraging existing components from legacy desktop applications? If the answer is yes, are there any design principles to follow? If not, is there a general approach to bridge the gap from desktop to Web applications?
- What is the core problem of Web application deployment? How would one abstract the problem into a generic model that best describes the features of diversified Web applications? Based on this model, is there a design paradigm that lasts while detailed Web technologies change over the time?

We employed a variety of empirical approaches, which includes building prototypes, to gain a systematic understanding of the design principles. Our work is based on the investigation of fundamental design models: MVC paradigm of desktop applications and messaged-based Web services of Web applications. This work also involves substantial study of approaches to Internet collaboration. We propose M-MVC as a uniform architecture for the deployment of desktop and Web applications with automatic

collaboration capability. This dissertation seeks to answer the following research questions:

- Can MVC be implemented in a message-based fashion?
- What principles are there to govern the decomposition of a given application into M-MVC components?
- What is the performance of the message-based MVC and what factors influence it?
- How does M-MVC depend on the operating system, the application, machines and network?
- What is the relationship of collaboration and Web services with M-MVC paradigm?
- What is the way to define state and state changes in collaborative applications?
- 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?

Each of these questions is discussed at the end of thesis in Section 9.2.

1.6.3 Methodology

To exploit our general approach of building distributed applications, we choose Batik SVG browser from Apache [APACHE] as the desktop application for experiments. Batik [BATIK] is an open source project from Apache Software Foundation, which involves industrial and individual effort that is lead by IBM. Batik SVG browser is an interactive presentation style application that implements Scalable Vector Graphics (SVG) specification version 1.0 [SVG], a recommendation of World Wide Web Consortium (W3C) [W3C]. SVG implements W3C Document Object Model (DOM) [DOM]

interface, which is an important model for Web application development with distributed resources. The combination of its high interactivity with rich vector graphics content in Extensible Markup Language (XML) [XML] format, DOM structure, and open source implementation with open standard makes Batik SVG browser an ideal experimental case for our investigation of building forward-looking Web applications architecture.

In this scenario, the de facto building blocks of Web applications should consist of a document structure for the abstraction of Web resources (e.g. raw data, text, vector graphics, and media stream), a generic model and architecture that defines an effective mechanism for collaboration as a Web service, and a powerful messaging infrastructure provides underlying support for communication services (e.g. a variety of publish/subscribe models including peer-to-peer overlay network JXTA [JXTA] and Java Message Service (JMS) [JMS] emulation, traversing firewalls, and multiple protocol support) between application components and interface with other applications. In our experiment, the triple factors become SVG document object model (SVGDOM), M-MVC paradigm and NaradaBrokering middleware [NARADABROKERING]. The composition is illustrated in fig. 2.6 and the details of using messaging services with NaradaBrokering's publish/subscribe and point-to-point interfaces will be elaborated in section 3.6.

We start from a survey of up-to-date Web technologies and set up an experiment methodology which enables us to have a complete analysis of a real desktop application with open source and modify it into a distributed system as a testing base for further prototype construction, evaluation and extension. Key steps of the research process are briefly listed below:

1) survey of new technologies to build Web applications for advanced architecture systems with rich Web content [QIU-10-2000]
2) monolithic SVG experiment
3) collaborative SVG
4) MVC decomposed SVG experiment
5) architecture of collaborative message-based MVC in SMMV and MMMV patterns
6) extensions of current work

The experiment is to make a presentation style client system with high interactivity over vector graphics content as a collaborative Web service. The problem itself is representative and complex in nature (see Table 1.1 and 1.2 for summary of typical Web applications). In design space, MVC paradigm and message-based Web Services are fundamental architectures from desktop to Web applications. Explicit message-based MVC unifies traditional method-based MVC and message-based Web services model seamlessly with automatic collaboration capability. These suggest that our approach has general importance. It can be generalized and extended to other presentation style applications (such as shared whiteboard, Office suite, and collaborative visualization for computing that are separately developed in Community Grids Lab (CGL) [CGL]) and non-presentation style applications (e.g. distributed computing applications like Grids computing). We have tested our prototype infrastructure with application samples (e.g. teacher-student scenario of shared SVG browser and multiplayer online chess game) that demonstrate its viability in supporting of complex functionalities with stringent timing constraints. This work provides a framework based on which one can carry on the ideas

and experience gained from this thesis to explore interesting research topics such as achieving performance optimization in depth and applying to other applications in broadness.

1.6.4 Organization of the Dissertation

This dissertation is arranged in nine chapters. An overview, which consists of the Introduction of Chapter 1 and Conclusions of Chapter 9, covers the general context of the research. There is a review of core technologies which focuses on discussions of architectural design schemes in Survey of Technologies of Chapter 2. The M-MVC Architecture which is the central focus of the design approach is described in Chapter 3 which includes descriptions of the system structure. My research with experiments associated with prototyping, implementation and architecture abstraction is contained in Chapter 4 to 7. Chapter 8 presents a detailed performance evaluation of our results. Other details of the thesis project in terms of background, underlying technology, and implementation environments are attached in the appendices at the end followed by the references. We now give a little more detail on the following chapters.

Chapter 2 surveys underlying technologies related to the dissertation for understanding the composition, communication, and interoperation aspects of software systems. These include section 2.1, an introduction to the foundation of interactive applications: event-based programming and event models; in section 2.2, we study the MVC paradigm which is the prevailing architecture for desktop and distributed application with GUI interface; section 2.3, has a discussion of the important messaging oriented middleware scheme for development distributed applications; as a specific messaging middleware system, we describe Naradabrokering and its publish/subscribe

communication service for event-based messages; section 2.5 delineates DOM structure and event model which provides key programmable interfaces for building distributed applications.

Chapter 3 covers the incentives and main concepts of M-MVC design. We start with a summary of the characteristics of typical distributed applications via event-based interaction between system components in section 3.1. An introduction to a Web Service pipeline model in section 3.2 depicts how to decompose and analyze applications with a fine grained pipeline diagram. Section 3.3 compares distinctions of MVC and Web Service pipeline model and provides a unified approach ─ M-MVC to bridge the application domains. We further propose SMMV and MMMV as two important collaborative paradigms that deploys M-MVC model in section 3.4. In section 3.5, we give an overview of various MVC approaches and provide an in depth discussion of different design trade-offs. Then, a discussion in section 3.6 delineates the important role of publish/subscribe messaging that provides support for delivering event messages with group communication capability. Section 3.7 provides explanations of how to deploy M-MVC model with Web Service architecture.

Chapter 4 describes our monolithic SVG experiments; by monolithic we imply that the application is not decomposed but model and view are contained in the same program. Section 4.1 includes a complete analysis of our test example ─ the open source Java Batik SVG browser. Section 4.2 shows how to build collaborative SVG without decomposing of Batik by intercepting events and sharing them among participants. We present event structures that enable one to make monolithic applications collaborative using SVG technology as an exemplar.

In Chapter 5, we propose two event-based collaborative Web Service model SMMV and MMMV which can be applied to instructor-led learning and participatory education applications respectively. Our experiment with Batik SVG browser decomposes it into separate view and model components. Chapter 6 shows an approach to convert a standalone client application to distributed system with M-MVC. In section 5.1, we summarize collaboration framework in both monolithic and decomposed (including Web Service) collaborative models.

Chapter 7 presents performance measurement and their analysis for the decomposed SVG model. In sections 7.1 to 7.3, we describe various test scenarios and timing issues and analysis performance results to identify key factors that impact the message-based approach for building applications. From these SVG experiments, we have derived lessons that can be instructive for future development, and these are contained in Chapter 8.

Finally, we conclude in Chapter 9 with a discussion of future work. Expansion of current work may generate many interesting research topics for future works.

Chapter 2

Survey of Technologies

This chapter is dedicated entirely to core technologies employed in the thesis project. We have conducted investigations of foundational concepts pertaining to event-based programming, MVC model, and messaging, in addition to NaradaBrokering middleware and DOM technology. The following subsections are composed of an introduction to each of the above subjects.

2.1 Event-based programming

2.1.1 Concept of Event-based Programming

Event-based programming is a programming style driven by events rather than data/state for an application system's runtime behaviors. In an even-based system, components coordinate by interactions of generating and receiving events. In opposed to the rule-based approach, event-based programming promotes system modularity and asynchronous response. The term "event" is used in a very broad sense. This dissertation conceives "event" as any information that invokes the change of system state whilst "state" is a transient status of a system set by a chain of runtime changes.

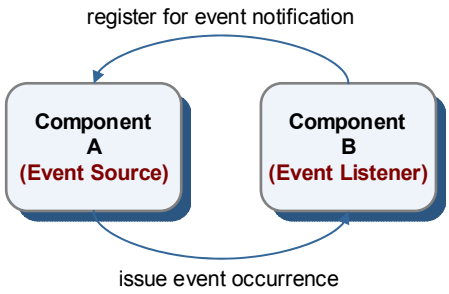


Figure 2.1 The general event/listener model

In general, an event-based system works in the following way: an event source entity issues an event; an event target entity receives the information and handles it with appropriate action. Fig. 2.1 shows the basic idea of how two different components coordinated within a system by communicating with each other through an event, where A and B are designated as event source and event listener components respectively. The general event model of source-target pair may have variant implementations (e.g. event/listener and producer/consumer models). However, this approach consists of a few key features: there is no single flow of control through the program; an event occurs spontaneously or asynchronously; the responses of the system are based on event contents and the current state. This can be compared with rule-based programming that has tight data flow control while the event-based model integrates system components with scattered decoupled events.

Event-based programming has been widely used in various systems including object-oriented systems, distributed component models, and a variety of Graphical User Interface (GUI) applications such as Microsoft Windows and Visual Basic, Sun Java AWT and Swing, Netscape and Internet Explorer browsers, and Macromedia Generator [GENERATOR]). It has become an extremely common framework for complex distributed system.

2.1.2 Event Models and Publish/Subscribe Mechanism

Event-based systems are supported by a number of event models that describe event flow and propagation among the system components. Event flow depicts the approach of event-based interaction between event source and event listener components. Event propagation defines the mechanism how event listener (target) propagates the event further on to notify adjacent components. Examples of event model are Java event delegation, CORBA OMG Event service [OMG] [CORBAEVENT], Web Service Notification [WSNOTIFICATION] and the Java Message Service (JMS) [JMS], Netscape and Internet Explorer browser event models [GOODMAN], W3C DOM event model [DOM2EVENT]. These are all variants of the publish/subscribe mechanism and share the general event/listener concept although they differ in implementation details.

The inherent system interaction pattern forms the essential signature of event-based systems. The following of this subsection introduces the main concepts of Java delegation event model and publish/subscribe model in an event broker based notification service. These two are representative cases illustrating tightly coupled and less coupled event-based systems respectively, which help our search for a uniform solution converging and bridging desktop and distributed systems. Our survey also covers briefly other major variants of publish/subscribe event models to show that our architecture of message-based MVC incorporates a general event model which can accommodate heterogeneous approaches and platforms.

Java delegation event model

Java-event model [JAVA EVENT] is denoted as a delegation model (e.g. JDK 1.1 event model) with multiple event listeners directly registering to an event source object and being invoked through call back methods when a fired event is dispatched to target

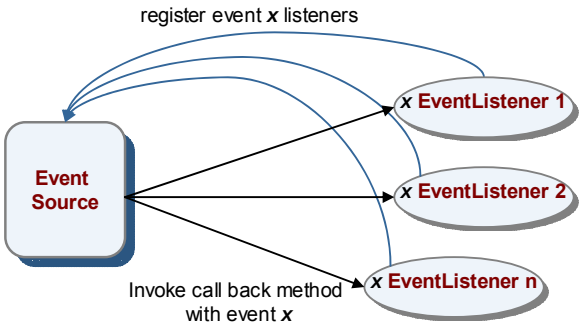


Figure 2.2 Java delegation event model

listeners.

Fig. 2.2 shows the basic one-to-many synchronous interactive relationship of event/listener in Java delegation event model. However, the whole event process can be more sophisticated in real applications such as Java AWT package. For instance, AWT hosts many components (e.g. Window and Frame) laid out inside containers in a hierarchical tree structure. Any of these components may become an event source object. One event source may contain/hold different types of events with each event type registered with multiple targeted event listeners; on the other hand, an event target component may add multiple event listeners to one or different event source components for notification of interested event types. Since W3C DOM as well as Java Swing package [JAVA SWING], Netscape and IE browser has very similar event/listener structure and share hierarchical tree model, a further discussion of system composition and event propagation will be given in DOM event model in section 2.5.3.

Publish/Subscribe Scheme

The classic publish/subscribe model is a topic-based communication pattern that enables multiple subscribers registering to one or more topics (e.g. content based information) while the publisher sends messages to topics; these messages are then dispatched to registered subscribers. It provides a many-to-many relationship between publishers and subscribers. The logic structure is illustrated in fig. 2.3. For instance, Subscriber 3 subscribes to Topic A, B, and C while publisher 2 issues Topic C related messages to subscriber 3, 4, and 5. In contrast with synchronous Java delegation event model, the publish/subscribe approach provides a separate logic ─ notification service between event source (publisher) and listener (subscriber) components. This can satisfy the system need for a decoupled, dynamic, and interoperable scheme that supports both synchronous and asynchronous event flow.

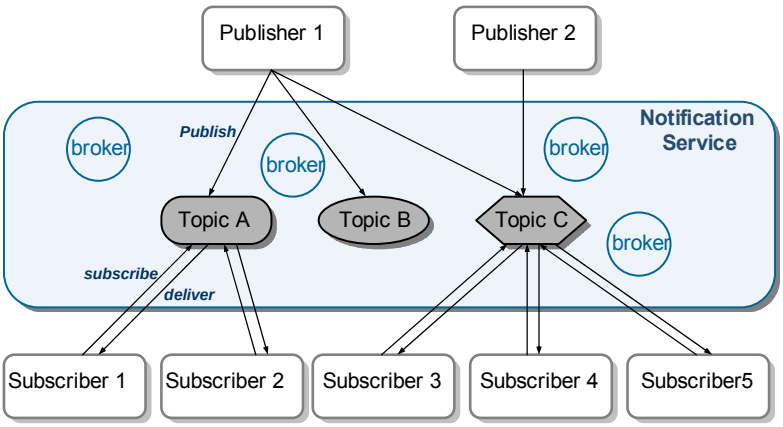


Figure 2.3 Topic-based Publish/subscribe model

As a key element of publish/subscribe model, a notification service plays the role as the "mediator" in between publisher and subscriber. The mediation may be conducted through "broker", a connector unit that routes topic events to the destination resource.

Literally, event and notification both refer to the information exchanged between the participants, where event source generates event and publishes notification of event occurrence to event listener. Notification may contain raw event data or processed or interpreted information. The notification service can be done through, for example, topic-based model or finer grained content-based model that embraces more expressive precisions.

The Publish/Subscribe model supports participants (publisher/subscriber) communicating through intermediary notification service in a many-to-many interactive pattern [VIRGILLITO]. The decoupling of event source (publisher) and listener (subscriber) has several desirable features for system integration: anonymity, many-to-many, virtualized and asynchronous communication. Anonymous addressing allows event routing between publisher and subscriber without them knowing each other. Many-to-many communication naturally supports group communication (as for example required in a collaboration system). Asynchronous communication promotes system decoupling for distribution over different software environments. In addition, subscribe and unsubscribe provides a mechanism that not only supports a long-term relationship (e.g. durable subscription in JMS pub/sub model) but also allows interactive short-term subscription (e.g. nondurable subscriber receives published messages only when it's actively registered within a session and remains in the context). All together, these traits of publish/subscribe model increase distributed system for scalability and interoperability for distributed systems with advantages of flexibility and dynamic response.

CORBA event model

The OMG CORBA event model [CORBAEVENTSERVICE] [CORBANOTIFICATIONERVICE] has supplier and consumer coordinated indirectly through an intermediary event channel with push, pull and hybrid modes [KLEINDIENST]. Push and pull define two approaches with event flow initiated from supplier and consumer respectively. The event channel typically provides notification service that "broadcast" non-typed or typed events from publisher to subscriber and enables a many-to-many relationship. More sophisticated implementation may offer services such as Quality of Service (QoS) [QOS], filtering, fault tolerance, and real-time scheduling.

JMS

Java Message Service (JMS) [JMS] is a message-based communication service in supporting of Java programs. JMS supports messages that contain serialized Java objects and other formats including messages that contain XML format. Using the JMS interface, a programmer can invoke the core messaging services of IBM's MQSeries [WEBSPHEREMQ], Progress Software's SonicMQ [SONICMQ], NaradaBrokering [NARADABROKERING] and other messaging product vendors. JMS supports two domains: publish/subscribe and point-to-point and both have intermediary message queue

Figure 2.4 JMS Point-to-point model

structure that buffers and manages messages [JMSTUTORAL]. Within a single-threaded

transactional context — session, the former has subscriber and publisher exchange messages through topics; the latter has producer sending a message to a specific message queue and a receiver requesting messages by requesting from the queue and acknowledging of acceptance (shown in fig. 2.4). JMS and the Web Service Notification described in next subsection are and can be expected to be used as interoperability frameworks between different messaging systems.

Web Services event

Web Services event defines how to construct an event-based message exchange pattern, which enables Web services to act as event sources for subscribers. It specifies interfaces of subscriber and event source, as well as event notification mechanism, resource identification, and message structure. Each vendor or organization comes up with a design in their own way. For instance, existing specifications of Web Services Notification (WS-N) [WSNOTIFICATION] and Web Services Eventing (WS-E) [WSEVENT] are from IBM and Microsoft respectively.

WS-N comprises of a family of specifications that defines event-based communication using a publish/subscribe pattern, which include WS-BaseNotification, WS-BrokeredNotification, and WS-Topics. They separately defines Web Services interface for point-to-point notification, intermediary notification broker, and topic expression dialects. WS-E defines a simple asynchronous messaging model with "push" delivery mode while general Delivery Modes extend to provide flexible customizations for various subscriber requests that could be "pull" and "batched". WS-N and WS-E have similar structure with common features such as event-based approach, event/listener pattern (e.g. Subscriber/Publisher versus subscribing EventSink/EventSource), and

asynchronous one-way message using SOAP [SOAP]. But there exist some differences. For example, intermediate brokering is supported in WS-N but not with WS-E; all resources are identified by an EndPointReference (EPR) in WS-E while WS-N refers to WSRF [WSRF].

Recently, major vendors such as IBM and SUN have joined Microsoft for a Web Services event specification that provides the base standards of asynchronous publish/subscribe style notifications to interested parties. The specification is expected to define a baseline set of operations for web services communication.

2.1.3 Summary

Event-based programming supplies a flexible and dynamic asynchronous interaction through event notification among distributed software components. An event model plays the key role in event-based system design. The event models examined in this section show an increasing important feature of distributed architecture ─ namely, they allow broadcast event messages to multiple recipients with such version differing in detail of a publish/subscribe framework. The topic-based publish/subscribe notification service logic can be implemented either by one or more distinct brokers or internally to the publisher and subscriber. For example, the former can be found in OMG CORBA event mode [SCHMIDT], and NaradaBrokering [NARADABROKERING]; the latter is used in Java delegation model, JMS [JMS] point-to-point mode and the Web Service WS-Eventing [WSEVENT] and WS-BaseNotification [WSNOTIFICATION] specifications for multiple subscribers to a single publisher. The event capture type of DOM event model is similar to Java delegation model, which we will introduce in section 2.5.3.

Publish/subscribe can be deployed in tightly coupling monolithic system like Java and CORBA or in a message-based loose coupling platform independent approach such as NaradaBrokering. The latter provides essential features that conform to our big picture of building Internet systems with explicit separation of Web applications from messaging infrastructure (refer to section 1.3 and fig. 1.1) and offers an important mechanism for the integration of our message-based MVC architecture with messaging infrastructure NaradaBrokering, which we will elaborate in the architecture design issues in section 3.6, 5.1 and 5.2.

2.2 MVC

2.2.1 MVC Concept

Model-View-Controller (MVC) is a fundamental paradigm for interactive applications, which includes almost all modern graphics user interface (GUI) design and becomes a popular design pattern of object-oriented programming (OOP) [Cox+Novobilski]. MVC initially appeared openly in Smalltalk-80 implementation [Goldberg+Robson], the “blue book”. The concept was explored in the “green book” [G. Krasner]. MVC inherited from object-oriented programming idea of Simula 67 [SIMULA67] with integration of graphical user interfaces and interactive program execution. Simula was a programming language originally designed and implemented for discrete event simulation, which built on extension of Algol60 [ALGO60]. It introduced main OO features including class, object, inheritance and virtual method. The great success of MVC popularized OO concept, which has been highly influential on modern programming methodology.

MVC proposed the logical separation of presentation from behavior and data structure in an interactive multiple windows programming environment. The main idea of dividing a system into *Model*, *View*, and *Controller* components is elegant and may appear simple, which can be found in many references as shown in figure 2.5. However, pervasive MVC deployment has much richer implications and MVC concept itself evolves over the time.

In the programming environment of Smalltalk-80 version 2.5 applications [S. Burbeck], the definitions of the three components are as following: the *View* manages rendering of bitmap buffered image of graphical and/or textual output; the Controller interprets device input events (from mouse and key) and commands appropriate changes to *Model* and *View*; the *Model* manipulates the behavior and data structure, responds to requests for its state information from the *View*, and instructions to change state from the *Controller*.

In the MVC triad, the relationship between constituents is build up on links. The View and the Controller constitute a unique pair that associated with each other via an instance variable pointer. At meantime, both have a "model" variable pointing to the Model object. The Model is relatively loose-coupled with the other two components and has a subtle communication scheme. In a simple passive mode, the Controller requests the change of the *Model* (e.g. a character input from keyboard) and takes responsibility to inform the *View* of this change. In a more sophisticated scenario, the *Model* may update due to interaction from a third party component that is outside of the View-Controller pair. The modification is notified to all dependents (e.g. view, subview and other components) of the *Model*, which is facilitated by the event/listener mechanism as

discussed in section 2.1.1. Note that the *Model* retains a hierarchical structure that allows dependents (or listeners) to register. In Smalltalk-80 v2.5, a global mechanism IdentityDictionary is used to keep track of the dependents and changed/update mechanism is chosen to coordination between *Model* and its *View* so as to keep a consistent context for the system state.

The workflow of an interaction shows the dependency relationship between the components as marked in fig. 2.5. The process may involve multiple steps in an interaction cycle.

1. It starts with a user input event (e.g. mouse click or key stroke).
2. The *Controller* receives and interprets the event and then authorizes the *Model* to change.
3. The *Model* updates its state through modification of its data structure and broadcast its completion to both *Model* and *View*.

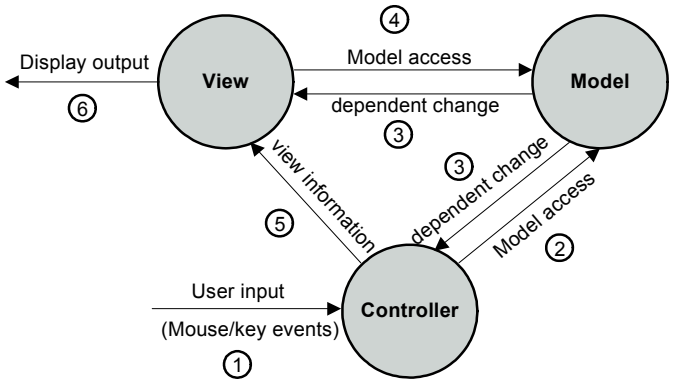


Figure 2.5 MVC model

Note that in different implementations, the *View* can obtain the updated rendering information directly from the notification in step 3.

4. Or request from the *Model* after being notified of the change by the *Controller*.
5. Or through the *Controller* as in step 5.

6. The *View* renders graphics and/or text image buffer reflecting the change accordingly as the last step.

Krasner and Pope delineated such MVC interaction process [Krasner+Pope] using the FinalcialHistory example from Goldberg's MVC tutorial.

The MVC model embodies a system design principle ─ namely, dividing a complex system into smaller subsystems. The separation of MVC produces a coherent triad: *Model*, *View*, and *Controller*, which follow the disciplines of software programming for modularity. Inherently, it promotes component reusability of the *Model* and helps good system development and maintenance of these modules. The principles of original Smalltalk design suggest a "single paradigm language with very simple semantics and syntax for specifying elements of a system and for describing system dynamics [ANSI SMALLTALK]."

2.2.2 Communication mechanism ─ method-based versus message-based approach

A key to MVC is an effective event handling mechanism for the communication between Model and View components. Since user input triggers mouse (or key) events at GUI, the Controller component is responsible for forwarding these events to the Model and transforming them so as to invoke execution of methods that modifies data structure. MVC is a common event-based programming model for interactive applications.

We classify the communication mechanism of MVC into two types: one is "method-based"; the other is "message-based". How much difference between method-based MVC and message-based MVC?

Method-based MVC works with the Model and the View communicating through method calls provided by underlying system that plays the role as the Controller. In a desktop Windows application, Windows operating system acts as the Controller and provides an event handling mechanism for invoking application Window's call back method on receiving of the Model change. The system puts an event (e.g. mouse click) message in the corresponding application's message queue. Target application Window invokes methods of the application's functional core on receiving of notification of an event occurrence. The messages are hidden in operating system level so as to achieve optimized performance result within tight-coupled components. A MVC Web application built on RMI uses "skeleton" and "stub" interface to establish interactions between Model and View with RMI framework being the Controller. In this scenario, the Controller is implemented in a Remote Procedure Call (PRC). For both case, either classic method call or the distributed version of RPC, are method-based.

Method-based MVC applications are typically targeted for a specific system (e.g. RMI and Windows) that designed for the optimization of individual platform functionalities and performance. Therefore, these applications are not directly portable with each other and do not scale well.

Message-based MVC tends to minimize minimizing explicit or implicit context so as to decouple components and enable messaging logic separating from specific implementation. It employs messaging for the exchange of context information between the Model and the View components. The messaging mechanism can be as simple as using HTTP protocol to deliver HTML page from an Apache Web server to a client browser, or as sophisticated as messaging middleware systems such as NaradaBrokering

that provides various messaging services including multiple transport protocol (e.g. UDP, TCP, and HTTP), publish/subscribe, fault-tolerant, and security.

If borrowing the concepts from shared memory versus distributed memory of parallel computing models to characterize method-based versus message-based approach, the design tradeoffs are optimized for performance of the former and for distribution and portability of the latter. The essential difference between method-based and message-based communication approaches are as follows: RPC corresponds to taking a traditional method-based application and executing it in a message-based fashion. Its goal is to distribute parts of an application and implement the method calls with messaging. It has the implication that the linked parts share both interfaces and internal class structure. Service oriented architectures (SOA) focus just on the messages and the interfaces they link, where the interfaces are thought of as contracts and not as traditional method signatures. There is no implication that there is any correspondence in internal structure between the two linked services even if the messages are used in a RPC request-response pattern. Conceivably, message-based approach is going to surpass method-based approach in future distributed systems, especially with the rapid growth of computer processing capability and network bandwidth that compromise performance disadvantage.

2.2.3 Decomposition strategies

A decomposition strategy refers to the way that a system is divided into functional modules. Model-View-Controller (MVC) is one way to decompose system into sub-components. It was originally introduced to decompose a GUI desktop system into three modules: the Model serves as computation core of the system; the View represents visual

components; Controller conveys interaction between the above two. MVC is a frequently used as the observer design pattern [J.COOPER] in object-oriented programming.

Because interactive style application commonly has GUI with complex composition and corresponding graphics rendering issues, the logic separation of Model from View implies that the Model (computation core) is encapsulated and does not have to interact with user interface directly in any way. Hence, it makes coding more flexible and maintenance relatively easy.

Although the definition of the Model, View, and Controller triad is relatively clear, the composition of these components is somewhat dependent on the eye of beholder. For instance, a simple text editor may contain string data structure for the Model and its display for the View. In a client/server application (see fig. D.1), Web server and Web browser forms the front tier and back end, entailing Model and View structure. JavaBeans and JSP, which are building blocks of JSP Model 2 architecture (ref. fig. 3.5), represent corresponding Mode and View constituents on server side. In our experiment of Batik SVG decomposition (ref. fig. 6.2), DOM tree is the Model while GVT tree and rendering constitute the View. Particularly, we show in fig. 3.3 that there are different division possibilities even with the same application.

2.2.4 Interactive patterns

Interactive pattern defines how Model and View components are organized to form a system structure. In SmallTalk80 [Goldberg+Robson], the interactive patterns was deployed with multiple Windows layout (View) shares the same data structure (Model). In typical client/server Web applications, multiple Web browsers (View) share a file server (Model) to retrieve Web contents. For complex environments like J2EE and .NET,

distributed components (Model) can be separately configured for corresponding customized client interfaces (View) such as mobile devices.

2.2.5 Summary

MVC is a widely used paradigm for interactive applications that separates presentation from data structure and behavior. This section attempts to introduce the key concepts of MVC from the view of basic architecture elements: communication mechanism, decomposition strategies, and interactive patterns. Further analysis of Variant MVC approaches that include legacy desktop system, homogenous distributed system, and (Web) service-oriented system is given in section 3.5. A summary of representative application models such as Microsoft Windows, distributed Smalltalk, J2EE (JSP), REST, and M-MVC are also provided with elaborated discussion.

2.3 Messaging

2.3.1 Why messaging?

Messaging provides a ubiquitous solution for all people and all forms of communication by exchanging of message information among participating entities. The term “messaging” subject to a wide band of definitions for describing various systems such as email, online chat, voice-enabled communication. Here, we define messaging as an asynchronous communication approach between computer resources.

Messages are used in a broad based. In this thesis, we refer to the term “message” as formatted information that coordinates between source and destination software application entities. The content (or payload) of a message are comprised of diverse data types, such as image files, text documents, and audio/video streams.

With the augment of Web and increasing complex of software applications, there are compelling reasons as to why it is important to deploy messaging as the communication scheme for Web-based applications, which we will elaborate in the following perspectives: Web applications and communication features, Internet structure, and IP traits and limitations.

Web applications and communication features

Communication aspects of computer technology have become an incredible important evolution area. The broadband data communication provides high-speed Internet access through means of options such as cable, phone line, wireless, and satellite. The array of communication forms, which are integrated into the global information infrastructure, makes it possible for millions of households to access the Internet through an effective and practical manner. This development provides the framework for deploying assorted real-time collaborative Web applications enabling a futuristic multimedia enriched and highly interoperable Web, such as Voice-over-IP, video-on-demand, interactive TV, rich graphics and animation online game over mobile devices, and virtual enterprise support. The perspective, in turn, demands reliable and efficient communication services for prevailing data or media stream exchange between distributed entities.

The early Web applications were deployed using client/server model (ref. Appendix B and D). Examples like email and Web browser typically sent a request to the remote server and receive a response message that embraces text or a mix of text and images. The Web server can extend its access to backend database or business logic, which forms a three or multi-tier model. However, traditional centralized client/server model presents

an approach with limitations to simple point-to-point connection and can no longer keep up-to-date with a new set of application requirements towards general distributed features of complexity and heterogeneity.

Middleware, which encompasses distributed Tuples, RPC and RMI, distributed object model like CORBA and DCOM, and message-oriented model such as JMS [BAKKEN], provides a common infrastructure for large scale distributed applications. For instance, enterprise applications commonly build on top of J2EE and .NET that are updated versions of middleware systems. However, RPC based middleware systems are based on callback methods in a distributed environment, which are typically with synchronous and closely coupled implementations. They have limited parallel support and exception handling facilities.

Different forms of group communication and real-time collaboration systems have been investigated as natural models better served in simulation of information dissemination patterns in the experience of daily life. For instance, broadcast type of group communication allows distribution of media content from one source to multiple recipients (e.g. Internet TV). Peer-to-peer (P2P) model provides a dynamic framework with virtual online community sharing of information in a decentralized self-organizing fashion (e.g. P2P MP3 file system). Message-oriented middleware (e.g. JMS, WS-Notification, and WS-Eventing) is designed at application level and support multicast style communication (e.g. publish/subscribe) as well as unicast (point-to-point) and broadcast patterns.

Internet structure

Internet is a network of interconnected networks (ref. appendix C). The variety of networks may build on top of directly connected physical network such as Ethernet [ETHERNET], token ring [TOKENRING] and wireless [WIRELESS]. They may have their own media access protocols, addressing scheme, and service model. Building on top of packet-switching technologies, Internet extends its linkage to provide a Wide Area Network (WAN) by connecting the heterogeneous autonomous Local Area Network (LAN) together. This hardware infrastructure makes up the network core of the global information communication platform, as we illustrated in fig 1.1. At the meantime, a wide band of communication protocols composed of LAN protocols, WAN protocols, routing protocols, and network protocols provide conceptual layers supporting information dissemination between computer resources scattered along the edge routers.

However, the Internet evolution presents an elusive approach ─ it does not adhere rigorously to the seven layered Open systems interconnection (OSI) [OSI] model, which was developed by International Standards Organization (ISO) attempting to provide some standard framework of networks; rather, it has a conceptual architecture composed of TCP, HTTP, UDP over IP. This structure has been challenged, although the Internet protocol stack has evolved into stability at low level that is equivalent to the three lowest level of OSI [OSI] (ref. fig. C.3). As an important evolving area for emerging technologies, messaging middleware provides a generic framework above the IP layer and below the application layer, which embraces publish/subscribe service to distributed systems between dissimilar networks and software components.

Internet Protocol

Over the time, the evolution of Internet Protocol (IP) [IP] as the de facto and the foundation of the Internet since IP protocol paved the way for the Internet being an information infrastructure that empowers dispersed computers not only as computing devices but also as communication devices. It defines how to route data frames or packages from host to host and supports multiple networks interconnected into a single, logical network (appendix C). IP connects diverse IP-based networks and has evolved to the de facto low level software communication protocol of overall network protocol domains.

Unfortunately, the necessary capabilities needed to realize multicast are not adequately supported in the current networks. The IP multicast solution has serious scaling and deployment limitations, and cannot be easily extended to provide more enhanced data services. IP was originally designed as the best-effort protocol that ensures reliable delivery of a packet to its destination, which lacks incentives of critical services for applications such as Quality of Service (QoS) [QOS] providing delivery assurance. This is due to the features of earlier network infrastructure with limited abilities of switching technologies (e.g. best effort delivery) at the hardware level and primitive designated functionalities (e.g. TCP providing reliable transportation but time out causing lost of data) that deployed at application level. With the deployment of applications like internet telephony, streaming media, which are delay-sensitive, it is in need of transportation for these applications with traits of end-to-end QoS assurance, which can be provided by messaging services deployed at application level.

As a distinctive feature, the messages are time stamped and ordered, which facilitates various communication services with properties such as QoS, Robust Delivery

mechanism (e.g. fault tolerance), storage, and support for P2P interaction. These capabilities suffice the demands of diverse applications and are especially important for time-critical collaboration systems. Additionally, the form of rich messaging services supplies a far more flexible and scalable framework over heterogeneous environments than individual platforms.

2.3.2 Messaging Middleware

Messaging middleware provides a general communication channel which consists of software host-to-host channel and Process-to-Process channel and facilitates the sharing of various resources over a heterogeneous environment. In a publish/subscribe messaging service scheme, senders label each message with the name of a topic ("publish"), rather than addressing it to specific recipients. The messaging system then sends the message to all eligible systems that have asked to receive messages on that topic ("subscribe"). This form of asynchronous messaging is a far more scalable architecture than point-to-point alternatives such as message queuing, since message senders need only concern themselves with creating the original message, and can leave the task of servicing recipients to the messaging infrastructure. It is a very loosely coupled architecture, in which senders often do not even know who their subscribers are.

In conjunction with event-based programming, publish/subscribe provides a mechanism that allows event producer to consumer(s) of event occurrence. Publish/subscribe messaging service has become a widely used programming style that supports interrupt-handling mechanisms for user input-device interactions. In parallel computing, MPI [MPI] and PVM [PVM] provided “all the features one needed” for inter-node messaging. NaradaBrokering aims to play the same role as a messaging

infrastructure support Internet and Grids systems. However, the requirements and constraints are very different. An Internet messaging infrastructure provides a seamlessly communication channel with reliability and substantial flexibility services to application-level deployment.

Message-oriented middleware (MOM) provides a new powerful messaging paradigm that makes it easier to uncouple different parts of an enterprise application. Messaging clients work by sending messages to a messaging server, which is responsible for delivering the messages to their destination. Message delivery is asynchronous, meaning that the client can continue working without waiting for the message to be delivered. The contents of the message can be anything from a simple text string to a serialized Java object or an XML document. Messaging is often used to coordinate programs in dissimilar systems or written in different programming languages. Most systems use the event-based messaging paradigm for capability such as notification, publish/subscribe, and direct socket-to-socket transport. Apart from NaradaBrokering [NARADABROKERING], examples of MOM that support publish/subscribe interface include OMG Data Distribution Service for Real-Time Systems [OMG-MESSAGING]), and Publish-Subscribe Notification for Web Services [PUB/SUBNOTIFICATION].

2.3.3 Summary

We overview messaging as a key communication technology to improve interactions, reduce overhead, enhance work-process efficiencies, provide Quality of Service (QoS), reliability, scalability and facilitate information sharing across Internet. Messaging provides assurance that diverse data steams tailored for different applications can be reliably and efficiently transported to their destination over the Internet. Messages

contain contextual information (e.g. events) between distributed entities and allow virtual addressing (indirectly via publish/subscribe scheme) that enables dynamical binding to diverse evolving hosting environments for the communication between source and destination in a loosely coupled manner. Based on our framework of building Web applications centered on messages, event-based programming is an approach well-suited for development of synchronous and asynchronous interactive systems. In the following section, we introduce NaradaBrokering messaging system that provides the underlying communication infrastructure for our message-based applications.

2.4 NaradaBrokering

NaradaBrokering [NaradaBrokering] is an event and messaging infrastructure that can manage the unicast and multicast delivery of messages between the different processes. NaradaBrokering copes with multiple protocols (e.g. TCP/IP, UDP, HTTP, SSL, RTP, GridFTP, and HHMS) and tunnels through firewalls. It uses XML-based publish/subscribe semantics generalizing that are familiar from JMS (Java Message Service) [JMS] and we have shown this conveniently supports collaboration. Each shared object corresponds to a topic and NaradaBrokering manages the associated topic-labeled event streams with high performance and reliable messaging.

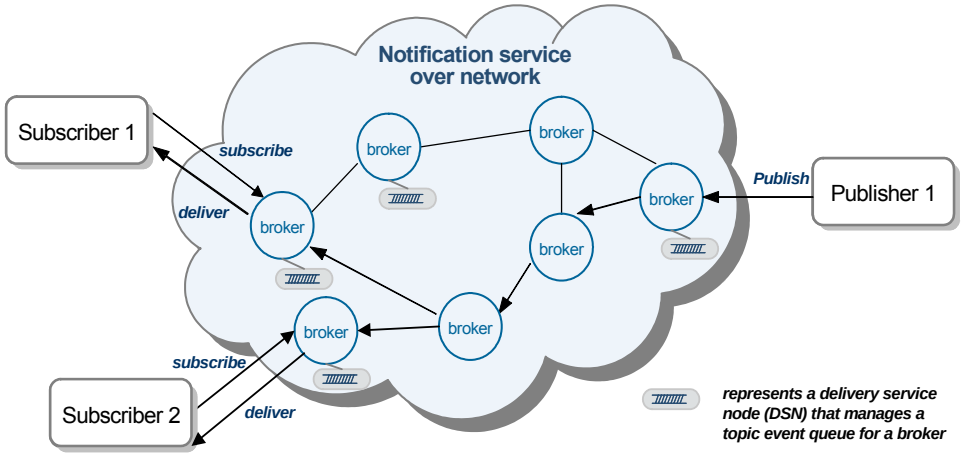


Figure 2.6 Architecture of Publish/Subscribe model based on **NaradaBrokering** event broker notification service

Fig. 2.6 depicts the relationship between publisher, broker, and subscriber elements that based on NaradaBrokering notification service [NARADABROKERING]. The brokers are organized in hierarchical clusters for scalability and routing efficiency [PALLICKARA-06-04]. Each broker can attach one to multiple delivery service nodes (DSN) that manage the topic(s) and queue(s) for events. In order to establish a publish/subscribe relationship (as shown in fig. 2.3) under a common topic ─ "topic A", publisher 1 and subscriber 1 and 2 may initially connect to a local broker for registration. Then, system routing algorithm assigns a published message is routed along a chain of routing brokers which forms a virtual path, and eventually reaches to the destination resources across the network. NaradaBrokering maintains routing tables to record and update the latest network configuration (e.g. assignment of brokers and load balance monitoring). These tables are used starting from local cluster, and propagated to a remote broker in a larger cluster domain, to publish an event message to remote subscribers.

2.5 DOM

2.5.1 DOM, HTML, and XML

Internet-based Web applications started from World-Wide Web (see Appendix D). The key technologies of WWW comprises of accessing HTML [HTML] contents via HTTP [HTTP] protocol and viewing them from different client user interface. HTML defines abstract information in text tags. Document Object Model (DOM) [DOM] provides "an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated".

DOM was the descendent of "Dynamic HTML" [DHTML] and originated to allow JavaScript scripts and Java programs to be portable among Netscape [NETSCAPE] and Internet Explorer [IE] version 4 Web browsers. At the time, web developers were able to make HTML pages and adding animated interactive scripts for retrieving document on remote web servers. Nevertheless, the content is manipulated with different implementations of document object model in these earlier version browsers. Developers accessing HTML document with JavaScript [JAVASCRIPT] had to write wrapper code to reconcile the incompatibility between Netscape and Internet Explorer. In 1998, W3C proposed Document Object Model (DOM) level 1 specification [DOM1] that defines "a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents", which provides support for HTML 4.0 [HTML] and XML 1.0 [XML]. DOM level 2 standard further specifies a generic event model [DOM2EVENT]. Version 5 browsers (Mozilla NGLayout engine (Gecko) [MOZILLA] and Microsoft Internet Explorer 5) are implemented in support of

the DOM level 2 specifications. Up till now, DOM has been updated with level 3 enhancement versions.

DOM is important because it enabled far more than building of cross-browser dynamic pages ─ it is an open standard with a generic hierarchical model that defines and facilitates manipulation of structured information or object. DOM carefully defines "just" the logical structure of "document" and an API (application programming interface) in Object Management Group (OMG) IDL [OMG IDL]. Such a standard can effectively specify HTML and any XML [XML] describable information. Particularly, it means that it can represent the structure of Meta data abstracted by XML schema for any object including distributed software component. As specified in DOM, it allows any language-specific bindings and can be implemented with language-independent system such as COM or CORBA. As examples, Java and JavaScript are provided as binding cases with the specification. Therefore, DOM is regarded as a specification that can be used by variety of environments and applications. We expect that DOM model will be applied more extensively as a framework in building of future distributed systems.

The term of "document" is used here for defining any structured information or object in an abstract manner. The concept of "object" has been used in a broad spectrum: in lieu of the Internet and Web, we take the view that anything having to do with resources ─ no matter it is in form of data, text, image, audio/video stream, MP3 music, software, printer, fax machine, sensor, and even human being ─ is an object; in terms of DOM, an object implies a document node that has both data structure and semantics of attribute and behavior, which compatible pretty well to the naming of "document object model". In the following subsections, we give further discussion of two prominent

aspects of DOM: the hierarchical structure of the organization and the event model for interoperability.

2.5.2 DOM structure

DOM has a logic structure to represent a document in resembling the relationship of composed objects. A hierarchical tree-like model is used as DOM model while it can be employed in many other alternatives. Document is used as an indispensable concept associated with information, resource, or object. It encompasses content and presentation two parts. Document content with well-defined structure provides efficiency for access, manipulation, storage and interpretation. Presentation allows tailoring content's rendering features such as positioning, coloring, and fonts in visual browsers, aural devices, printers, and handheld devices etc.

Since the tree structure is a fundamental topology widely used as the data structure or the object model by various systems, DOM model has general indication and applicability as an effective mechanism for complex systems in organization, management, and manipulation of constituent objects. Although naming and implementation details may vary, numerous applications, either proprietary or open source, are deployed using tree-like objects model for documents. These systems include common file system (e.g. UNIX), Smalltalk, OpenDoc [OPENDOC], OpenOffice [OPENOFFICE], OpenInventer [OPENINVENTER], VRML [VRML], Java AWT [JAVA AWT] and Swing [SWING], COM [COM], and OLE [OLE].

Node is the primary data type of document object. In another words, all objects of DOM document implement the Node interface. There are twelve node types defined in DOM (Core) Level 1 specification [DOM1CORE], which include document as the root

node while element, text, entity, entity reference, and processing instruction etc. as child nodes (see fig. E.1). Except for text node, element node is the most frequently used DOM node and often associated with attributes. For visual convenience, we use following rectLinking.svg example in fig. 2.7 to describe how DOM is employed to represent structured XML information.

```
1  <?xml version="1.0" standalone="no"?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
   "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd">
3
4  <svg id="body" width="300" height="350" viewBox="0 0 300 350">
5
6     <g id="content" transform="translate(0,0)">
7
8        <text x="10" y="30" class="title" style="fill:black">Click on rectangle linking</text>
9
10       <a xlink:href="http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/">
11          <rect id="targetRect" x="30" y="40" width="100" height="60" style="fill:blue;" />
12       </a>
13
14    </g>
15
16 </svg>
```

Figure 2.7 A XML document rectLinking.svg with hyperlink element

The corresponding DOM tree representation is displayed in fig.2.8. It shows that the first node to encounter is the root document node when traversing a document. There are three child nodes of the root node: ProcessingInstruction node that indicates processor-specific information; DocumentType node providing doctype attribute such as DTD entity that defines logic structure for the document; element node with svg content body. Within the container g element, there’re text element and element 'a' that references to a predefined hyperlink. One complication is that a graphical rect element is used as the target element nested within the simple link element 'a'. In implementation, the mapping between flattened XML document for storage and corresponding DOM tree objects model in memory can be achieved by using XML parsers and serialization utility classes respectively.

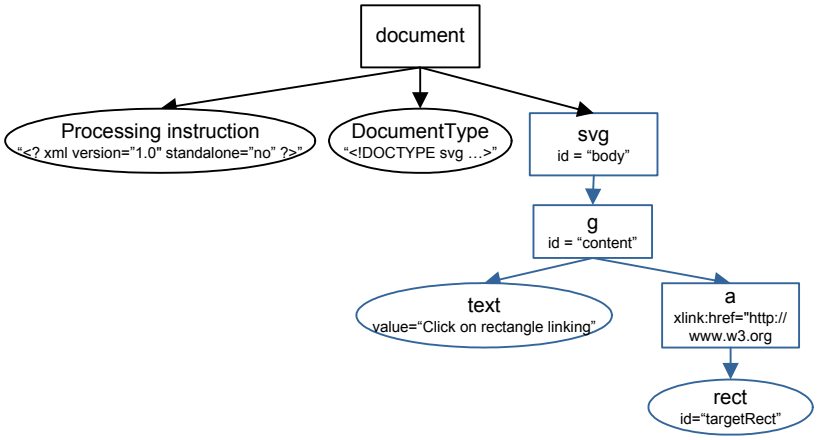


Figure 2.8 DOM tree representation of rectLinking.svg document

Although the structure of a real application would be orders of magnitude more complicated, the example shows that the DOM may be used to manage XML documents in a hierarchical tree model that resembles document structure of the objects. As DOM specifies just an interface (like interface class in Java and abstract class in C++) to access and manipulate of the internal representation of corresponding applications, it is up to specific implementations that define the semantics of the content. For instance, the document object may be composed of a graphics, a text, a file, an audio/video stream, or a software component. It the last case, DOM is a well-suited distributed object model. Software programs, whether building around DOM or wrapping an interface over existing assets that support of the DOM, will have full DOM features available provided by the mechanism.

2.5.3 DOM event model

As discussed earlier in section 2.1 of event based programming, an event system is the backbone of interactive applications. The DOM event model specifies rules and patterns of how to exchange and propagation information through events that drive

system behavior. W3C defines the DOM event model [DOM3EVENT] as “a generic platform- and language-neutral event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event”.

DOM event model [DOM2EVENT] provides two types of event-based communication: one is the exchange of contextual information between DOM node and external event handlers’ class; the other is event propagation along nodes of the document tree structure.

As a primary event handling mechanism, the former is derived from the generic event/listener model (see fig. 2.1). It allows external components (the event listeners) to associate with any individual node (event source) in the DOM tree structure such that they get notified when an event is dispatched. To achieve this, DOM event model specifies EventTarget interface as shown in fig. 2.9. Each DOM Node that implements the interface inherits the capabilities for the registration of EventListeners and dispatching of event at the target Node. Since event contains contextual information, the event model supplies an effective communication channel between internal DOM data structure and external components.

```
interface EventTarget {
  void            addEventListener(in DOMString type,
                                   in EventListener listener,
                                   in boolean useCapture);
  void            removeEventListener(in DOMString type,
                                      in EventListener listener,
                                      in boolean useCapture);
  boolean         dispatchEvent(in Event evt)
                                   raises(EventException);
};
```

Figure 2.9 Interface of EventTarget

The other event process is the internal event flow along the DOM tree. Since DOM nodes are organized in a hierarchical parent-children relationship and events can be forwarded from the root node downward to the event target node or in an opposite upward direction, the DOM event model offers two methods of event flow. One is "event capture" that propagates downward and notifies any registered EventListeners of corresponding type of events along the path. The other is "event bubbling" that starts invoking event listeners from the event target node and then propagates up parent chain to the document root. An event listener can call "stopPropagation" method of the Event interface to prevent any further event propagation. It can also choose to cancel default action associated with the dispatched event.

Taking a common interactive application ─ Web browser for example, Netscape 5.0 and Internet Explorer 5.0 have been designed to support DOM level 2 Event model [DOM2EVENT] that specifies the operations and queries that can be made on a HTML (or XML) document. For instance, hyperlinking is a frequently used interaction for browsing webpages. The basic user input is "mouse click" on Web links that defined by 'a' element so as to retreive hypertext (or hypermedia) documents that are portable from one platform to another. Considering a simple blue rectangle associated with a hyperlink to a DOM website in the example displayed in fig 2.7, which can be tested by loading the

file “rectLinking.svg” into a SVG enabled Web browser. A mouse click on the blue rectangle would invoke downloading a new webpage.

An event flow chart is provided in fig. 2.10 to illustrate underlying event process of the hyperlinking interaction via the DOM event model. The capability can be achieved by using two basic DOM event interfaces: one is communication between external event handler module and internal DOM element (e.g. link element 'a') where event listener is registered; the other is event propagation along DOM tree structure, which can use either capture or bubbling model depending on the direction of event flow (downward or upward) between parent and child node.

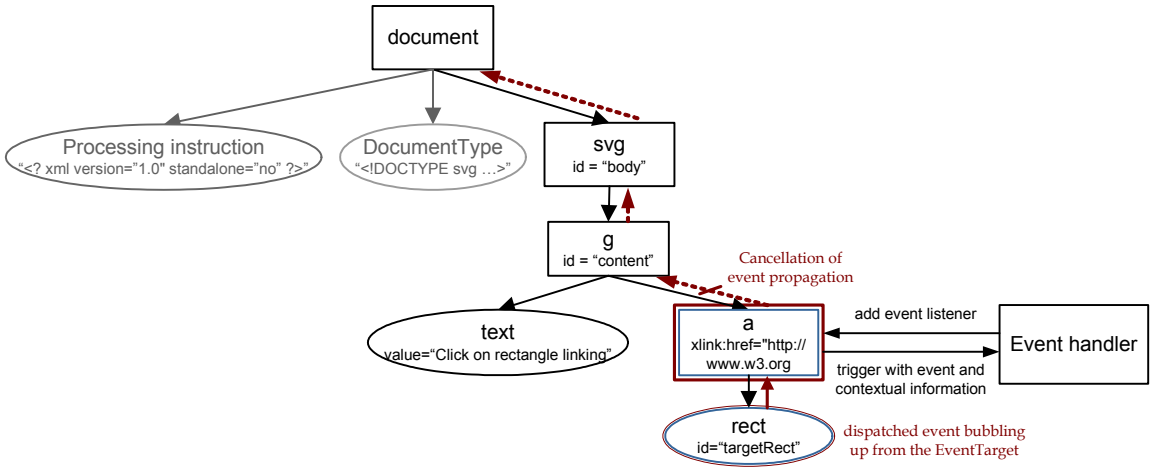


Figure 2.10 Event handler registration and event flow of DOM in a case of rectLinking.svg document

As the example shows, a bubbling process is employed with the event target object ('rect' element) forwarding contextual information up to its parent link element. In fact, any event listeners added to a node along the path of event propagation up to the root document element would be triggered unless “stopPropagation” operation is performed. In the graph, event target node 'rect' is trigged by an UI event (when user clicks on the rectangle object in Web browser) and pass it on to its parent node. The node 'a' invokes

event handler module but chooses to cancel further event propagation. Conceivably, the event handler module would conduct hyperlinking operation by fetching URI (Uniform Resource Identifier) referenced document from a remote server and rendering it as a new page in the Web browser.

2.5.4 Summary

DOM originated from a concept of cross browser document model for HTML and XML but has extended to be a generic platform and language independent programmable interface for distributed interoperable systems. Through a XML document with link element, we've described how essential DOM interfaces ─ the DOM tree structure and the DOM event model are coordinated in support for an interactive Web application to accomplish common user interactions (e.g. hyperlinking in a Web browser). Particularly, the DOM event model supplies a powerful platform of event-based communication that controls event flow over orthogonal directions: one is a derivative event/listener model (like the java delegation model) that provides interoperability between DOM component and external component; the other is internal event propagation along the document tree structure for exchanging contextual information between DOM nodes. All together, the DOM promotes a generic framework that encompasses an effective object model and interoperability mechanism for complex systems.

Chapter 3

M-MVC Architecture

As in many ways, this dissertation is about designing of software architecture, the sections of this chapter add greater background context for the incentives, key concepts, and principal building blocks associated with our system architecture in composition and interoperation. Specifically, we analysis characteristics of distributed applications based on event-based system interactions; discuss message-based MVC (M-MVC) as a uniform architecture of distributed and desktop applications; an Internet collaboration framework with SMMV and MMMV as general Web Service paradigms for collaboration. Related work that encompasses various MVC approaches is examined. We describe how M-MVC is deployed with messaging infrastructure in a Publish/Subscribe scheme; additionally, we give a brief introduction to double-linked multiple-stage pipeline model and a summary of collaboration framework. This chapter combines with the next five chapters to act as a foundation for the conclusions that follow.

3.1 Characteristics of distributed applications

3.1.1 Human computer interaction

ACM SIGCHI defines human computer interaction (HCI) [HCI] as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them." As

an interdisciplinary area, HCI has a broad relationship with computer science, psycology, sociology, and industrial design.

It is of intrinsinc research interest for us to look into how human and machine factors affect application design and engineering. Many morden software applications inevitably have to build interface components to interact with a user. Prevailing interactive Graphics User Interface (GUI) components commonly represents a large propotion of overall system code lines. In addtion, performance and reliability issues arise due to spatial distance and bandwidth constraint that are particularly challenging for building distributed application. We try to understand perspectives of user interaction pattern, user interface design, system functionality and behavior, the communication structure between user and computer, environment settings, and the impact of these factors on design trade-offs, implementation, and testing model of system efficency and viability. For instance, visual persistence (a human visual phenomenon of perceptual sensitivity to latency) generates tremendous impact on interactive applications and we will discuss this in section 7.2.3.

There are many theoretical approaches to the studt of HCI in research area of Cognative Psycology [CARROLL]. “Interaction Analysis” is a method that studies human interactivities based on social science [Jordan+Hendersion]. It is difficult to draw design principles and other abstract lessons from a technique that is concerned with detail of a particular situation [Viller+Sommerville]. Thus, it is essential and helpful to extend the survey to a variety of distributed applications in order to generalize interactive design features. In following subsections, we focus on human context of utilizing Internet and Web technologies in terms of social behavior and interaction to a computer in section

3.1.2 and give further summarization of the characteristics of typical distributed applications based on "events" ─ a quantitative measurement of system interaction in section 3.1.3.

3.1.2 Distributed application and user interaction

Nielsen's reports reveal that more people are using the Internet than ever before: in 2004, three out of four Americans have access to the Internet from home [NIELSEN]; an estimation of total online population worldwide reaches over 800 million [INTERNETWORLDSTATS]. There is a diverse range of distributed systems or Web based applications for the sharing of distributed resources, which includes email, file transfer, Web browsers, distributed simulation, distributed database, instant messaging (IM) [IM], voice-over-IP, blogging [BLOG], video/audio conferencing, Web-based media-on-demand (e.g. playing MP3 music or movie on PC or even cellular phone by downloading streams or files), interactive online game, participatory learning tools (e.g. whiteboard), virtual enterprise, large-scale distributed computing, and many others covering areas of e-Science, e-Learning, e-Business, entertainment, and general purpose communication.

The following lists typical sample applications in table 3.1. The semantics of user interaction define the primary interactions that a user invokes. The mode property that is marked with positive or negative sign ("+"/"–") delineates whether or not a user interaction causes modification of the system model or original data structure. Major capabilities are provided in the system behavior field. Two types of communication ─ asynchronous and synchronous are defined here to distinguish whether the communication requires participants to present over a period of time or around the same

time. Although there isn't a strict line between them, it is reasonable to expect that a synchronized interaction will complete within a few seconds' time so as to achieve a real-time experience. For example, email is deployed as an asynchronous communication since it is not necessary for a receiver to access the mail immediately after receiving it from the sender; an online chess game has two players involved in a timely interactive manner, which is considered as a synchronous approach.

Table 3.1 Typical distributed applications and properties of user interaction, system behavior, and communication

	User interaction		System behavior	Communication	
	semantics	Mode		asynchronous	synchronous
Email	send/receive email	+	store and forwarding email	√	
Shared File System (P2P)	download files	–	document management and distribution	√	
Instant Messenger	online chat (e.g. text)	+	sharing text information		√
Distributed Simulation	n/a	n/a	computation and synchronization		√
Internet TV Broadcast (Streaming Media)	download and play media	–	live broadcast of media	√	
Video/Audio Conferencing	online meeting	+	session management and real-time sharing of video/audio streams		√
Shared Browser	browsing web pages	–	session management and real-time sharing of web pages		√
Shared Whiteboard	editing text and graphics	+	session management and real-time sharing of graphical editing tools		√
Multiplayer Online Game (e.g. chess)	move pieces	+	session management and real-time sharing of interactive game		√
Parallel Computing	n/a	n/a	computation and synchronization		√

3.1.3 Characteristic of events and distributed Applications

Computer-based technology started with computing but has evolved to empower both computing and communication capabilities (ref. Computer-based computing in

appendix A). The advancement of broadband and Web application technologies open up new vistas of communication, collaboration and coordination. While it is hard to compare different aspects of disparate applications, we find that it is possible to summarize these systems based on the fact that they have common features ─ "sharing" of distributed resources, which are dominated by factors such as the nature and intensity of event-based interactions; structure and features of resources or information for the sharing.

Here, we use "events" to refer to any forms of initiation or driving power for system interactions and changes. As an example, an event may be defined as an input device event (e.g. a mouse click or a key stroke from GUI) for an online game application. Because system interactions are mainly conducted through events at different semantics and level of granularities, we separately define some typical events such as "MacroEvent" for major system interaction and "MicroEvent" for a small scale semantic measurement in Table 3.2. Then, we list some applications in a comparison table (see table 3.3) to highlight their characteristic centered on these events.

Table 3.2 Definition of typical events of distributed applications

	MacroEvent	MicroEvent
Email	writing an email	a key stroke
Instant Messenger	writing one line of message	a key stroke
Shared File System (P2P)	downloading a file	n/a
Distributed Simulation	messaging passing (exchanging a message between components)	n/a
Internet TV Broadcast (Streaming Media)	one-way buffering	inter-frame delay
Video/Audio Conferencing	multi-way buffering (interactivity)	inter-frame delay
Shared Browser	loading a new URL	a mouse click on a hyperlink
Shared Whiteboard	drawing a new graphics component (e.g. a rectangle, a line, a path)	a mouse movement
Multiplayer Online Game (e.g. chess)	moving a piece	a mouse movement
Parallel Computing	updating a region held in a single node	message passing (exchanging a message between

		components)

As distributed applications comprise interactions within internal constituent components and with external applications through frequent exchange of event-based messages, event plays an important role in the deployment of interactive distributed systems. Every application has its own event semantics corresponding to different level of measurement of the system changes and associated level of complexity in handling of system behavior. Event model defines communication scheme and controls event flow and propagation. The degree or tightness of coupling among different parts of an application is indicated by nature of system interaction, semantics of inherent event, and communication overhead (including network latency of these events) that have significant influence on the overall system design and functionalities.

Table 3.3 allows some interesting observations. Several system aspects are compared in terms of content and rendering features, timing of event-based interactivity, and associated network connection issues. Applications work on different scope of data set ─ some are simply byte streams like in shared file system and parallel computing; some are in text, sound, graphics, or hybrid forms with rendering complications especially for media rich content. The extra rendering cost includes various graphics processing (e.g. filtering, mask effect, and rasterizing of vector graphics) and codec (compression and decompression) effort.

Table 3.3 Summary of typical distributed applications and characteristics

application	Content		avg. min level of interactivity		network connectivity features			
	type	Rendering complexity	Macro Event	Micro Event	band-width	latency tolerance level	reliability	connectivity type
Email	text, image	low	min	milli sec	low	minutes or above	no	point-to-point

Instant Messenger	text	low	sec	milli sec	low	second	no	multicast
Shared File System (P2P)	byte stream	n/a	min	n/a	high	minutes or above	yes	multicast
Distributed Simulation	byte message	n/a	sec	n/a	high	100's millisecs	yes	point-to-point
Internet TV/Broadcast	image, sound	high	sec	.033 sec	high	second	no	broadcast
Video/Audio Conferencing	image, sound	high	sec	.033 sec	high	100's millisecs	no	multicast
Shared Whiteboard	text, image	high	sec	milli sec	low	10's millisecs	yes	multicast
Multiplayer Online Game	text, image, sound	high	sec	milli sec	low	10's millisecs	yes	multicast
Parallel Computing	byte message	n/a	large	> 10 microsec	high	10 microsecs	yes	point-to-point

As listed earlier in table 3.1, the "sharing" can be done asynchronously and synchronously (usually referred to as "collaboration") among participating parties within a system. For instance, an email application typically provides text–based message storage and forwarding service in loosely-coupled asynchronous manner while an instant messenger (IM) is an instant messaging service that allows multiple clients hook up to it for real time conversations. Also, collaboration services, such as IM, require session control that hosts and manages the online status of the users.

As summarized in table 3.3, the deployment complexity of a synchronous collaboration system is mainly decided by synchronization granularity and timing. Typically, lower granularity corresponds to more stringent timing constraints. Specifically, the timing of system features, especially their interactivity and fulfillment of collaborative functionality, influences the deployment. For example, sharing of a presentation style system (e.g. graphics authoring tool such as whiteboard) is more complicated than that of a text based application like instant messenger; while shared browsers with collaboration functionality of interactive animation presents greater challenges than sharing of a Web page change.

To guarantee functionalities, synchronous collaborative systems (e.g. video/audio conferencing and multiplayer online game) have to deal with complex synchronization problem while this is less constrained for loosely coupled asynchronous systems (e.g. email and newsgroup). For instance, to achieve near real-time experience, network latency tolerant level for collaboration system is typically around hundreds of milliseconds for video/audio conferencing; tens of milliseconds for intensive interactive applications such as multiplayer online game and collaborative whiteboard. Currently, network transit times of transcontinental links are 100's of milliseconds while local area network including intranet of organization area could achieve a few milliseconds in latency.

3.1.4 Summary

The analysis in this subsection aimed to provide a common ground for bridging diverse interactive applications with a unified event-driven message-based architecture. We've examined scenarios of typical distributed applications in terms of user interaction, system behavior and communication. The feature table 3.3 further summarizes characteristics of these applications based on event interactions in a quantitative manner.

3.2 Web Service pipeline model

In the Web Service pipeline model [Fox03], the workflow of a Web application is marked as a chain of objects linked with bi-directional connectors ─ in which events flow and rendering results counter flow from resource to client end. In this context, an "object" is literally a software component that implements certain functionality with input and output connections to neighbor objects. The "connector" is event-driven message-based

linkage between adjacent objects. This basic double-linked multiple-stage pipeline structure, as depicted in fig 3.1, can be used as building blocks to form more complex hierarchical pipeline model for distributed systems like what is presented in shared input port and share output port model of collaborative framework (ref. section 3.9.4). Note that although we use the term "object", it does not emphasize particularly on the object-oriented model. Rather, it can be replaced by "resource" and each stage is a service operating on a resource.

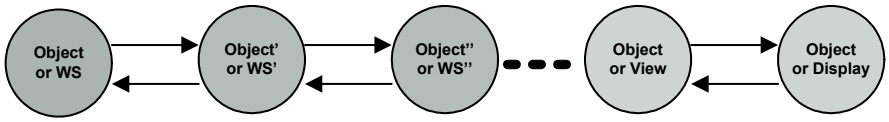


Figure 3.1 Double-linked multiple-stage pipeline model of Web applications

The double-linked multiple-stage pipeline model suggests a number of properties that promise advantages:

1. *Fine grained Modular structure*

   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. The advantage of using this modular multiple-stage approach is that system process can be controlled in a fine grained fashion for dynamic distribution, which is impossible in a canonical two-tier client-server model for Web applications and MVC model for desktop applications. Further more, each service constitutes the "off-

the-shelf" building block that can be quickly integrated into a complex and large scale systems.

2. *Bidirectional double linked stages*

A Web application is decomposed into components or stages. Each stage accepts input, either information of component or the change of component, applies some filters, and passes forward the results on to the next stage along the pipeline. Each object forms a *stage* along the path of pipeline whilst the stage comprises a system *state* in the perspective of the overall workflow. A state change is triggered by an input *event* (e.g. a mouse click). A stateless system has design advantage of simplicity since there's no extra effort needed for keeping *state* information. The prominence is that the double-linked structure enables "finite state change architecture". In a finite state architecture, every system change is labeled by an "event" representing the changes of the system. Operationally, it is important to have the invertible changes between adjacent stages. The structure of adjoining stateless stages enables event-invoked traversal along bi-directions, which is crucial for designing of event-based Web applications with collaboration capability. In contrast with single directional forwarding pipeline model (e.g. UNIX pipe and common software filters), the system automatically keeps track of its states through manipulating of ordered events while no extra storage and management are needed for keeping transitory *states* information. This model also allows participatory components to synchronize via a common state at each stage. We will discuss two different approaches of collaboration through sharing of either *states* or *events* in detail in chapter 5.

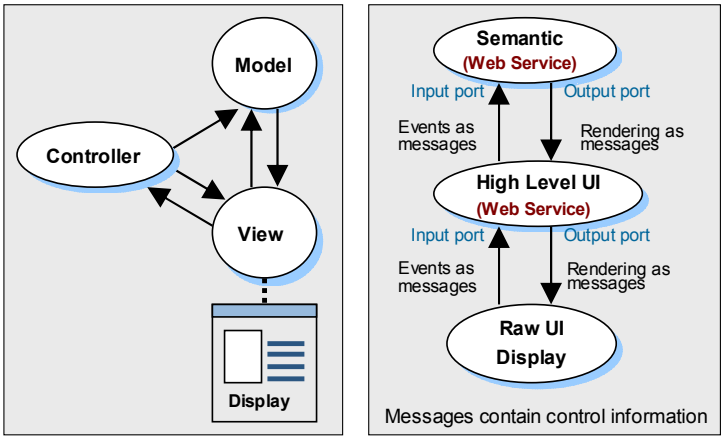
3. *Messaging communication*

The objects communicate with their nearby neighbors via messages. Messaging allows dissemination of the messages in a very flexible manner (e.g. routing through diverse network protocols), which facilitates the formation of dynamic communication patterns (e.g. unicast and multicast) accommodating to different system structure.

3.3 Message-based MVC (M-MVC)

3.3.1 Comparison of MVC model and Web Service pipeline model

Classic MVC model (ref. section 2.2) is originated from Smalltalk (ref. section 2.2.1) and defines a triad that separates the View from the Model with coordination of the Controller component. Distributed applications with service oriented architecture can be deployed using the Web Service pipeline model as described in section 3.2, which constitutes a three-stage structure here, is designed especially to accommodate dynamic, interoperable, and scalable Internet domain. While both application models present modularized architecture as shown in fig. 3.2a and fig. 3.2b, they have clear distinctions.



a. MVC Model b. Three-stage Web Service pipeline

Figure 3.2 Comparison of MVC and Three-stage Web Service pipeline

a) different runtime environment

Classic MVC model is frequently used in legacy desktop or client side applications such as Microsoft Window and Office suite. These systems assume that a software application executes on local computer environment (e.g. multiple processes scheduled to run on single CPU and memory). The Web Service pipeline model is designed for distributed applications that may run over heterogeneous platforms and accessible for multiple users. Various resources, which include CPU, memory, software program, and data, are scattered over the edge of networks and interoperate through Web Service interface by messages.

b) tightness of system coupling

In classic MVC model, *Model*, *View*, and *Controller* components are comprised of classes with runtime instances sharing the same memory space. The interactions are tightly coupled through global variables and method calls, which can be achieved at microsecond level for optimal system performance. However, specific implementations often use global data structure and tend to produce awkward inheritance structures (e.g.

interface class). High coupling of MVC greatly reduces interoperability and reusability of software components [Y. Shan]. The Web Service pipeline model conducts communications via loosely coupled messages over diverse platform via network protocols. Table 3.3 shows the performance of event-based interaction and network latency tolerance for typical message-based distributed applications, where millisecond level fine-grained interaction is common for real time collaboration systems.

c) decomposition strategy

The canonical MVC model has a crude split of the triad with Controller being a separate module in MVC model. Web Service pipeline model allows system division in a fine grained manner at every stage. It delineates an interoperable relationship of geographically distributed components as services. Other than the obvious difference in system structure, two implications play important role in distinguishing their architectural and implementation features.

Firstly, the MVC model provides the separation of an application's presentation from its data structure and behavior; the pipeline model allows further dividing of the data structure from the operations conducted on them (e.g. separating business logic from structure). Here, the data structure refers to a set of resources, the state, and operation is comprised of stateless services.

Secondly, original Smalltalk MVC has a single Controller that takes responsibility for accepting device input (e.g. mouse event) like a "sensor" whilst visual rendering is performed in a separate View component. In distributed systems, the UI component combines these two functions in a single user interface since it is not no sensible to arrange a user input and its feedback (display) in geographically disparate location.

Further more, the processing of incoming request can be conducted at any possible pipeline stages and control information is contained in messages. Namely, Controller is leveraged for distribution.

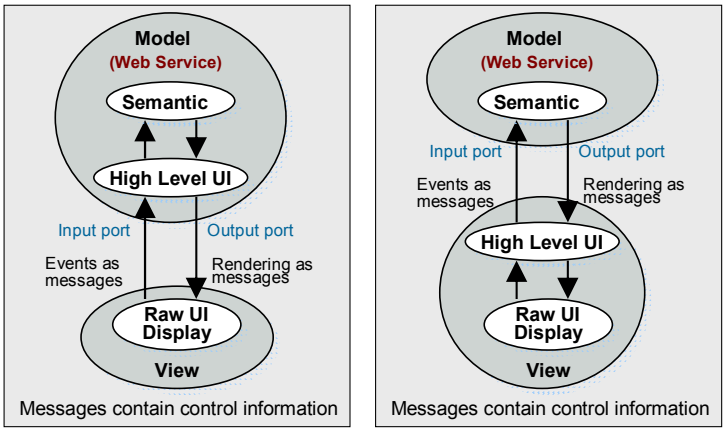
d) application context

Most desktop applications are designed for single user usage. Namely, there's only one participant interacts with UI for the application service. Interoperability and group communication are essential capabilities of distributed applications. These systems commonly involve multiple participants communicating in an asynchronous (e.g. email) or synchronous (e.g. online game) fashion.

3.3.2 Generalization of MVC and Web Service pipeline model

Major differences between the design characteristics of classic MVC model and Web Service pipeline model, as described in previous section ─ tight coupling versus loose coupling, are consequence of disparate system runtime environment and context. The difficulties associated with bridging the gap between desktop and distributed applications have led the research to achieve a unified architecture that is adequate for both application domains.

In order to provide some insights of the M-MVC concept, we provide two among many M-MVC decomposition strategies side by side in fig. 3.3. Both graphs show clean separation of presentation (View) from content (Model) with a delineation of refined three-stage pipelines. Since each pipeline stage constitutes a modularized component that linked by messages, the same structure can be conveniently rearranged with service interface adapting to various distribution profiles.



a. M-MVC model for thin client b. M-MVC model for Web browser front tier

Figure 3.3 Variations of M-MVC decomposition

It has been known that small device such as mobile phone has low memory and CPU; limited display and bandwidth. To facilitate thin client profile sharing comparable collaborative experience as regular profile, one solution is shared display (bitmap rendering) to keep minimum UI functions. The graph on the left (fig. 3.3a) shows a scenario for such thin client interface.

In contrast, graph 3.3b represents a model that suitable for applications with a thicker front tier optimized for performance and rich profile. Common web browsers are typical examples. Notably, runtime environment like J2ME is introduced in particular to support the light weight demand of small devices. With the micro edition, client side can take more responsibility such as high level UI event processing and sophisticated graphics rendering or layout styling. For instance, mobile applications allow cartoon animation [Girow+Mitgartz], map [Zaslavsky+Memon], and other graphics enriched content being rendered with high quality using SVG Tiny technology.

Fig. 3.3 suggests that there're many ways to decompose MVC. It also implies that M-MVC model requires strict and delicately controlled modularity to promise desirable

design features. Several research issues are identified to achieve the design goal of M-MVC.

a) Can tightly coupled MVC model be converted to loosely coupled service oriented architecture?
b) How to support general collaborative paradigms?
c) How to evaluate the performance of system internal processes? Does the message-based approach provide acceptable performance?

Briefly, the following are conceived to be important explorations in addressing the issues described as above.

a) Modification of system architecture from method-based MVC to message-based MVC.
b) Communication with Publish/Subscribe interface provided by underlying messaging infrastructure for real-time collaboration.
c) Performance testing model that tracks down to smallest grained interaction (e.g. mouse event for per pixel change).

Batik [BATIK] is a standalone client application for scalable vector graphics. To test different aspects of M-MVC design, a set of extensive experiments with Batik are conducted including collaborative applications (e.g. shared browser and interactive game) and converting tightly coupled structured to distributed model. The distribution of Batik poses a number of design or implementation issues that challenge the process. As consequences, some interesting observations are obtained such as shared hidden state in MVC; corresponding shared context between distributed components; role of object serialization for messaging and synchronization; problems of excessive use of interface

class; influence of user interaction style, runtime and network environment to system performance. Detailed discussions of the primary indications of design tradeoffs are elaborated under related topics from Chapter 4 through 7.

3.3.3 Summary

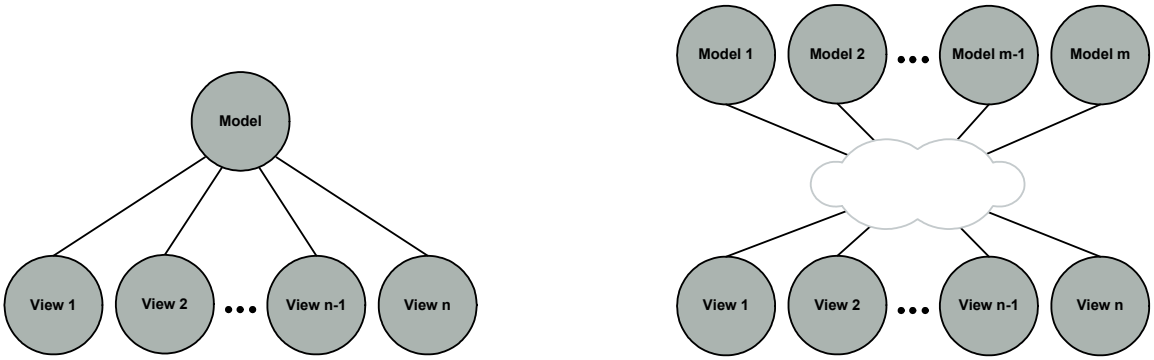
M-MVC is a service-oriented architecture with messaging linkage that unifies distributed and desktop applications. In synopsis of section 1.4, we already entailed its design novelty of bridging several application domains with Publish/Subscribe communication and support of generic collaboration paradigm (SMMV and MMVC). In this section, we elaborated the essential building blocks of M-MVC.

3.4 SMMV and MMMV Interactive patterns

The MVC decomposition for modularity empowers component reusability. Based on the interoperating relationship between the constituent Model and View components, there are three models: one-to-one, one-to-many, and many-to-many. Correspondingly, we propose three MVC interactive patterns: Single Model Single View (SMSV), Single Model Multiple Views (SMMV), and Multiple Models Multiple Views (MMMV).

SMSV is the simple use of MVC in applications and is not our focus of this thesis. However, following the terminology from parallel computing, we emphasize two interactive patterns: MMMV generalizes the concept of MIMD [MIMD]; SMMV generalizes the concept of SIMD [SIMD]. In practice, SMMV and MMMV patterns (fig. 3.4) can be applied in both asynchronous and synchronous applications, thus form general collaboration paradigms.

A commonly used interactive pattern is SMMV, which promotes different presentation layouts sharing of a single content model. This pattern represents the structure of a class of desktop and Web applications: original SmallTalk-80 supports multiple browser panes sharing the same content (e.g. text and/or graphics) through multiple View-Controller pairs versus single Model structure; legacy interactive applications (e.g. Microsoft Windows and Office) allow multiples window layouts sharing the same data structure; typical client/server applications have variety of clients (e.g. Windows, Unix, and Linux) using Web browsers to access a Web server for documents through HTTP protocol.



a) Single Model Multiple View

b) Multiple Model Multiple View

Figure 3.4 SMMV vs. MMMV as MVC interactive patterns

SMMV has limitations as it assumes a single model structure. The problem arises for a complex system, which tends to involve multiple model profile to support corresponding customized view. New generation of Web application architecture emphasizes ubiquity and interoperability, so as to accommodate dynamic and expandable nature of the Internet. In another words, it demands for a framework that supports diverse clients from heterogeneous platform accessing a variety of services in either asynchronous or synchronous fashion. MMMV is a generalization of SMMV that

exploits a many-to-many mapping of MVC decomposition. Thereby, it represents a virtually central feature of new generation of interactive applications.

SMMV and MMMV are two interactive patterns that are well suited pattern for building general collaboration framework. Chapter 5 elaborates on how to use them in building collaborative Web Services through exploiting message-based MVC and Web Service in a unified approach. We note that both SMMV and MMMV can be deployed in an asynchronous and synchronous fashion. Earlier client/ server example shows a typical Single Model Asynchronous Multiple View (SMAMV) pattern. On the other hand, computer users watching CNN live news through media player is viewed a Single Model Synchronous Multiple View (SMSMV) application since network buffered multimedia streams are multicasted to subscribers over a shared period of time. Conceivably, the harder problem is development of synchronous architectures with compelling time constraints.

3.5 Related Work on MVC

As an imperative technology, MVC has been predominantly used in the design of interactive style applications to solve the real-world problems. Nevertheless, the use of MVC concept alone is not sufficient if factors of modern technologies are not taken into consideration. Notably, programming environment has fundamental changes compared with slow CPU and limited memory space in early 80's when original Smalltalk was implemented. It seems that across the publications in both industrial and academia, there's much emphasis on individual development. However, documentation examining MVC paradigm under different stages of computer technology evolution is limited.

In this section, we will provide a detailed classification based on the general perspectives that comprise of a system design: decomposition strategy, interactive pattern, and communication mechanism. In order to highlight communication and interactive features, we classify MVC evolution into three stages. Fig. 3.5 lists three sample models: a) classic method-based model; b) request/response model in method-based or message-based style; c) message-based publish/subscribe model. A discussion of corresponding work delineates variations of MVC approach from legacy desktop application, Web application, to distributed application. A more complete summarization with each category and corresponding applications is shown in table 3.3.

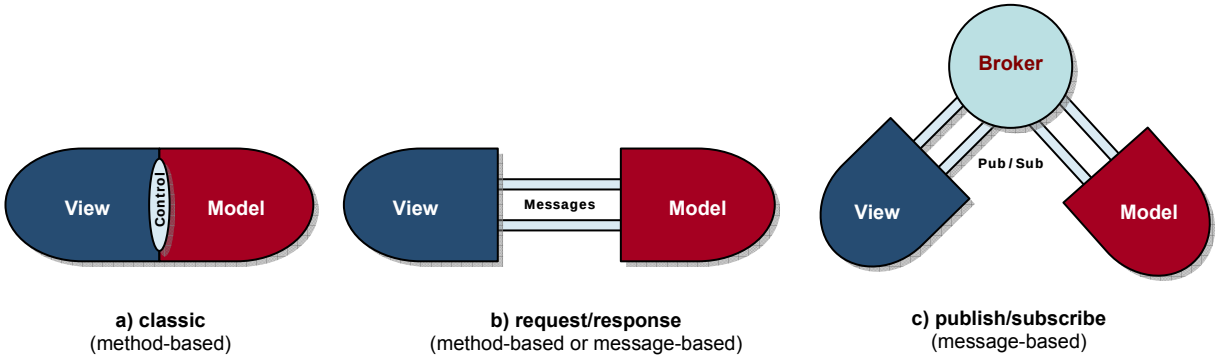


Figure 3.5 Three MVC approaches based on different communication mechanism and interactive pattern between *model* and *view*

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 contracted 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; "The rule of milliseconds" suggests that millisecond and 100 milliseconds are typically found for the communication in intranet and internet scope [Fox04]. These different timescales imply different fundamental building ground for application architecture and viability.

The communication patterns refer to the three models depicted in fig. 3.5. Interactive pattern describes the interoperating relationship between the model and the view: one-to-one, one-to-many, and many-to-many. Details of SMMV and MMMV concepts are covered in section 3.4.

Classic MVC model in fig. 3.5a represents a tight coupling structure for desktop or client applications. The interaction process is illustrated in fig. 2.5 of earlier section. Apparent features include Model, View, and Controller instances run with multiple process using shared memory and single processor; inter-component interactions via method calls with messages hidden at system level; and single-user system. Legacy desktop applications such as Microsoft Office suite employ asynchronous method calls. However, original Smalltalk-80 defines messaging passing with similar semantics to procedure call and sequential computation.

Real-world systems commonly contain entities that exist and do things concurrently. A number of research efforts exploit parallel and distribution features for object-oriented applications [Gao+Yuen] to facilitates a natural extension of Smalltalk. ConcurrentSmalltalk [Yokote+Tokoro] introduces asynchronous method call to achieve parallelism and atomic objects to maintain process request sequentially. Smalltalk (DS) [J. Bennett] provides a Smalltalk implementation with modest supports for multiple users sharing distributed objects. Object interaction in DS is conducted over a high bandwidth network of Sun workstation. As depicted in fig. 3.6, it introduces proxy objects to refer remote objects and employ message forwarding (blocking send at sender) and reply service (invoke receiver's action with returning result at receiver) for communication.

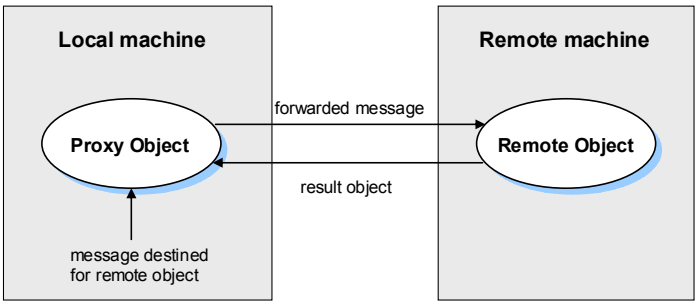


Figure 3.6 Message forwarding and reply between sender and receiver

[P. McCullough] defines "migration" as movement of an object from the remote machine to the local machine and poses questions about whether arguments should be passed by copying, by proxy, or by actual movement. Four possibilities are introduced to address difficulties associated with moving object structure. They are: pass-by-value, pass-by-reference, pass-by proxy, and pass-by-migration.

The Webjinn/DDD framework attempted to address presentation mixing with content structure ("intra-crosscutting") and code tangle within MVC using XP structure that was introduced by [Kojarski+Lorenz].

J2EE and its counterpart .NET are two major platforms that host Web applications. They are based on Java technologies (Servlet/JSP, EJB, and JDBC) and Microsoft technologies (ASP, VBScript, MTS, ADO, COM, and COM+) respectively [MML]. Both platforms incorporated MVC pattern to supply interactive UIs for emerging Web Services framework. The expansion produced corresponding new Web platforms ─ Struts/J2EE and ASP.NET.

Apache Struts [STRUTS] is a server side technology based on JSP model 2 architecture [G. Seshadri] using front controller pattern [SSJ]. By integrating with Java Servlets [SERVLET], JavaServer Pages (JSP) [JSP], JavaServer Faces (JSF) [JSF], EJB,

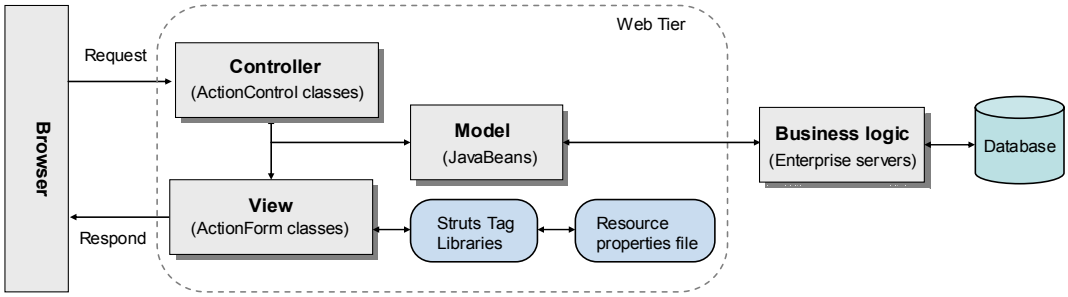


Figure 3.7 Struts/J2EE architecture

and JDBC technologies, it provides a MVC framework. As shown in fig. 3.7, the *Controller* portion of the MVC architecture is focused on receiving requests from the client and executes appropriate action for each request. In Struts, the primary component of the *Controller* is a servlet of class ActionServlet. ActionMapping maps a URI request path to an action class. The action class of the *Controller* servlet invokes business logic beans and passes appropriate ActionMapping instance. An ActionForm of the *View* represents HTML-like tag library to collect user inputs. When an action completes, ActionForwards is used to facilitate *Controller* in selecting output pages for next display to the user interface.

Distributed enterprise architecture tends to involve several components for fulfillment of complicated business transactions. 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. In addition, the connection between Web clients and the single Web-Tier *Controller* is via HTTP transportation while the interaction between Web-Tier MVC components is handled by method calls. We still classify Struts/J2EE as message-based communication here. When ignoring the subtleties of Web GUI

implementation, server side custom tags can be viewed as a transitional form of the *View* in opposed to bitmaps in image buffer at client.

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.

In the Web portal example, multiples clients access backend computing and database management services asynchronously through Web browsers from different platforms (e.g. Windows, UNIX, and Linux). Web portal [PORTAL] provides a middleware technology that provides a gateway interface for clients to communicate with backend services through portlets. Currently, Web client interfaces are typically Web browsers and application specific interface. JetSpeed ─ a relatively unsophisticated Web browser based interface (HTML table with possibly embedded Java Applets) is commonly used as the portlets container.

Representational State Transfer (REST) [R. Fielding] proposed a simplified version of message-based approach that extended from client/server Web application architecture. M-MVC (ref. section 1.4) 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 buffering and pre-fetching [AHMET] 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 3.4 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	√		+ +	√		√			√	
DS		√	+	√		√			√	
Struts/ J2EE		√	+		√		√		√	√
JSR-168 & WSRP		√	n/a	√	√		√		n/a	n/a
REST		√	−		√		√		√	
M-MVC	√	√	−		√			√	√	√

In summary, table 3.4 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.

3.6 M-MVC and messaging infrastructure with publish/subscribe scheme

In section 2.1, we've examined different versions of publish/subscribe as an important asynchronous communication paradigm for distributed systems. Here, we elaborate how M-MVC interfaces with event brokers of NaradaBrokering messaging middleware. Fig. 3.8 shows two decoupled components A and B interact with each other via a topic-based notification service. Initially, component A sets up a topic session using a broker's client interface. The name of the topic is a reference path that typically represents an event class. In order to register for the topic, component B subscribes to the broker using a matching topic name established by A in step 2. Brokers maintain all the active subscriptions and publishers under this topic. Whenever component A publishes an event (step 3), the Broker routes it to current subscribers of the topic including B in step 4. Component B can unsubscribe to disconnect from the session.

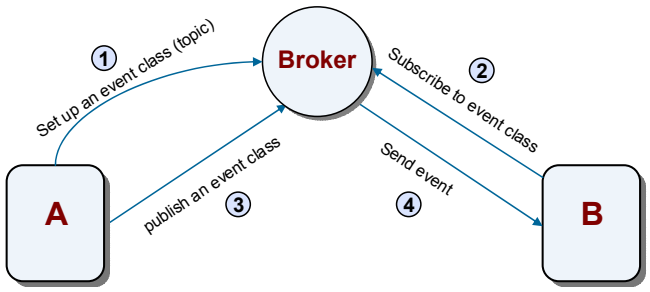


Figure 3.8 Message-based Publish/Subscribe with broker intermediary

The interaction between View and Model components are bi-directional as depicted in fig. 3.9. The View subscribes for notification of the Model changes and execute corresponding update rendering; the Model register to get informed of UI events which eventually invoke method calls that modify Model structure.

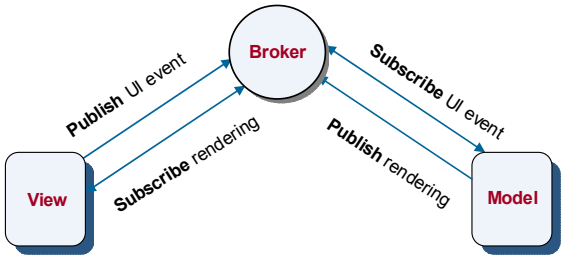


Figure 3.9 Bi-directional interaction in M-MVC with Publish/Subscribe scheme

M-MVC emphasizes a message-based architecture of Web applications enabling Model and View distribution. The approach requires us to support the model-view linkage with a high performance messaging middleware infrastructure. NaradaBrokering has been separately developed and provides a variety of publish/subscribe models including peer-to-peer and Java Message Service (JMS) emulation. M-MVC is not sensitive to the details of NaradaBrokering and do not currently exploit its ability to

traverse firewalls and support multiple protocols. The use for collaborative SVG would exploit these latter Grid messaging capabilities of NaradaBrokering.

3.7 M-MVC and Web Services

Web Services (ref. Appendix D) provide an implementation version of 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 composition issue and application developers have to determine which component should reside in the service versus client interface.

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 represent client interface. M-MVC employ 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 properties:

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).

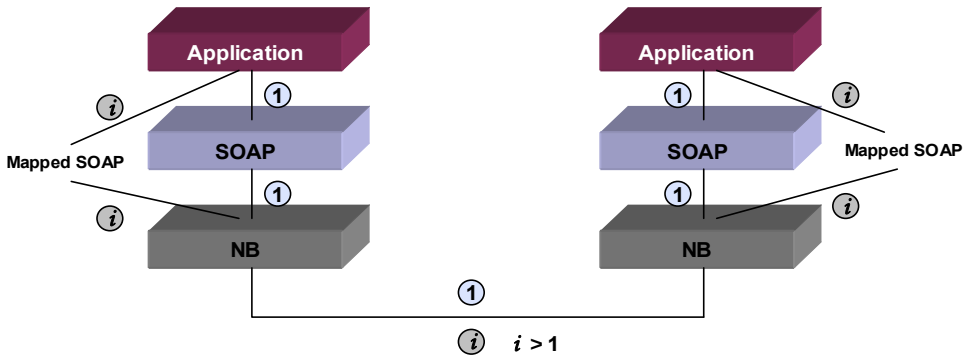


Figure 3.10 Web Services composition of M–MVC application, SOAP, and NB

The Web Services composition of M-MVC is further depicted in fig. 3.10, 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]. Future releases of NaradaBrokering 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 compliance. Our performance testing with SVG experiments generate consistent results under both scenarios.

Chapter 4

Monolithic SVG experiment

4.1 Summary of SVG

Scalable Vector Graphics (SVG) (ref. Appendix F) is a programmable vector graphics technology that embraces following features: mix of vector and raster graphics, open standard, XML format, DOM structure. Particularly, with its support of the W3C DOM event model, SVG satisfies the need for building complex interactive and scriptable Web applications. As a specific DOM application with SVG semantics and XML syntax, it provides a rich and scalable context for prototyping and evaluation of M-MVC design.

The underpinning event model is a key capability that distinguishes SVG from a language that is simply for describing rich graphical content with XML and facilitating static graphics rendering. Interactivity enables SVG to response to user interaction such as zoom in/out and hyper linking with dynamic rendering results. With Java and JavaScript binding, sophisticated applications including scripting and animation can be accomplished by manipulating SVG DOM elements.

We show in fig. 4.1 that a SVG application can be divided into visual component (View), SVG DOM (Model), and application modules (third party software components with language binding such as JavaScript). The decomposition indicates a clear separation of data structure from its representation and system behavior. The logic of

interactive SVG applications can be illustrated by event-based interactions supported by DOM event model.

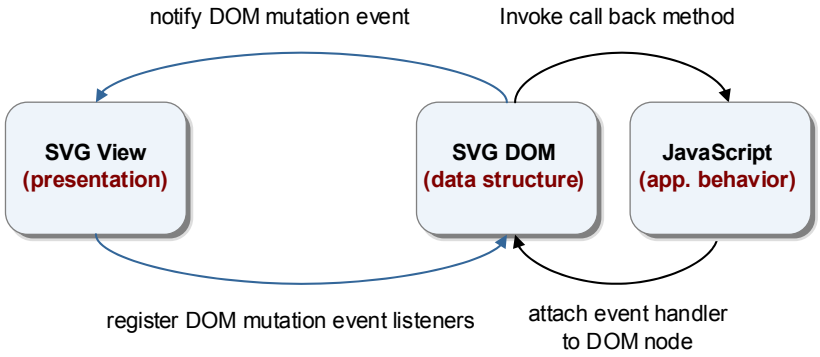


Figure 4.1 Architecture of interactive SVG application

As discussed in section 2.5, there exist two types of event handling in a DOM structure. One is the interaction between DOM and other external components; the other is event propagation within DOM tree. The external event process is derived from basic event/listener model (see fig. 2.1) and applies between decoupled model-view and model-JavaScript components. The former assures that any SVG DOM structure change would get notified in visual component and trigger corresponding update rendering. The latter allows user input invoking access and modification of DOM from call back methods. DOM Node interfaces (see fig. E.1) facilitates adding event listeners on element nodes to receive notification of various types of events. We provide a simple example using a toggling rectangle (ref. Appendix F.4) to illustrate how SVG animation is supported based on user interactions.

4.2 Summary of Batik SVG Browser

4.2.1 User interface of SVG browser

Batik [BATIK] is “a Java-based toolkit for applications that want to use images in the Scalable Vector Graphics (SVG) format for various purposes, such as viewing, generation, or manipulation”. It has been built to conform to SVG 1.0 specification [SVG]. As shown in fig. G.1, it provides developers with a set of core modules (e.g. SVG parser and generator, DOM implementation, GVT and image renderer) that can be used together or individually as building blocks for host Web applications.

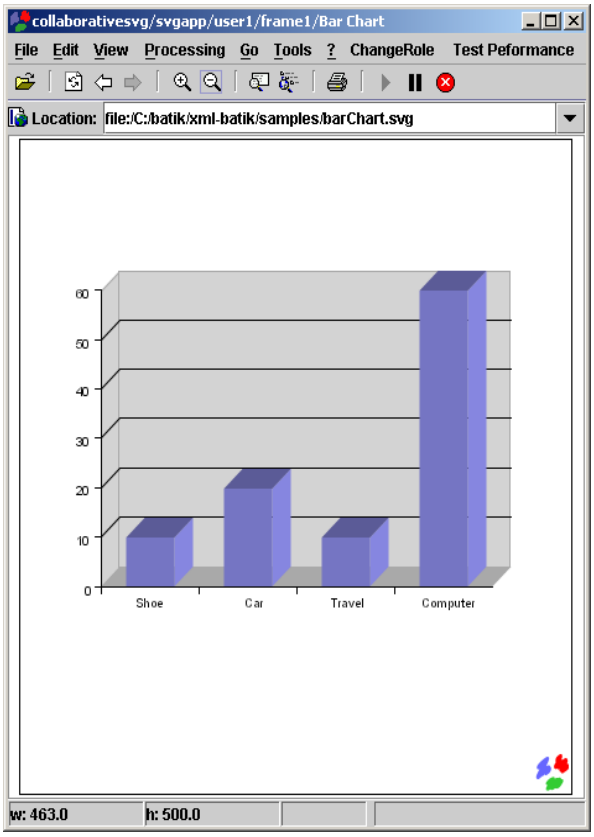


Figure 4.2 Screen shot of Batik SVG browser

Common Web browsers are used for browsing HTML document. Likewise, Batik SVG browser is a client application for navigating SVG content. As vector graphics allow zoom in/out, rotation, translation, and visual effects (e.g. clipping, masking, and alpha channel), SVG browser is very suitable for presentation style client interface and

various display sizes. A screen snapshot of a modified SVG browser with a displayed document from Batik “barChart.svg” [BATIK] is shown in fig. 4.2. The window is comprised of two main parts: menu bar at the top and canvas area in the center. In addition to file operations such as “open” and “reload” a local or remote SVG document, the menu bar mainly provides a set of graphics functions including zoom in/out, rotation, and translation. As discussed in section F.1, vector graphics are converted into bitmap-based images (so called “rasterizing”) at the last moment before display. This allows the flexibility of applying various imaging processing methods (e.g. affine transforms) while keeping rendering with high resolution. Fig. F.2 compares the original and 2 times scaled varChart.svg document.

Batik browser has three distinctive ways in presenting a SVG document, which are described in the following:

One operating mode supported by Batik is loading in a SVG document with basic image rendering options (affine transforms) provided by menu items. Since it doesn’t change original SVG content and there is no real user interaction at all, this approach is called “static SVG”. Batik defines it as “static mode” with parameter ALWAYS_STATIC. There’s no event association between DOM and GVT image renderer. In another word, operations are conducted only at the root node of SVG DOM tree. That is, taking SVG document as a whole. In fact, there’s no DOM structure needed theoretically as all rendering state is held at GVT tree.

Another mode is so called “interactive SVG”, which supports hyperlink functions. Since the semantics of mouse event (mouse click over a URI reference element), cursor behavior (changing appearance), and action (invoking a SVG file loading) are straight

forward, limited DOM and GVT links is required. Batik uses ALWAYS_INTERACTIVE to refer to the mode. Note we use URI here to include references to an internal document fragment contained within a document file while URL generally only indicates a path to a file location.

The third and most complete interaction mode is provided by "dynamic SVG" in ALWAYS_DYNAMIC model. It allows a user to manipulate SVG content at run time through adding, modifying, and deleting elements over SVG DOM structure. To enable interactive and animation experience, substantial event-based interactions are involved between SVG DOM, backend JavaScript, and client user interface components as illustrated in fig. 3.9. Because mouse events are detected and captured at per pixel change based, the process usually demands for frequent accessing of GVT and internal SVG DOM structure. DOM event model is used to handle event invocations and propagations.

In summary, static, interactive, and dynamic are different mechanisms that Batik constructs and renders SVG in the browser. Interactive and dynamic SVG are performed in canvas area by invoking mouse events that can be controlled in a fine grained per pixel change manner. Corresponding Batik implementations provide reusable components for building highly interactive client interface with SVG.

4.2.2 Architecture and implementation of SVG browser

Batik SVG browser is an application built with core modules provided by Batik toolkit. The architecture includes a set of XML handling, bridge/transcoding, rendering and display classes, as shown in the patches from left to right of fig. 4.3 [T. DeWeese].

For convenience, we introduce each major package based on system workflow and refer to a more detailed illustration in appendix if necessary.

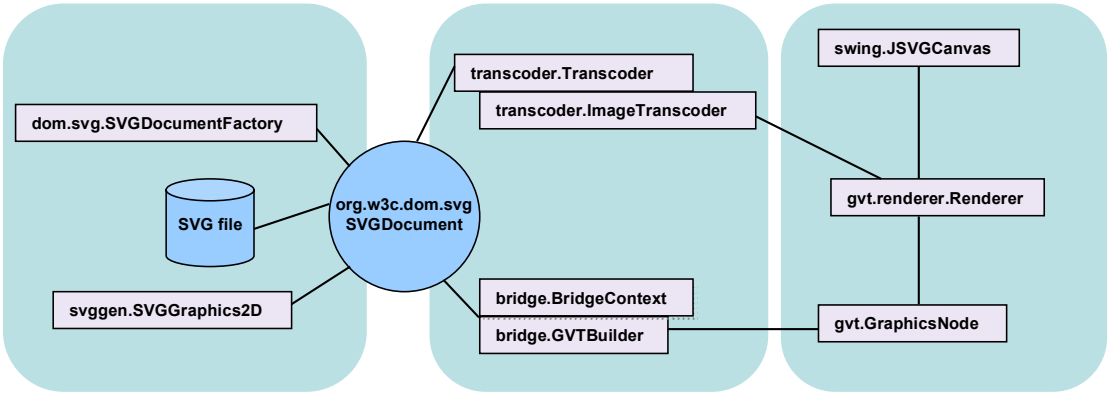


Figure 4.3 Architecture of Batik SVG browser

A key piece of work for a SVG browser is to load in a SVG document from a URL and prepare it for display. This process includes following steps: parsing URL, loading SVG document, creating SVG DOM tree, building GVT tree, and rendering GVT. A complete workflow from opening an URL link to rendering to display devices is provided in fig. H.3 and H.4 in the appendix. To enable interactive and dynamic SVG features, Batik deploys JavaScript coding, DOM event handling and JavaSwing update rendering to repond to real-time user interactions.

The relationship between `SVGDocument`, `bridge.GVTBuilder`, and `bridge.BridgeContext` are shown in fig. 4.3. The class of `dom.svg.DocumentFactory` defines an interface to create SVG Document. A critical pre-processing step is loading SVG into memory. Based on parsed document path, the `org.apache.batik.bridge.DocumentLoader` class decides whether to retrieve SVG from a local file system or open a network connection for a remote server. Since SVG is composed of XML stream, XML parser plays an important role in validating and/or

parsing each input tag. The process is depicted in fig. H.2, where `SAXDocumentFactory` provides utility classes to build SVG DOM. Note that when a parsed SVG element is inserted into the DOM structure, it also fires an event to invoke methods in `org.apache.batik.bridge.SVGGElementBridge` class, which results in a correponding graphics node being added to the GVT tree as well (ref. fig. H.5). As a distinctive design feature, Batik introduces a GVT tree to seperately handle rendering and display classes. DOM provides interfaces for accessing geometric information. Bridge classes implement the interfaces by using GVT classes so as to accomplish the mapping between DOM nodes and GVT nodes. It worth mensioning that `bridge.BridgeContext` holds useful context information to discribe the mapping. The following example shows key steps to load in a SVG document and build a GVT tree.

```
ua       =  new UserAgentAdapter();
loader   =  new DocumentLoader(ua);
ctx      =  new BridgeContext(ua, loader);
svgDoc   =  loader.loadDocument(url);
svgRoot  =  svgDoc.getRootElement();
builder  =  new GVTBuilder();
gvtRoot  =  builder.build(ctx, svgDoc);
```

Although most sophisticated graphics systems use vector graphics, mordern display devices are based on rasterized images (ref. Appendix F.1). SVG enables rasterizing to occur before writing to an offscreen image buffer, which allows much more flexibility for transformation (e.g. scaling) and rendering optimization. Batik SVG browser uses the canvas area for SVG display. The rectangular area maps to a corresponding image buffer in memory. SVG rendering is handled by `swing.JSVGCanvas`, a Java Swing UI component as depicted in fig. H.1.

Interface gvt.renderer.Renderer can be implemented by two ImageRenderers: `StaticRenderer` and `DynamicRenderer`, which correspond to the rendering of static SVG and dynamic SVG document. The "`renderer.repaint(Shape area)`" method does the real work of rendering GVT tree into raster images (ref. fig. H.13). Returned offscreen image is used by the renderer for rendering. Critical steps of the SVG rendering process are included as following.

```
DynamicRenderer renderer = new DynamicRenderer();
renderer.setTree(gvtRoot);
renderer.setTransform(ViewBox.getViewTransform(null, svgRoot, Width, Height));
renderer.updateOffScreen(Width, Height);
area = new Rectangle(0, 0, Width, Height);
renderer.repaint(area);
BufferedImage image = renderer.getOffScreen();
```

At lower level implementations, `renderer.repaint(area)` invokes rendering of GraphicsNode for tiles that cache image data for future usage.

```
BufferedImage offScreen = new BufferedImage(cm,
                              wr.createWritableTranslatedChild(0,0),
                              cm.isAlphaPremultiplied(),
                              null);
Graphics2D g2d = GraphicsUtil.createGraphics(offScreen, hints);
CompositeGraphicsNode  node.primitivePaint(g2d);
AbstractGraphicsNode   node.paint(g2d);
```

The image buffer is created based on properties of ColorModel and Raster, where `cm` is `ColorModel` assigned to `GraphicsUtil.sRGB_Unpre`; `wr` is a `WritableRaster` provides pixel writing. The rendering starts from GVT root node. On occurrence of composite node, it needs to loop through the child nodes (ref. fig. H.11). The renderer paints different types of node separately (see fig. H.12). GVT content in bitmap image is represented by the offscreen image of JGVTComponent, a Swing UI component in

canvas area (ref. fig. H.8). For dynamic SVG document, the update rendering of JGVTComponent can invoke other UI components such as border and menu container to re-display as well. Note that SVG rastering can use either single or double buffer as shown in fig. H.9.

Two additional packages are transcoder.Transcoder and svggen.SVGGraphics2D. The former converts SVG to traditional raster image format like PNG, JPEG, PDF, and TIFF; the latter generates SVG content from Java Graphics2D applications. Since these classes are not associated with the center work of the thesis project, we will not go into further detail here.

4.3 Intercepting Events in Batik SVG Browser

We've designed a set of monolithic SVG applications based on Batik SVG browser [QCF-07-03]. The experiments use SVG as a framework to build interactive collaboration applications with publish/subscribe scheme. One type of applications is mainly sharing of static SVG documents. Specifically, it allows sharing of URL (including URI that pointing to document fragments) and affine transforms such as zoom in/out and rotation. Because interactions on static SVG document are conducted only at document root node, there's no internal DOM event involved. The other type of applications enables sharing of dynamic SVG features, which include sharing of advanced capabilities such as hyperlink and animation. As JavaScript add event listeners on SVG DOM elements to invoke call back methods in the process (ref. fig. 4.1), it enables development of separate application logic (e.g. intelligence for online game using JavaScript) from management of the DOM structure. DOM event model is extensively used in this mode for user input invoked interactions.

Identical SVG applications can be made collaborative by sharing of events via messaging broker, as illustrated in fig 4.4. As we have logical decomposition of an SVG application into fine-grained pipelines (ref. fig. 3.1), a system interaction can be represented clearly by the event workflow along the pipeline path from following stages: user interface, GVT, SVG DOM and JavaScript. Interception of events can occur at any stage of the pipeline on both legs. The intercepted event can then be published to a common topic managed by the messaging broker, which is forwarded to other participant applications. After receiving the event, target applications insert it into local system pipeline at the same stage that invokes similar functions.

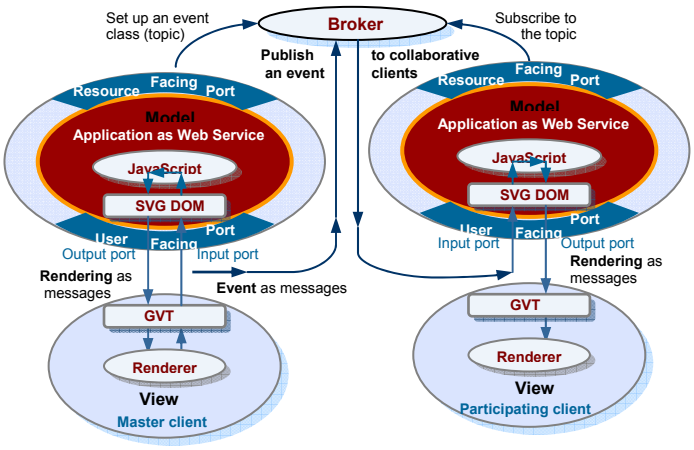


Figure 4.4 Making SVG collaborative by sharing of intercepted events

The concept of "event" can be defined in a broad sense for different applications (ref. section 3.1.3). However, the complexity of a collaboration system largely depends on the granularity level of events for sharing: the smaller, the more time constrained. In an interactive SVG game application, for example, it is common to require event-based interactions being controlled at per pixel change level. In this way, it allows timely reaction to sustained user inputs to achieve real-time experience. On the other hand, sharing of URL event can be considered as large grained interaction, since large grained

events like loading and rendering a new graphics document are expected to be a less frequent operation due to the huge overhead of I/O and computation.

We provide in Table 4.1 a list of events shared in the implementation of monolithic SVG collaboration. These events can be used independently or combined to achieve a variety of collaborative capabilities ─ as simple as a shared zoom in operation on a static SVG document or as complicated as a shared chess online game. We classify the events into Batik GUI Event, UIEvent, DOMEvent, SemanticEvent, and ControlEvent to show that events are handled at different level of the system design. Further more, within an application, the mechanism of processing events belonged to the same type (or feature) is quite similar while events from disparate types are often treated very differently. The complexity is dominated by event granularity and corresponding intensity of interaction and we have summarized this feature in Table 3.3.

Table 4.1 Events for monolithic SVG collaboration applications

Event type	Features	Event name
Batik GUI Event	Window/Viewer action	COLLABORATIVE_NEWWINDOW_ACTION
		COLLABORATIVE_WINDOW_RESIZE_ACTION
		COLLABORATIVE_COMPONENT_RESIZED_ACTION
		COLLABORATIVE_CLOSE_ACTION
		COLLABORATIVE_EXIT_ACTION
	Load a document	COLLABORATIVE_OPEN_ACTION
		COLLABORATIVE_OPEN_LOCATION_ACTION
		COLLABORATIVE_OPEN_HYPER_LINK_ACTION
	Document history	COLLABORATIVE_RELOAD_ACTION
		COLLABORATIVE_BACK_ACTION
		COLLABORATIVE_FORWARD_ACTION
		COLLABORATIVE_LOCAL_HISTORY_BUTTON_ACTION
	Affine transformations of SVG (e.g. scaling, rotation, and translation)	COLLABORATIVE_AFFINE_ACTION
	Affine transform history	COLLABORATIVE_RESET_TRANSFORM_ACTION
		COLLABORATIVE_SET_TRANSFORM_ACTION
		COLLABORATIVE_PREVIOUS_TRANSFORM_ACTION
		COLLABORATIVE_NEXT_TRANSFORM_ACTION
	Animation through thread control	COLLABORATIVE_PLAY_ACTION
		COLLABORATIVE_PAUSE_ACTION
		COLLABORATIVE_STOP_ACTION
	Functions of the panning window	COLLABORATIVE_THUMBNAIL_DIALOG_ACTION
		COLLABORATIVE_THUMBNAIL_DIALOG_CLOSE_ACTION
		COLLABORATIVE_THUMBNAIL_DIALOG_MOUSE_RELEASE_ACTION
		COLLABORATIVE_THUMBNAIL_OVERLAY_MOUSE_RELEASE_ACTION
		COLLABORATIVE_THUMBNAIL_OVERLAY_MOUSE_DRAGGED_ACTION

		COLLABORATIVE_THUMBNAIL_OVERLAY_MOUSE_PRESSED_ACTION
	Other menu functions	COLLABORATIVE_VIEW_SOURCE_ACTION
	Collaboration changing role request. Role change between participant/master clients	COLLABORATIVE_CHANGE_ROLE_ACTION
UIEvent	RAW Event or AWTEvent. Currently only mouse event is processed (no key event)	COLLABORATIVE_UI_MOUSE_CLICK_ACTION
		COLLABORATIVE_UI_MOUSE_DOWN_ACTION
		COLLABORATIVE_UI_MOUSE_UP_ACTION
		COLLABORATIVE_UI_MOUSE_OVER_ACTION
		COLLABORATIVE_UI_MOUSE_OUT_ACTION
		COLLABORATIVE_UI_MOUSE_MOVE_ACTION
DOMEvent	Mutation related DOM tree change	COLLABORATIVE_SET_ATTRIBUTE_ACTION
		COLLABORATIVE_REMOVE_CHILD_ACTION
		COLLABORATIVE_APPEND_CHILD_ACTION
SemanticEvent	JavaScript function related event	COLLABORATIVE_JAVASCRIPT_ALERT_ACTION
		COLLABORATIVE_JAVASCRIPT_PROMPT_ACTION
		COLLABORATIVE_JAVASCRIPT_CONFIRM_ACTION
		COLLABORATIVE_JAVASCRIPT_SET_TIMEOUT_ACTION
		COLLABORATIVE_JAVASCRIPT_SET_INTERVAL_ACTION
	JavaScript function dependent event	COLLABORATIVE_JAVASCRIPT_SET_PROMPT_INPUT_ACTION
ControlEvent	A simplified version of session control events. To be replaced by XGSP events.	DEFAULT
		JOIN_IN_SESSION
		REQUEST_FOR_CHANGING_ROLE
		CHANGE_ROLE
		REQUEST_FOR_A_BARRIER
		SET_BARRIER
		RELEASE_BARRIER
		CONTROLLER
Miscellaneous events	Serialization of DOM action	COLLABORATIVE_SERIALIZED_DOM_ACTION
	Serialization of Graphics2D object	COLLABORATIVE_PAINT_SHAPE_ACTION
	Synchronization for the performance testing model	COLLABORATIVE_BEGIN_SYNC_ACTION
		COLLABORATIVE_END_SYNC_ACTION
		COLLABORATIVE_IMMEDIATE_BEGIN_SYNC_ACTION

Batik GUI Events are mainly comprised of `ActionEvents` that produced by Swing UI components such as menu bar items. Corresponding reactions are predefined in an application as `AbstractAction` classes with call back method `actionPerformed(ActionEvent e)`. On receiving of an ActionEvent, the event listeners added on the Swing UI components invoke the actionPerformed method and execute containing operations. An action map is used in `org.apache.batik.apps.svgbrowser.JSVGViewerFrame.java` to provide mapping between reference names and instances of the AbstractAction classes.

UIEvent, DOMEvent, and SemanticEvent are related to RawUI, High level UI, and Semantic stages of the M-MVC decomposition in fig. 1.2. Unlike Batik GUI Events that deal with static SVG documents and auxiliary user interface functions, the group of events associated with internal DOM structure and DOM event model ─ the core parts of interactive SVG design and implementation. Because UIEvent, DOMEvent, and SemanticEvent are based on per pixel change, per element node mutation, and potential multiple modifications of the DOM structure, the granularity of events become very small. Thereby, event sequence and time constraints are particularly sensitive due to the nature of synchronization. For instance, a missing mouse move event over the network in a sequence of mouse down, mouse move, and move up would generate a totally different semantic meaning to a receiver since original "mouse drag" is replaced by a "mouse click" instead. For another example, a prompt input dialog window of a master client would block other treads' execution until it receives the user input and returns. At participating client side, however, there needs similar thread synchronization implementations to share the prompt window action but using master client's user input value instead. Because open a prompt dialog and set prompt input are two events with dependency, they must be processed in a row. These examples illustrate some subtle event correlated situations that may occur requiring to be taken care of for proper functioning. At meanwhile, it shows why a high performance, QoS, and fault tolerance enabled messaging service that provided by underlying messaging infrastructure is essential for real-time interactive applications.

4.4 Properties and structure of events

We need a sophisticated understanding of events as our research relies extensively on the idea that the state of any entity (object, component of SVG browser, Web Service) can be defined in terms of initial state and a stream of change events. In this chapter sharing the change events gave us the monolithic collaboration of fig. 4.4. In section 4.4.1, we describe some broad classes of events while section 4.4.2 gives some specific details of the SVG browser case. An important feature discovered from the research in this chapter is the hierarchical structure of events for sophisticated interactive SVG applications. This is shown and discussed later in fig. 4.5 where an initial single "root" event such as a user mouse click, generates a multitude of different events as its pipelined processing evolves.

4.4.1 Classes of Events

Here we describe three aspects of events that apply to general collaboration scenarios and have been explored in earlier research [Fox98]. We implemented these ideas in our systems.

1) Master and non-master events

In our collaborative session, all participating clients subscribe to a session (shared application) topic through NaradaBrokering system. Note NaradaBrokering supports traditional (in publish/subscribe systems) hierarchical topic labels and this is used to conveniently label related event streams. Among the clients in a given shared application, only one client holds the "master" token and generates master events that trigger collaborative behavior in the collaboration group. We term events that come from other

participating clients as non-master events. The master token can be changed dynamically. Non-master clients can – as in all such collaborative architectures – choose whether or not to follow precisely the master's state. In the case of the chess game, this general characteristic is refined to "Master whose turn it is to move", "Opponent who will get the next move" and "Observer". This is typical of such games and "Opponent" and "Observer" act as "non-masters" while the "master" token is exchanged between the black and white players. As part of this early research, we built (using the same NaradaBrokering infrastructure) a protocol where observers could bid for the player (black or white) role. Although this was successful we did not pursue it as the intended overall framework XGSP described below was delayed and this type of work was not needed for our core M-MVC research.

2) Major events versus 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 possible synchronization point with major events. Major events are selected semantic events (such as load a SVG file and open a new window), which fully specify the application state. Chess game major events correspond to the completion of each move. 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 replay can always 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. One commits 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". CGL has shown that NaradaBrokering can efficiently support both full state and change events; for example, the Anabas commercial web conferencing system can use NaradaBrokering to handle multi-megabyte shared display events with excellent performance [ANABAS].

This idea is important as it supports the concept of "undo" in an M-MVC application. We suggest that systematic application of the event based application model – namely an application is represented by initial state and a stream of change events – will produce a very interesting computing environment where one can undo in a systematic powerful fashion. However we did not pursue these ideas in our research presented here.

3) 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?) needs to be specified. CGL has developed a general architecture termed XGSP

or XML General Session Protocol for this [WUBF] [WBUF]. XGSP is the protocol that controls a Collaboration Web Service. This service initiates collaborative applications such as SVG discussed here and for example generates the "master token". Thus the SVG MVC Controller event handler must process both events specialized to the application and such overall control events.

We note that XGSP has not been fully developed although the GlobalMMCS project [FWUBP] has produced a powerful service-oriented audio-video conferencing collaborative environment with a session server playing the role of "the Collaboration Web Service". This session server plays the role of a software MCU (Multipoint Control Unit) and supports the parts of the H323 protocol needed for audio/video conferencing. However although XGSP was architected to support general collaborative applications, this capability was not implemented due to lack of resources in CGL. My research was not developing "production collaborative SVG" and so we did not pursue this area in our research. It is another topic for future research.

4.4.2 SVG Browser Events

Now we discuss special features of SVG events that are illustrated in fig. 4.5. As already discussed, we classify SVG events into three categories – Raw UI events, High Level events and Semantic events. Raw events are low level events that are directly generated by user input ─ for example, mouse and keyboard events; High Level events are generated by SVG from Raw events and W3C SVG/DOM events are of this category. Semantic events represent functionality of the SVG application or service. "Zoom" in a SVG browser and "I Resign" in chess are such examples of semantic events.

We introduce a collaborative event as an object that wraps original SVG events with additional context information needed by the collaboration and Service model. The context information helps guide the events through the NaradaBrokering system to reach other clients (subscribers in the same session). The receiving client un-wraps 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).

All events contain the information such as follows:

- An indication as to their category: either original Raw or High Level UI Event or semantic events as generated by JavaScript or directly from the DOM by SVG
- Event characteristics (e.g. master or non-master, major or minor)
- Context information of the collaboration (e.g. client ID, session/topic, black or white for the chess game or more generally application specific meta-data, windows name in a multi-SVG viewer application, event sequence number)
- Context information of the Web services specifying application and collaboration session.

The collaborative SVG event processing chart is show in fig. 4.5. Note that we serialize

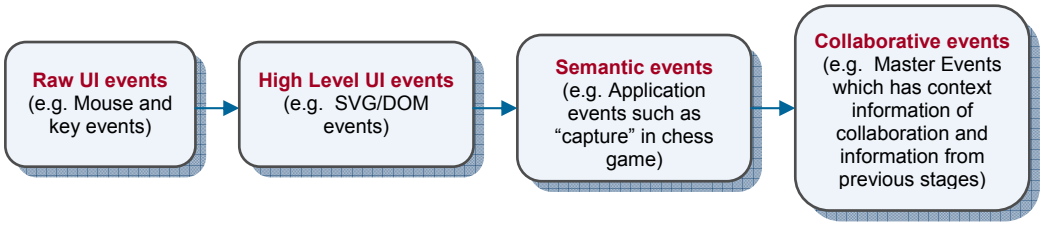


Figure 4.5 Collaborative SVG event processing chart

SVG events using a natural XPATH syntax to specify DOM node position and its properties.

4.5 Conclusions

We use this experiment to develop experience with Batik to design events structure that can be used for later research. We also of course used the understanding of the source code developed in this work to design the more delicate decompositions and event captures needed in the message based MVC described later. Success at this stage was an essential step to our later research.

We found good performance of the system but we do not present this here as it has been demonstrated in other CGL systems such as the collaborative Anabas, PowerPoint [WFP+04], and IDL applications [WFP+05] in CGL, which have also used NaradaBrokering for such "monolithic" collaborations. Performance in collaboration between inevitably distributed machines (Often separated by a 100 milliseconds or more network delay) is much less critical than that for M-MVC where one is dealing with response on a single machine; thus our discussion of chapter 7 of M-MVC performance is very thorough. We also used this monolithic system to develop the new collaborative paradigms SMMV and MMMV described in Chapter 5. We note however, these paradigms are actually better suited for the decomposed SVG browser in Chapter 6 and 7.

Chapter 5

Collaborative SVG

5.1 Collaboration framework

Internet collaboration presents emerging important features with participatory and interactivity in Web applications development. We have already explained briefly how one can make message-based network applications collaborative in two modes – *shared input port* and *shared output port* [Fox03]. We give more details here.

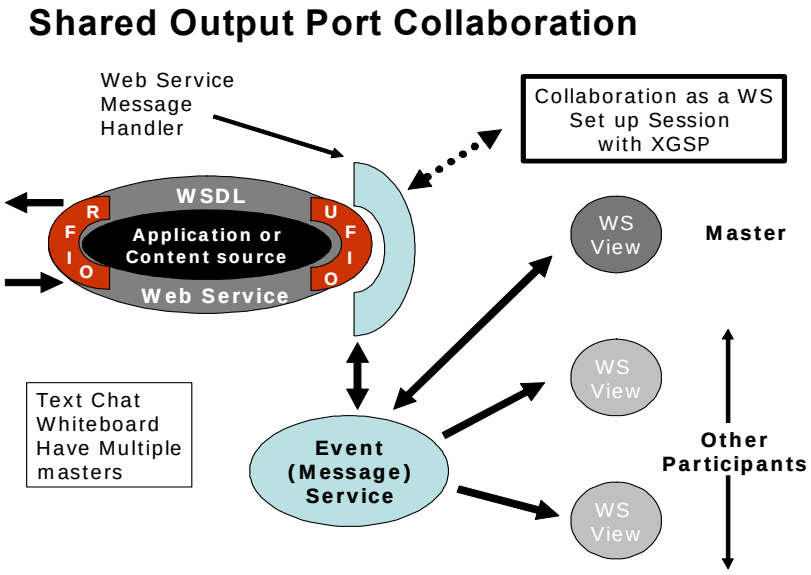


Fig. 5.1 Shared Output Port Collaborative Web Service Paradigm modified from a figure in [Fox03]

Web Services interact via messages input or output through ports. These messages are called "user-facing" (UFIO in figs. 5.1 and 5.2) or resource (service) facing (RFIO in figs. 5.1 and 5.2). The user facing ports handle all the negotiation and data for producing the rendering associated with the Web Service. The negotiation is often associated with

Portlets [JSR168] and the WSRP protocol [WSRP]. The shared output port model shown in fig. 5.1, has a single Web Service with the user facing messages on the output port multicast to all clients. On the other hand the shared input port model replicates Web services and they are synchronized by sharing input messages on resource facing ports. There are many similarities between these two modes of collaboration.

1) The Web Services use all the usual protocols (WSDL, SOAP) on each port; this key characteristic is left unchanged.
2) In each case, one multicasts the messages – either those arriving at a shared input port or those produced by shared output port.
3) Further in each case a client assigned with "master" token has "master role". Requests for switching between different roles (e.g. "master" versus "nonmaster" and player versus observer) can be done dynamically as discussed in section 4.4.1.
4) Each model uses the "Collaboration as a Web Service" control service described in the XGSP discussion in section 4.4.1.

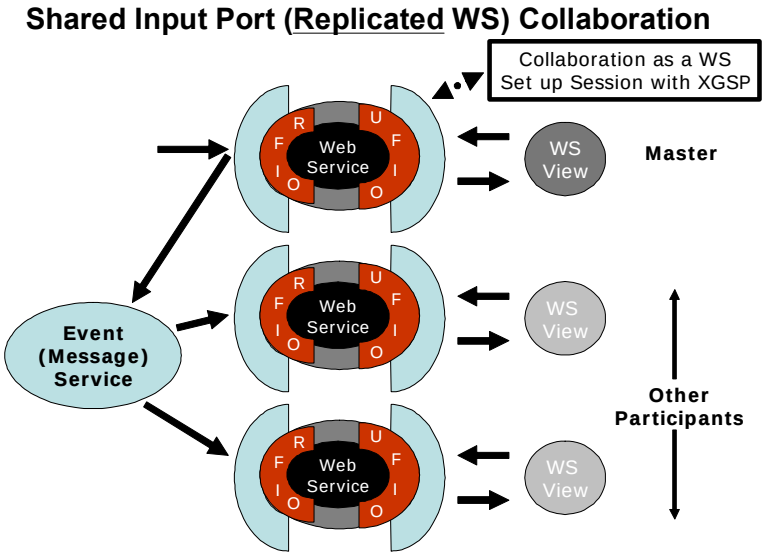


Fig. 5.1 Shared Input Port Collaborative Web Service Paradigm modified from a figure in [Fox03]

The CGL has systematically developed this model with audio-video conferencing [UWBF], text chats, whiteboards, PowerPoint [WFP+04], OpenOffice [Wang+Fox] and shared display [LFKWQ]. Although our research contributed shared SVG [QCF-07-03] to this suite, this is not our major contribution. Rather it is to extend this distributed Web Service model to desktop applications including the key idea of separating *Model* and *View* of the MVC paradigm by explicit messages and by controlling these messages by a publish/subscribe mechanism. Note in the MVC language, the Web Services in figs. 5.1 and 5.2 are the *Model* and the "WS View" the *View*. There are other interesting analogies between the distributed and desktop application case; for example we mentioned above the WSRP protocol which could be usefully adapted as the *Control* protocol in MVC. Portlets have a less fundamental role as enabling control of the layout of multiple service views; however such a layout model could be "borrowed" from the Web Service infrastructure and provide further powerful integration of the desktop and distributed models.

Note that sometimes one views shared display as "different" from the offered "event" based collaboration models. We view all these cases as just differing by where in the pipeline from *Model* to *View* one shares events. Shared display shares at one extreme end and its events correspond to changes in the bitmap of the rendered view. Other modes share at an earlier stage where the events define for example the state of the *Model* or *View* in a less concrete fashion than the bitmap of the shared display.

Collaboration on this particular SVG project served well to teach us about remote collaboration in general, both from a technological and an interpersonal standpoint. As

discussed later we developed two new models for collaboration based on our experience in this regard.

5.2 Event-based collaboration

We discussed the critical concept of events in detail in section 4.4. Here we extend the discussion focusing on issues important for collaboration. As we know, collaboration is corresponding to sharing of either system states or events among participatory components. Moreover, collaboration is accomplished through synchronization among participatory components by sharing of either *state* directly or *event* indirectly. This is described in section 4.4.1 in terms of state change being specified by events. In this thesis, we discuss two ways of building an event-based collaboration system: monolithic and Web service.

Our approach of event-driven message-based collaboration with Publish/Subscribe scheme (see figs 3.6 and 3.7) has following implications:

- An "event" defines the incremental change of system state. We have given in Chapter 4 a complete analysis of events and classify them as UI event, SVG/DOM event, and semantic event categories in our collaboration experiments with SVG. 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 fig. 6.2, 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 (ref. fig. 1.2), "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 2.4. 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.
- 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 as described later in this chapter. 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.

5.3 Monolithic collaboration

Monolithic collaboration (see fig. 5.3), 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

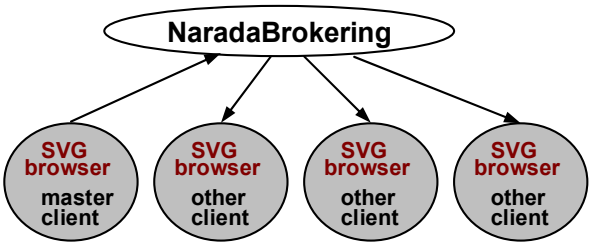


Figure 5.3 Monolithic collaboration

proprietary or open source implementations. We have described in detail in the case of SVG in chapter 4 although this mainly discussed the non collaborative case. Monolithic is contrasted with the explicit separation between model and view advocated in this thesis for MVC style applications. For the separated case which is automatic for distributed Web services we discuss in the following two sections the two modes depicted first in figures 5.1 and 5.2.

5.4 SMMV collaborative Web Service model

In the next two sections, we show that in the 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 [Fox94] which includes the two key architectures SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data). We show that the mode of fig. 5.1 can be thought of like SIMD and that of fig. 5.2 as MIMD.

Single Model Multiple View (SMMV) shown in figs 5.1 and 5.4 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.

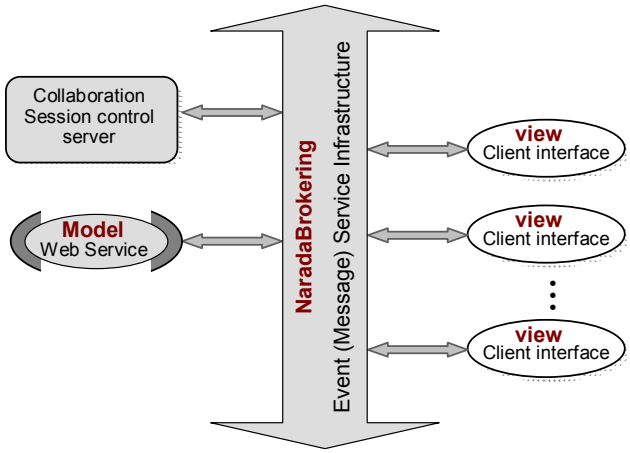


Figure 5.4 Architecture of SMMVC collaborative Web Service model

5.5 MMMV Collaborative Web Service model

MMMV is a generalization of SMMV, which enables ubiquity with the customization done from the *Model* at server side and is shown in fig. 5.2 and 5.5. 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. Now 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 fig. 5.6. As one example consider JavaServer Faces (JSF) [JSF], which extends JavaServer Pages (JSP) [JSP] and Java Servlet [SERVLET] 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.

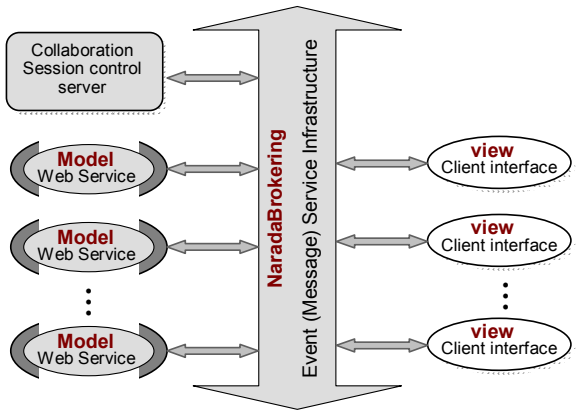


Figure 5.5 Architecture of MMMVC collaborative Web Service model

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.

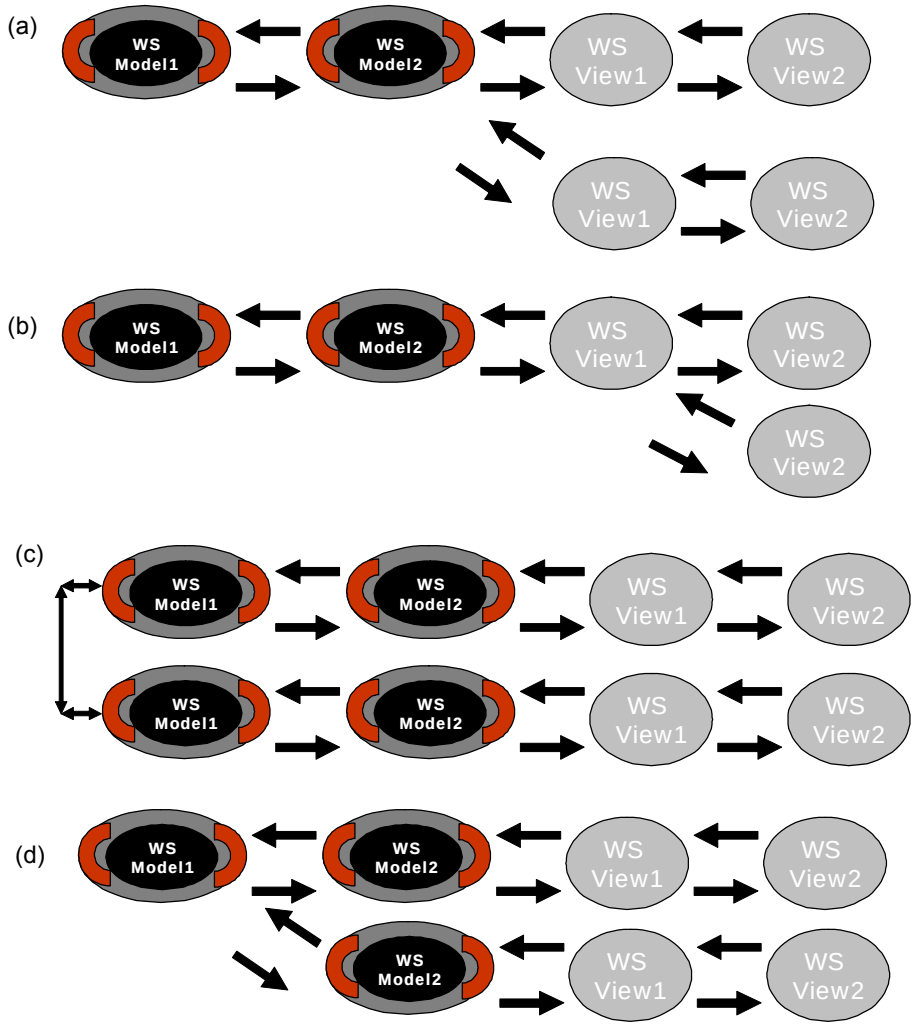


Figure 5.6: This shows an exemplar pipeline with 2 model and 2 view components and 4 different ways of breaking the pipeline. The case (a) corresponds to a basic SMMV situation and (b) would also be SMMV. (c) is MMMV while the classification of (d) is ambiguous.

Chapter 6

MVC decomposed SVG experiment

In this chapter we describe how be explicitly prepared a sample M-MVC desktop client by modifying the Batik Browser. The general approach is discussed in sections 6.1 and 6.2 while the following sections 6.3 and 6.4 describe features of the events used in Batik. The final section 6.5 gives some explicit details of the decomposition used.

6.1 Analysis of decomposition of Batik SVG Browser

Key features of the architecture of SVG and related applications can be derived from the MVC picture (fig. 3.2). We analyze all possible events of the SVG browser (ref. Table 4.1) and divide them into three types corresponding to the three stages of the pipeline in fig. 3.2(b). The event types are Raw Events (low level events including mouse and keyboard events), High Level UI Events (DOM/SVG events) and Semantic Events (application events such as shared SVG browser "Open file" events). Raw events 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 High Level UI events and semantic events to process in the *Model* and which in the *View* component.

When we look at the processes of interacting with and rendering in a SVG application, we can find that data typically flows through pipelines from one end (*View*) to the other (*Model*) and vice versa. There are many ways of decomposing the pipelines and currently we adopt three-stage pipeline architecture as shown in fig. 3.2(b). We can assume that

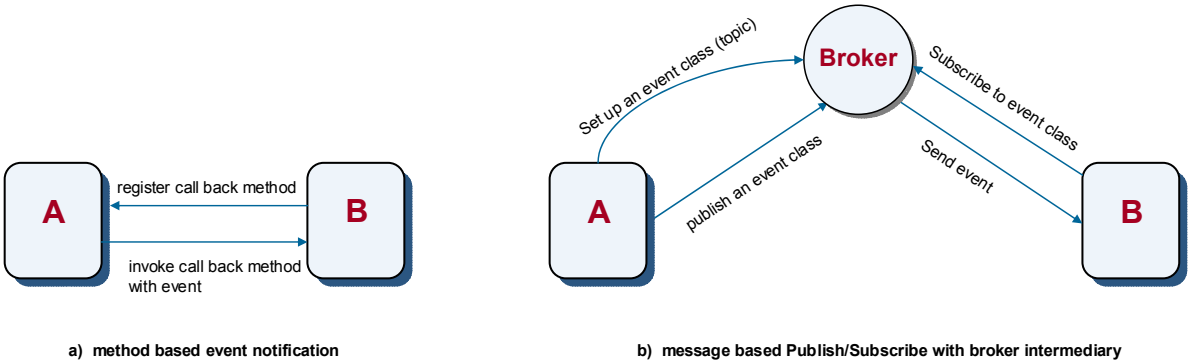


Figure 6.1 Method-based event notification versus message-based Publish/Subscribe with broker intermediary

each stage of the pipeline is a module that can be used independently or combined with others to provide a Web Service. Each stage also provides a natural synchronization point for a collaboration system.

The basic idea is illustrated in fig. 6.1. Traditional event-based programming is used extensively in the Batik SVG browser and most modern applications. Different parts of a program are linked asynchronously with one part producing events that are passed to listeners whose call-back method has been passed to the producer as shown in fig. 6.1a). This can also be implemented with explicit messages where listeners subscribe to an event class (topic) and events producers publish them to this topic. Our strategy is to replace the listener model by the Publish/Subscribe broker model of fig. 6.1b). Note that either approach can use explicit queues (maintained on a broker in the message case) or

alternatively integrate the broker into the producer as in most simple method-based event models.

In a SVG application, a complete pass of an interactive process starts with a UI event initiated by user input (e.g. a mouse click or a key stroke), interpretation and computation, and ends with an output mostly consisting of text or graphics for re-display of the updated image buffer. Mapping to the Web Service pipeline model (see fig. 3.2), Raw UI events represent mouse (or key) events while High Level UI and Semantic events imply DOM and application events. Each stage effectively is passed by twice during the procedure ─ one is along event propagation path; the other is on rendering approach.

In a "conventional" MVC, "*controller*" executes its tasks through method calls since messages are hidden in system level. We make a critical observation, namely "conventional" MVC has to be replaced by an "*explicit message-based*" MVC in order to of the application to be distributed. In our approach, we use "explicit control messages" to abstract the semantic meanings of "*controller*" so that messages of the original system are exposed and pulled into application level. Such abstraction generates structural changes as the following:

a) Original client application is physically split into client user interface ("*view*") and core functional component ("*model*"). The latter naturally becomes a Web Service on server side.
b) Method calls, which play the role as "controller" in a client application, are taken over by "explicit control messages" that communicate between client interface and Web Service server through the network.

c) Our approach requires us to support our model view linkage with a high performance messaging middleware infrastructure. Note that decoupled messages are exchanged via event brokers of our underlying messaging infrastructure, NaradaBorkering [NARADABROKERING], in a publish/subscribe scheme.

As depicted in figure 6.2, one can use this strategy in several parts of the SVG browser and in doing so produce multiple web services coordinated in a single application; there are natural event linkages between the client user interface and the GVT (or Graphic Vector Toolkit, an internal module to represent graphical view of DOM) tree used in Batik; another between GVT and the DOM tree and finally that between the DOM and the Java or JavaScript application. After substantial experimentation, we chose to split the SVG browser between the DOM and GVT tree. The resultant architecture is shown in fig. 6.2. This choice has the advantage that it naturally generalizes to other DOM applications. However we made for more pragmatic reasons as other choices appeared to require major restructuring of the existing software. Our search for appropriate places to split applications into message separated services illustrated two important principles.

- Firstly one should split at points where the original method based linkage involved serializable Java objects. Serialization is needed before the method arguments can be transported and this is familiar from Java RMI.

- More seriously we found that the Batik often involved large classes that implemented many different interfaces. These interfaces often came from different parts of the program and crossed the possible stages mentioned above. Such "spaghetti" classes as in fig. 6.6a implied that additional state information would need to be transmitted if we split at points where classes spanned interfaces from different modules. Of course the message-based paradigm (fig. 6.6b) tends to force a more restrictive programming model where all data is shared explicitly and not implicitly via interfaces crossing splitting lines.

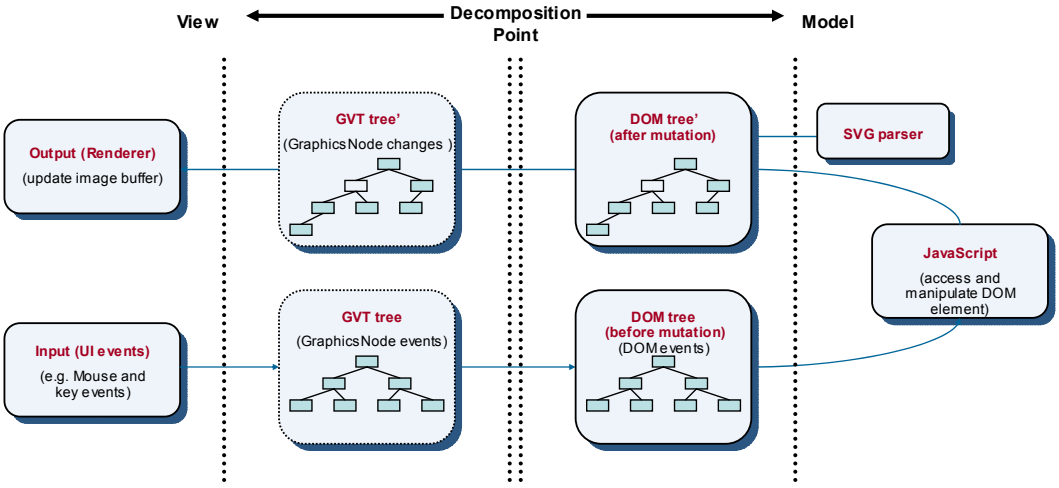
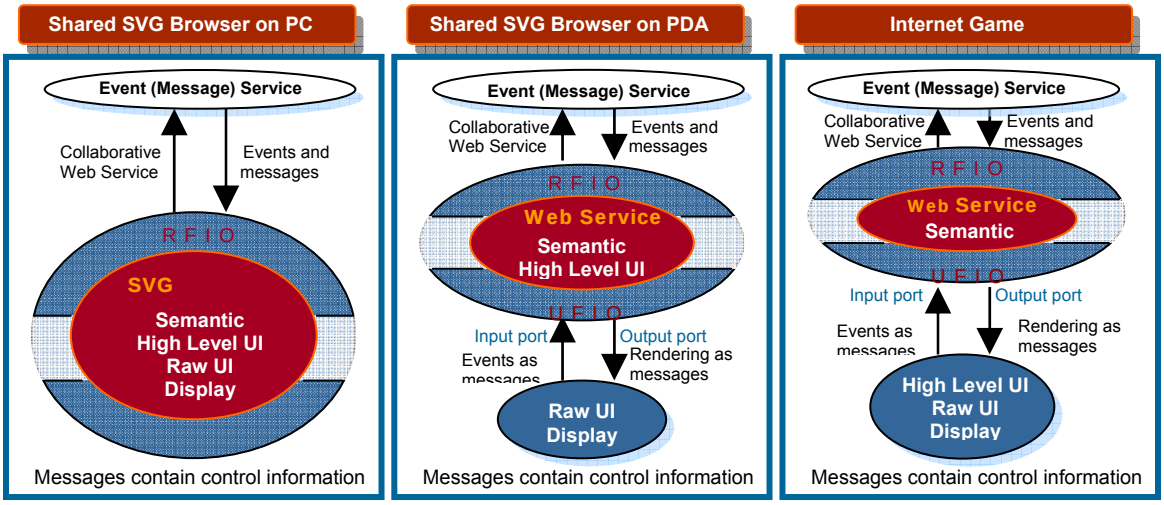


Figure 6.2 Decomposition of SVG browser in stages of pipeline

6.2 Architecture of decomposed SVG Browser in M-MVC paradigm

We discuss here three different ways of introducing explicit messaging into SVG. These are illustrative of several different possibilties. In the case of fig. 6.3(a), we aim at making original client side application collaborative with minimal changes to the source code structure. Master events are generated and replicated to participating clients through

the NaradaBrokering Message Service. We have elaborated this approach in monolithic SVG collaboration experiment in section 4.3. In the case of fig. 6.3(b), the heavy weight part of the computation is packaged as a Web Service that runs on a server thus make the client very thin. This design is optimized for ubiquitous access for SVG document over variety of hand held devices like PDA and cellular phone with bitmaps generated at



a. Non-decomposed collaborative SVG requiring minimal changes to the original source code

b. Decomposed WS optimized for thin clients

c. Decomposed WS optimized for performance

Figure 6.3 Three among the different ways of decomposing SVG between client and Web Service

server side and shared among clients. In the case of fig. 6.3(c), we have a high performance light weight version particularly designed for interactivity with compelling time demands of an Internet game. This chapter focuses on the latter case.

Note that we can 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. In the shared output port of collaboration service, a single *Model* instance uses NaradaBrokering to multicast rendering information to all collaborating *View* modules.

6.3 Analysis of User Interface generated events

Mouse events in the canvas area are the smallest grained UI events being capturing per pixel change. They enable various nifty interactive and animated functions. The trade-off, on the other hand, is that the fine-grained event-based control can dramatically increase the number of events for processing and add interaction overhead over the network. In the case of "move a piece" in chess, for example, a few to over several dozen mouse events may be generated depending on the path that a user drags the mouse. Therefore, we investigate in depth of the relationship between user interactive pattern and basic structure of mouse events here in this subsection and hierarchical event structure in section 6.4. The idea is to provide a common ground for decomposed SVG experiment and further event-based optimization in section 7.2.4.

In Java programming environment, an AWT event (UI event) is invoked when a user interacts with application GUI using an input pointer device such as mouse. In `java.awt.event.MouseEvent.java` class, mouse events are defined in following types:

Table 6.1 AWT Mouse Events

Event Type	Description
MOUSE_CLICKED	The "mouse clicked" event is generated when a "mouse pressed" and a "mouse released" occur in a row.
MOUSE_PRESSED	The mouse button has been pressed.
MOUSE_RELEASED	The mouse button has been released.
MOUSE_MOVED	The mouse position has changed.
MOUSE_ENTERED	The mouse pointer has entered a graphical component.
MOUSE_EXITED	The mouse pointer has left a graphical component.
MOUSE_DRAGGED	The "mouse dragged" event occurs when the mouse position changes while a mouse button is pressed.
MOUSE_WHEEL	The "mouse wheel" event is only used when a mouse equipped with a wheel has its wheel rotated.

We have illustrated in fig. 6.2 the event flow of SVG applications. JavaScript allows adding event listeners on SVG elements to invoke scripting functions (see lines 23 and 32

of fig. F.7). A mapping between major JavaScript events and corresponding AWT events are listed in Table I.1 in Appendix.

There're many possible ways that one can generate mouse events. We list in Table 6.2 typical user actions and corresponding events they fire to facilitate analysis of event interactions in interactive and dynamic SVG applications. Here, we refer as an element to any SVG graphical object rendered in canvas area. A mouse event suffixed with bracketed star (*) represents potential multiple entries.

Table 6.2: The relationship between a user interaction vs. AWT mouse events in SVG applications

No.	User action	Sample function	AWT mouse events
1	Click on an element	invoke hyperlink	MOUSE_CLICKED
			MOUSE_PRESSED, MOUSE_RELEASED
2	Drag an element	move a piece in chess	MOUSE_PRESSED, MOUSE_MOVED(*), MOUSE_RELEASED
3	Draw an simple shape	draw a line or curve	
4	Move mouse into an element	highlight a hyperlink	MOUSE_ENTERED, MOUSE_MOVED
5	Move mouse out of an element		MOUSE_MOVED, MOUSE_EXITED
6	Move mouse into and out of an element		MOUSE_ENTERED, MOUSE_MOVED, MOUSE_EXITED

The above table presents several interesting features of the relationship among user action and mouse events:

- A user action can be deployed in different ways. For example, a mouse click action in case 1 can be handled by responding either to MOUSE_CLICKED event or MOUSE_PRESSED and MOUSE_RELEASED events.
- Some basic event patterns are shared by different application functions. As shown in cases 2 and 3, superficially different application behaviors such as an animated "move a piece" in chess game and "draw a line" in whiteboard actually are based on the same MOUSE_PRESSED, MOUSE_MOVED(*), MOUSE_RELEASED pattern.

- Some actions fire multiple events. For instance, moving mouse into an element (case 4) invokes both MOUSE_ENTERED and MOUSE_MOVED events. However, not all events are associated with system behavior like cases in cases 5 and 6.
- Some AWT events are grouped sharing the same context, such as what occurs when a semantic event implies multiple atomic events (see fig. 6.4).

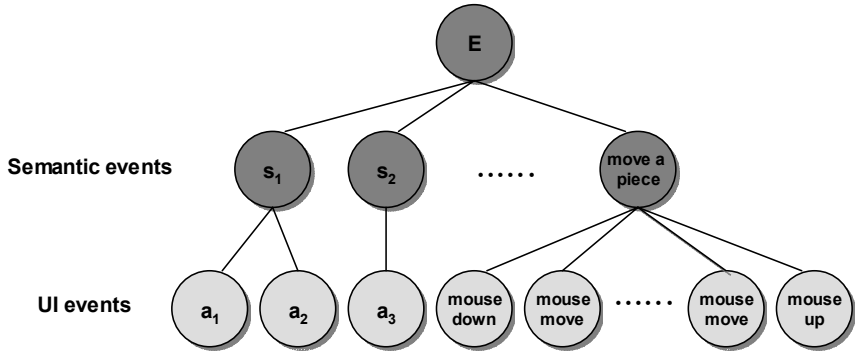
In order to reduce unnecessary event processing cost and network bandwidth, events with no significant role such as in cases 5 and 6 can be eliminated or compressed. At the meantime, events that are connected to the same semantic context should be packed into one message to aid effective processing. The pattern of MOUSE_PRESSED, MOUSE_MOVED(*), MOUSE_RELEASED in cases 2 and 3 shows a good example for such scenarios.

6.4 Hierarchical event structure

In an interactive application, there are a variety of events that are coordinated in driving system behavior. A complex system with modularized design is commonly decomposed into a hierarchical pipeline structure such as that presented in fig. 5.6. These events are generated at different stages in a pipeline and form a hierarchical event structure.

A hierarchical event structure is exemplified in fig. 6.4 for the situation of SVG applications. In general we start with a "root" event at the beginning of the processing pipeline and as the processing proceeds each stage generates one or more dependent events that form a hierarchical cascade. We can also expand this idea to include case where a semantic action includes more than one "atomic events". Dragging a mouse from

A to B on the screen could be an interesting semantic action; it can generate very many atomic mouse move and mouse over events.



6.4 Hierarchical event composition

For example, a semantic event such as "moving a piece" in a chess application encompasses the invocation of a sequence of UI events ─ "mouse down", "mouse move" … and "mouse up". The number of intermediary mouse move events reflects the distance of change in pixels. Likewise, "drawing a line" in a SVG whiteboard application embraces similar event stream. However, the semantic meaning and corresponding changes on the *Model* ─ SVG DOM are very different. The former triggers DOM mutation events with modification of x, y coordinate attributes of an existing graphics node, the piece. The latter adds a new graphics object, a "line", to the DOM structure.

This suggests that system behavior largely depends on application level events (or semantic events such as "move a piece") while the low level raw events (or UI events) mainly contains coordinates information. While it is difficult or perhaps even meaningless to compare application performance at semantic event level, it is possible to investigate system features by measuring at standard low level visual interaction, which

is based on UI event (mouse or key stroke event) that invokes corresponding data structure and graphical presentation changes.

6.5 Implementation

The diagram of decomposed Batik SVG browser in M-MVC structure is shown in fig. 6.5. It illustrates the design features of building distributed systems with simple services (ref. section 8.2 and [G. Fox]). Notably, we insert several event queues to buffer events between major SVG components, which include `JSVGCanvas`, `GVT`, and `DOM`. As introduced in section 6.1 (ref. fig. 6.1), replacing method calls with explicit messages allows decoupling of system classes with natural event connections. The event queues not only facilitate distributing small grained components as services, but also provide other two important supports ─ sequence events and performance optimization.

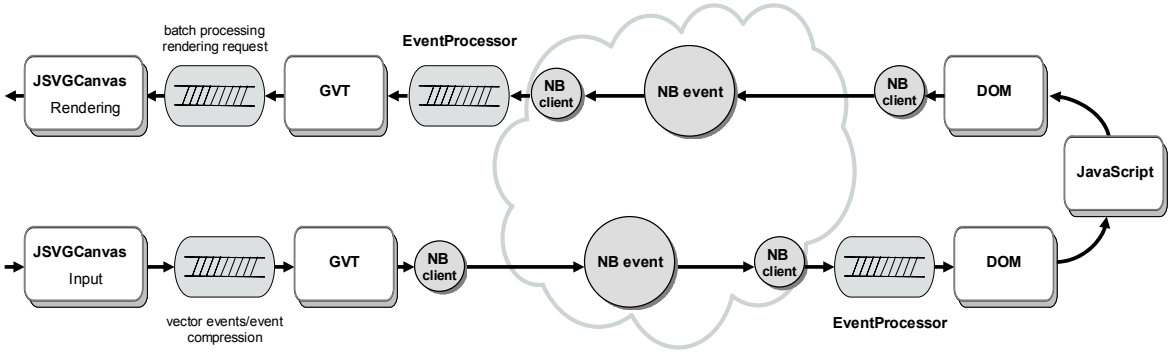


Figure 6.5 Decomposed SVG Browser in M-MVC paradigm

The decoupled `JSVGCanvas`, `GVT`, and `DOM` components constitute the three stages along the pipeline. They supply nature synchronization points for collaboration. The `JSVGCanvas` and `GVT` stages generate sequential events, which are invoked by AWT events from user interactions and propagated along GVT tree respectively. However,

DOM is not synchronized in Batik implementation. Namely DOM structure changes and the DOM Mutation events produced thereafter are not synchronized when triggered by UI events. Since one user interaction could invokes many atomic DOM modifications, fig. 6.7 shows that unsynchronized DOM Mutation events e_1, e_2 ... e_n are sequenced before feeding into the pipeline at GUI side for rendering.

In general, a pipeline structure expects events are all sequenced. As Batik implementation is not full-fledged in this regard, there exist some constraints of application functions. For example, JavaScript loop function "timeout" can generate unpredictable results in terms of the number of actions and the time they consume and it is not supported in Batik. The Batik's algorithm of DOM event processing and GVT update rendering works fine as long as all classes are tightly coupled sharing the same context such as local memory and unpredictable operations are prohibited so that each group of user invoked computations are guaranteed to complete in a short enough period of time. However the possible lack of sequencing of events is more problematic to MVC decomposed SVG for the distributed implementation. Disparate CPU processing capability and network latency add more uncertainty to the system runtime behavior, and represent some of the well known hard problems that distributed applications commonly encounter. We note that NaradaBrokering supports guaranteed delivery of messages in the order they are delivered to the system. Thus problems can occur not to the bad ordering but to change in correlation of events if their relative timing gets distorted. This can be addressed by good system design and well identified events. However Batik was not in our opinion well architected in this regard. These issues did not affect the chess application used in our tests.

The event queue between `JSVGCanvas` and `GVT` stages is devised for performance optimization. The class of `org.apache.batik.gvt.event.AWTEventDispatcher.java` dispatches an AWT event to its containing GraphicsNode of the GVT tree by coordination information. We've discussed in section 6.3 that although a large number of UI events are produced, not all of them are relevant to the application behavior. Therefore, it is beneficial to compress unused events at this early stage to reduce computation and network transportation cost. Groups of events can be identified for a vector event based on common event patterns (ref. sections 6.3 and 6.4) although this can also be done at a later stage when events are packed into a message payload for network transmission.

The `EventProcessor` component is designed to buffer and process all types of events. It is used at both event propagation and rendering legs of the pipeline. Currently M-MVC implementation provides one event queue (`EventRepository`) that hosts all types of events (ref. Table 4.1) and the `EventProcessor` access and process events in a synchronized and sequential manner. Events with different priorities are treated differently. For example, ControlEvent has higher priority than other Batik events thus is handled first. The `EventProcessor` also interprets event flags to identify events that share the same semantics context and schedules for batch processing of the events in a group. The `EventProcessor` in charges of forwarding events (e.g. to DOM in propagation leg), invoking Batik methods (which is conducted by UpdateManager via Runnable queue), and interfacing with NaradaBrokering via NB client to upload and unload messages for message passing over the network.

We note that events exist in different layers of the system stack. In the case of M-MVC decomposed SVG browser, there're two levels of event: system events like AWT

event and application events (including Swing events, Batik events, and collaborative SVG events). At meanwhile, we refer different types of event such as UI, high level and semantic events (ref. Table 4.1) to accommodate to different stages of the pipeline structure. The latter can be mapped to AWT event, SVG/DOM event, and application event.

At each level, there may be one to many event queues hosting the events. There may be only one system event queue while each application may have its own event queue. The operating system is responsible for making sure the right events get to the right programs. Java virtual machine has one main AWT event queue (`java.awt.EventQueue`). To access the native system event queue, a Java program (but not applet) use the `Toolkit.getDefaultToolkit().getSystemEventQueue()` method in the `java.awt.Toolkit` package. Swing and Batik both have its own event queue. We regard the Runnable queue that is managed by UpdateManager (ref. fig. H.13) as the event queue of Batik, which controls the system event flow. Especially, it coordinates between DOM changes and update rendering for proper responses to user invoked interactions. We keep the Batik queue in our implementation (e.g. for batch processing rendering requests) but limit its usage as a local queue rather than a global event structure.

For distribution purpose, we convert Batik architecture from a single global event queue into multiple localized event queues in M-MVC decomposition. Both tightly coupled (e.g. standalone or desktop systems) and distributed interactive style applications require an event approach, which implies that an event queue structure is needed for storing each user interaction invoked events (e.g. mouse or key events). In a broad sense, our experiment with SVG decomposition suggests an effective approach to change at

architecture level from a tight coupled MVC model to a distributed M-MVC pipelined model. We note that it elegantly support major collaboration paradigm (ref. sections 3.4, 5.4 and 5.5). The design of the needed distributed queues both at application and system (messaging) layer needs further research.

Batik separates the structure of the SVG DOM and GVT. However, they share BridgeContext as shown in fig. 4.3. The Bridge package provides critical mapping information between DOM tree nodes and GVT tree nodes that facilitates processing of SVG document and rendering of GVT tree. We refer as "shared state" in fig 6.6 to the global information shared by different parts of the system. The class interfaces and variables that are used by Batik across the system pipelines make the conversion to the distributed model harder as M-MVC structure tend to force a more restrictive modularized programming model.

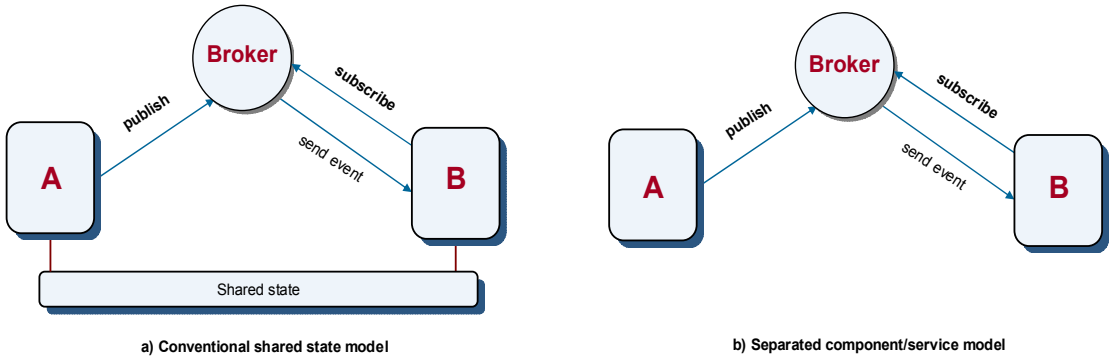


Figure 6.6 Implicit and explicit state

We employ a mirrored DOM structure at client side. In this way, we can achieve M-MVC with separated service model, which shares data explicitly and not implicitly via interfaces crossing splitting lines while avoid dramatic modifications of the Batik structure. Fig. 6.7 depicts the event flow chart that discloses the relationship of user input and system behavior in a fine grained event-based interaction.

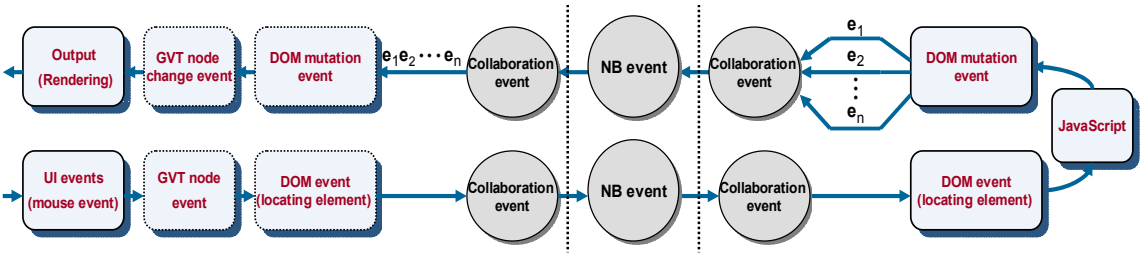


Figure 6.7 Event flow chart of SVG applications

There're two directions of event propagation in a pass of event processing, *View* to *Model* and *Model* to *View*, which accomplish separate functionalities.

- One direction is mainly for receiving UI events and locating the event target in the *Model* at server side. The GVT tree helps converting a mouse event from its device coordinate system to user coordinate system.
- As SVG applications add event listeners on the DOM model, this leg starts with the targeted node(s) invoking call back methods of listener's class with fired event(s). Typically, designated application functions (e.g. JavaScript) manipulate DOM structure that produces multiple DOM mutation events including insertion, removal, and modification attributes (e.g. x, y coordinates and color) and text value of the nodes. In response to the DOM changes, GVT tree keeps track a "dirty list" of graphics nodes that need.

We use "event stream" to describe the continual flow of events between *View* and *Model* ends. The composition of an event stream is largely depended on the pattern of system and user behaviors. Considering an example, when the user initiates a semantic event "move a piece"; then the input device invokes a stream of User Interface (UI) events ─ mouse down, mouse move… and mouse up. The number of intermediary mouse move events reflects the distance of this piece being dragged in pixels. A system state

change request propagates along the pipeline and communication between adjoin stages implemented as event-based messages. Therefore, one can describe the system interaction in terms of its events as shown in fig. 6.7.

Along the path of event propagation, an event is processed before being forwarded to the next stage. The performance of a pass for a mouse event, therefore, is composed of the latency at each stage and the connection cost between the stages. Presumably, the cost of a semantic event is the sum of each pass invoked by the UI events making up the semantic event. Nevertheless, some computation such as DOM mutation occurs in parallel as shown by events e_1, e_2 ... e_n in fig. 6.7. We present in the next chapter a performance testing model to measure the details of the event-based interactive approach.

The fact that common operations such as JavaScript “timeout” are forbidden in Batik suggest that the model is incomplete. It is not surprising that general M-MVC approach needs one to extend the Batik infrastructure. However, for the purpose for this thesis research of M-MVC, we just implemented with such constraints as we are not producing a new production SVG browser but rather exploring research concepts. This restriction was not a problem with the chess application used in the tests of chapter 7.

Chapter 7

Performance and Analysis

We have started an extensive series of performance measurements to demonstrate the viability of our approach. The main purpose is to identify key factors that influence the performance of M-MVC in particular and message-based model in general of building distributed applications. This section includes a complete description of the testing approach and presents key measurements.

7.1 Test Scenarios

Our investigation is carried out based on experiments with conversion of Batik SVG browser [BATIK], a stand alone client application from Apache. The process of converting it from a tightly coupled method-based desktop system to a loosely coupled message-based system with distributed *Model* and *View* components provides us a unique opportunity for an in depth understanding of the structures from both application domains and how architectural changes impact system functionality. The implementation used in these tests is fully described in chapter 6.

While one can employ various interactive applications built on Batik SVG API for performance measurement, we use the same chess application as we used for testing our collaborative model [QCF-07-03] so as to have a consistent experimental approach. Game applications allow one to generate real-time system processes in a highly

interactive manner. Many advanced facets of system design including graphical quality, user action processing capability, group communication efficiency and reliability, game engine robustness, and overall system integration and coordination can be tested in a comprehensive fashion.

Messaging plays a centric role in providing a software level communication channel that connects distributed components together. M-MVC employs NaradaBrokering [NARADABROKERING] as the underlying messaging infrastructure. 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 as messaging services (see fig. 1.3). Here are using messaging between the model and view of a traditional desktop application.

We adopt the decomposition strategy of the *Model* and the *View* of Batik SVG browser as delineated in fig. 1.2. 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

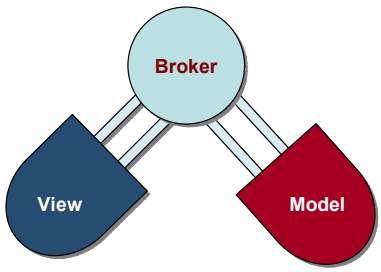


Figure 7.1 Single Model and View linked by messaging broker

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 fig. 7.1.

We list scenarios for a set of performance tests: environment settings in table 7.1 and system configurations in table 7.2. 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 remote site in campus or inter-city area. In test 1 to 6, the Broker either shared the same runtime environment or crossed operating system platform in communication with *Model* and *View*, where both were run with Windows on desktop computers. Hosting computers of Broker and *View* also alternated to delineate the influence of CPU processing power.

Table 7.1 Testing environment settings

Test scenarios		**Environment Settings**					
No	**Description**	**Event Broker** (NB0.97 Server)	**View** (Client)	**Model** (Service)	**Network connection type**	**Broker distance**	
						area	hop
1	Switch connects Desktop server	desktop2	desktop1	desktop2	direct switch	10 meters	1
2	Switch connects	desktop3	desktop3	desktop2	direct switch	10 meters	1

	High-end Desktop server						
3	Inter-City (Campus) area Solaris server	solaris (ripvanwinkle)	desktop1	desktop2	routers	40 miles	n/a
4	Office area Linux server	linux (gridfarm1)	desktop1	desktop2	hub	100 meters	1
5	Inter-City (Campus) area Linux cluster node server	linux HPC cluster	desktop1	desktop2	routers	40 miles	n/a
6	Inter-City (Campus) area Solaris server	solaris (complexity)	desktop1	desktop2	routers	40 miles	n/a

Table 7.2 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 (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

7.2 Timing Considerations

7.2.1 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.

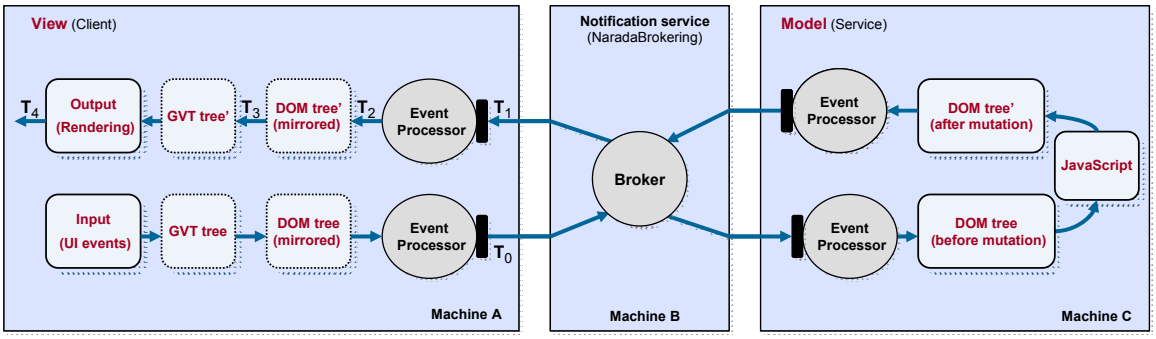


Figure 7.2 Performance testing and timing points

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 fig. 7.2. 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. When the system is active, continuous events are pumped in and propagate along the pipeline and invoke system state changes. However, as soon as user interaction stops, the application returns to its inactive status. The event flow, at high level, is pretty much like current flow in an open or closed circuit system. 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 at each of the marked timing points T_0, T_1, T_2, T_3, and T_4 in fig. 7.2 to scrutinize the cost and characteristics of the modules delineated by these timers, and we will give further explanations in following subsections.

7.2.2 Measurement Units

Considerations have to be taken as to at what granularity level conducting the performance measurement. The level of granularity affects the type of events available for observation, which further influence the structure of model classes [Veit+Herrmann]. In the SVG experiments, the timing model is built upon a refined multiple stage pipeline structure contained within MVC framework. This allows us to observe inter-component interactions at fine grained level within the M-MVC structure. We identify that an event has hierarchal composition that reflects the stack of system design described in chapter 6 (see fig. 6.4). The modeling is based on input event invoking interaction on GUI components, which correspond to element node of the SVG DOM structure. The semantics of an event comprises of the contextual information about a graphical object. These events drive the system state change.

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.

This approach has two advantages: firstly, it provides a common ground for examining event process at a fine grained level; secondly, it supplies a quantitative method that its measurement results imply essential architectural and environmental features influencing semantic event (application level event) and overall system performance.

7.2.3 User-perceived performance constraints

It is known that human visual system (retina and brain) retains an image for a fraction of a second after it views the image. Simply put, human eye can not detect a visual

change within about $1/20^{th}$ to $1/30^{th}$ of a second. This phenomenon is called *visual persistence*, which is essential to all visual display technologies. It has been understood since the first days of movies [A. Huk]. 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.

Human perceptual sensitivity to latency of human-machine interactions puts stringent time constraint over system design and is especially challenging to distributed media rich applications. This is due to notable 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 discuss network performance in the next subsection and this 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 [A. Uyar], this is an order of magnitude longer than the delay associated with visual persistence.

7.2.4 Performance optimization

Involving network for message sending almost always cause delay of system interaction. How should one improve the efficiency of communication for interactive applications? In real systems, people design various enhancement technologies to boost performance, which include buffering (or caching), pre-fetching, compression (or optimized codecs), optimization algorithms (e.g. pattern search, rasterizing, art effect, and font support) that tweak graphics rendering. High performance messaging is another achievable goal [HIGHPERFORMANCESTREAM] over current network transit latency at a few milliseconds of local area network (e.g. intranet of organization area) [RULEOFMILLISECOND] and 100's of milliseconds of the internet scope (e.g. transcontinental links). On reliable high latency links, one can get better performance by replacing TCP by UDP and using application level fault tolerance; another approach to improving bandwidth but latency is parallel TCP streams as popularized in GridFTP and other fast FTP modifications. Both of these ideas will be supported in NaradaBrokering and using UDP could be helpful in some circumstances but we do not explore this here.

Due to the central role that events play in deployment of interactive style applications (ref. section 2.1.1), in this thesis, we propose two event-based methods to optimize application performance: one is "vector event" and the other is "event compression". Packing small messages into a single larger one for transmission is a common way to

reduce network transportation overhead. Likewise, uploading multiple events into one message payload constitutes a vector event that can be unpacked at receiver side. Theoretically, any mouse event can be associated with specific action that changes system state. However, not all events have the same significance to application behaviors, which suggests that it is possible to eliminate some events without impacting the essential functionality. Based on the analysis in section 6.3, one can choose to use "vector event" and "event compression" methods independently or combined to optimize performance. As a reference, we provide analysis of typical mouse events and their impact on overall system performance in Table 7.3 described a little later.

7.2.5 Semantics of timing points

When measuring the duration of a method call, one can either stick with the simple elapsed time, that is the clock difference between method entry and exit, or make the best effort to correct for the multi-threaded nature of the JVM. In the estimated CPU time mode, the data of the thread status sampler and the cumulated CPU time of each thread as reported by the operating system are used to weight all method calls. For both modes, a calibration phase at startup calculates run-time parameters for a self-correction algorithm which account for the times used by the profiling process itself. We address this issue indirectly by timing performance of the some of the key operations with and without other threads running. As we will see the elapsed time of the messaging is greatly impacted by interference with other threads. This complicates the measurements of overheads as the simple clock differences give you an overestimate as they record the pure overhead time plus the time of concurrent threads that do not represent overhead.

We present here four timings for each of the test scenarios with the timing positions shown in fig. 7.2 which is a simplified version of the pipeline shown in fig. 6.2. The results in table 7.3 give mean, the error in its determination, and the standard deviation. The times T0, T1, T2, T3, and T4 are all measured in the View and defined as follows:

- 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

7.3 Performance measurement and analysis

The performance tests are designed to investigate overall performance of message-based MVC; the cost of messaging and interfacing between application and underlying messaging infrastructure; relationship of application behavior and functionality in a fine grained view, and the influences of environment settings. Since interactive style applications involve human and computer interactions, our performance measurement and analysis reflect the impact of both indispensable factors as well.

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: testing environment settings in table 7.1 and system configurations in table 7.2. Tables 7.3 to 7.5 contain a selection of measured data.

Table 7.3 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	48.90± 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	26.0	336.7 ± 22.0	189.0

Table 7.4 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 7.5 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

The results tables 7.3 to 7.5 record times between the processing markers T_0 T_1 and T_4 shown in fig. 7.4 (times for other markers are given in [M-MVC]). Figures 7.3 through 7.8 give detailed histograms extending the results of table 7.3. 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*.

In tables 7.4 and 7.5, we used the same JavaScript chess program described in earlier papers [QCF-07-03] [QIU-09-04]. 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 7.4 and 7.5 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 7.3, 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. Table 7.4 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*. These bounce back events are solely to help us understand better how much time is messaging overhead and how much is time spent automatically in the model. Table 7.5 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); the 4 hop

events correspond to messages going from *View* location to NaradaBrokering to *Model* location and back. In all cases for table 7.5, 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 7.5 records the natural overhead from NaradaBrokering without significant thread interference. This is about 2 milliseconds per event but is increased in some entries in table 7.5 and in the bounce-back event of table 7.4 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.

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 7.4 and 7.5 (Bounce back time versus 4 hops), the two tables are measuring the same computation and communication time but table 7.4 is 10-20 milliseconds longer than table 7.5. This can be explained by the large (extraneous to message passing) computations on the *Model* and *View* in table 7.4 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 7.4 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 7.4 with the 4 hop measurements of table 7.5. Comparing columns 1 and 2 of table 7.3 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 7.4 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*.

The following group of six graphs show detailed performance comparisons of average mouse events, mousedown event, mouseup event, and mousemove event corresponding . While in columns one and two of table 7.3, we only listed message transit

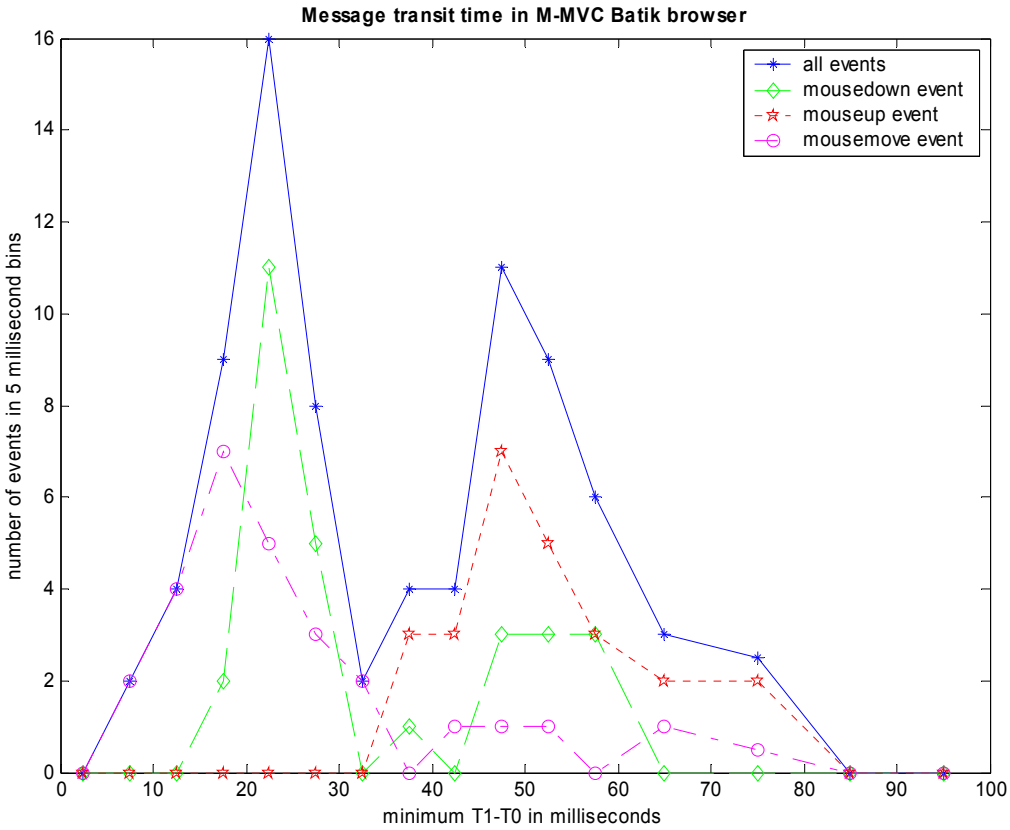


Figure 7.3 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 1 of Table 7.1 and the row labeled 1 in table 7.3. The configuration is in detail: 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

time for the mousedown events and the average of all events between T_1 and T_0 (first returned and sent) for each testing case.

Figure 7.3 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 useful to compare these results with those with the faster desktop in figure 7.4. The better

performance of the next figure represents about 2 years of “Moore’s Law’ improvement and illustrates our thesis that our architecture will get more attractive as computers continue to improve in performance!

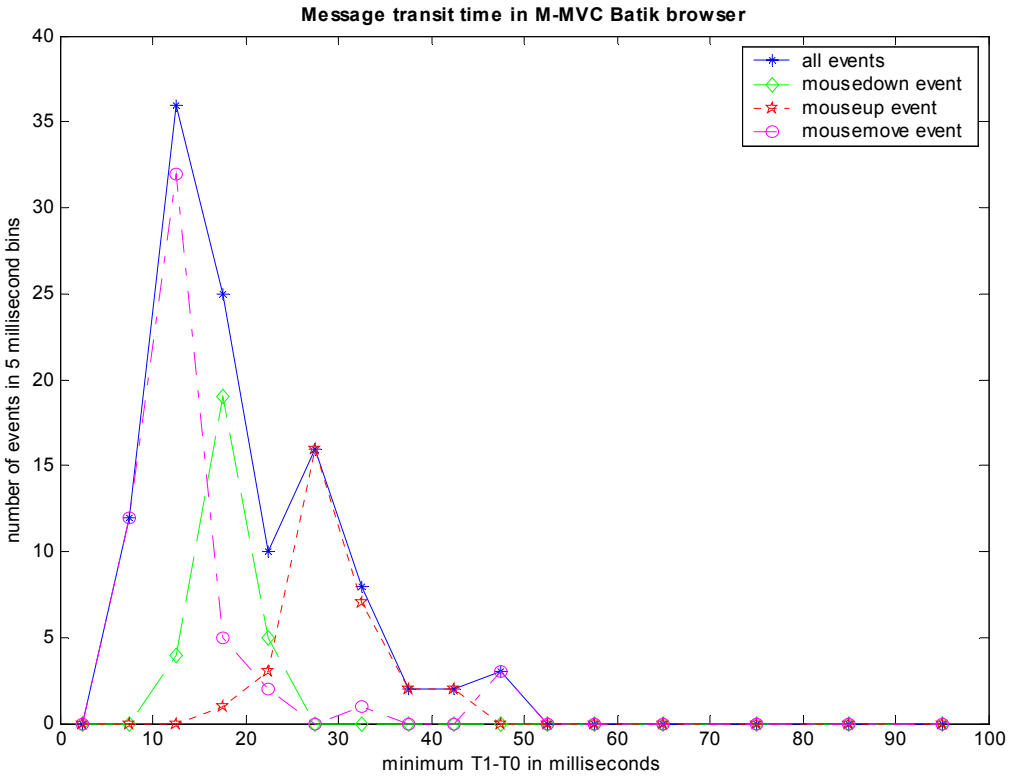


Figure 7.4 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 2 of Table 7.1 and the row labeled 2 in table 7.3. The configuration is in detail: NB on View; Model and View on two desktop PCs with "high-end" graphics Dell for View; local switch network connection. A few events with timing greater than 100 milliseconds are not shown on the plot

Figure 7.4 shows the best results of the set and highlights the importance of the client; this was a higher-end Dell desktop than the rather old 1.5 Ghz clients used in the other runs. Of course the 3 Ghz Pentium used in this desktop is now commonplace and one should consider this to be expected performance on a modern desktop. The means and standard deviations are substantially reduced from the previous figure while the relative performance of the different types of events is unaffected. Note that for the chess application used the view computation is greater than that needed in the JavaScript model component. One interesting deduction of this set of measurements is that the choice of client is more important than the server which can be moved from Windows to Linux or

to Solaris and from local machine to a server in the next town. This gives much less effect than switching from a 1.5 Ghz to 3 Ghz client.

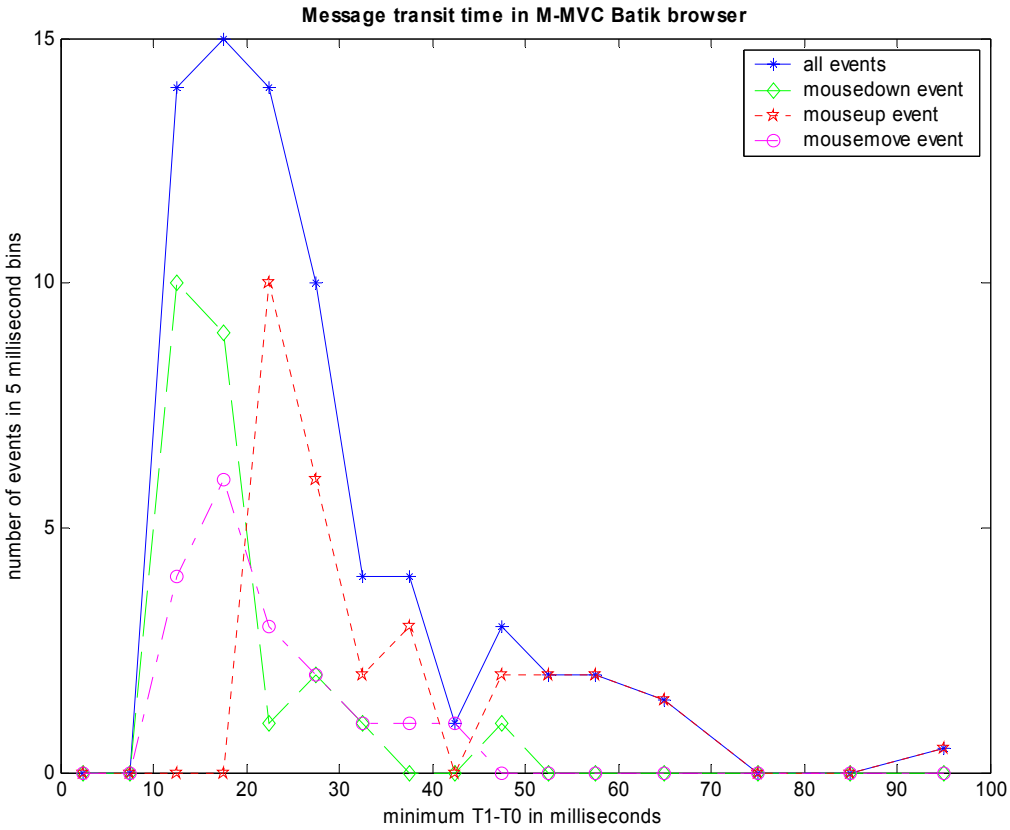


Figure 7.5 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 3 of Table 7.1 and the row labeled 3 in table 7.3. The configuration is in detail: NB on 8-processor Solaris server; 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

Here in figure 7.5, we see good performance even for a server located in Indianapolis 40 miles from the two desktop machines in Bloomington. We note that we give results averaged over events of a single run (i.e. a single game of chess). The results from run to run differ – presumably due to other applications on the desktops and servers – but the same systematics are seen.

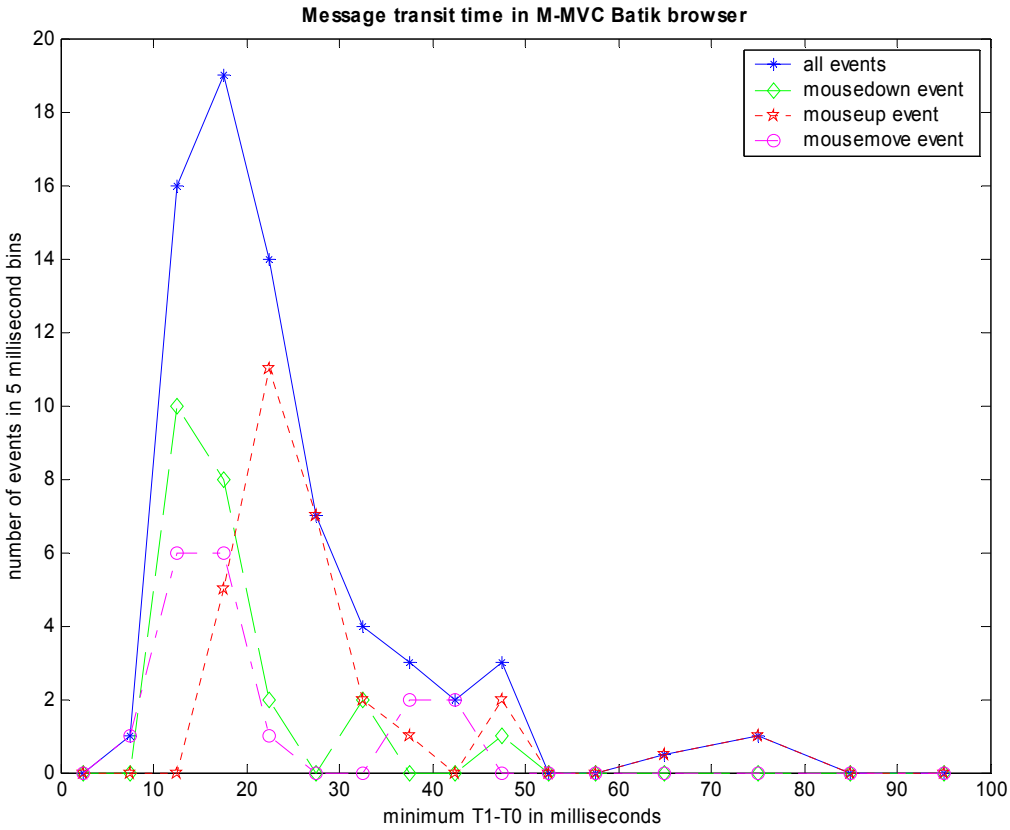


Figure 7.6 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 4 of Table 7.1 and the row labeled 4 in table 7.3. The configuration is in detail: NB on 2-processor Linux server; 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

The case in fig 7.6 is a very practical one; two modest desktops served by a local low-end Linux server. One finds excellent performance. We find it interesting that one usually gets better performance moving the NaradaBrokering broker off the desktops; the better broker performance (there are no scheduling overheads) outweighs the increasing network overhead. The typically good performance for the three scenarios with the NaradaBroker situated in Indianapolis highlight that excellence of modern institutional networks.

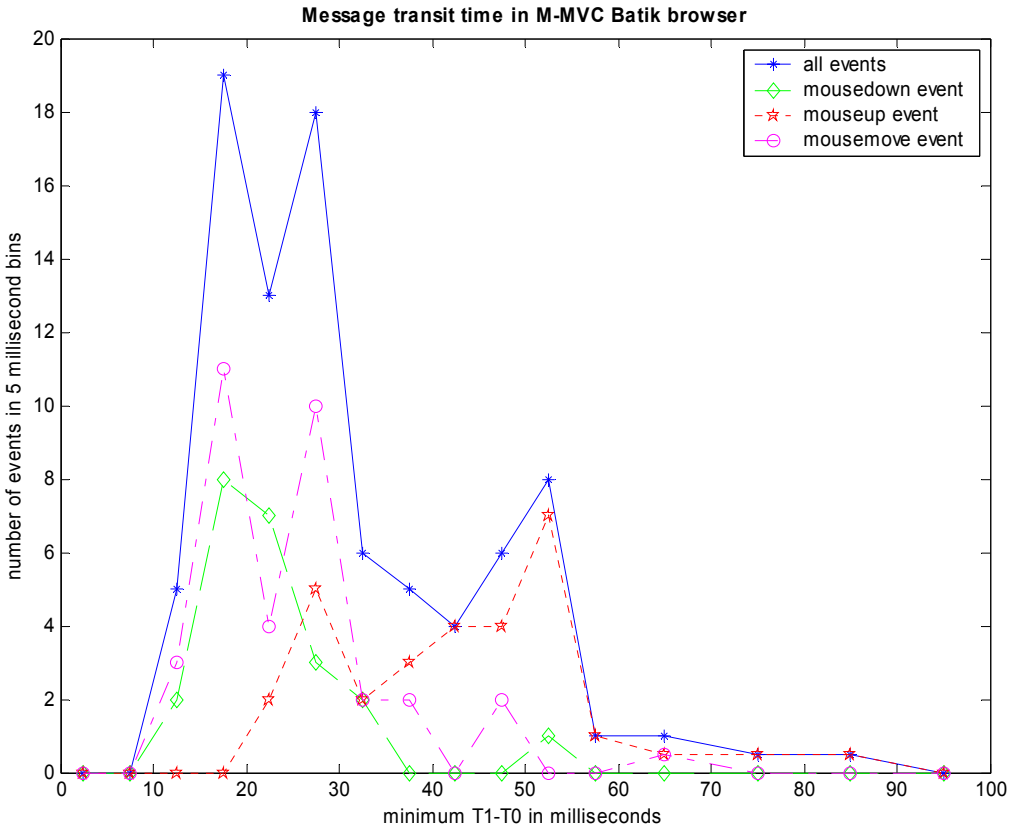


Figure 7.7 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 5 of Table 7.1 and the row labeled 5 in table 7.3. The configuration is in detail: NB on one node of HPC Linux cluster; 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

The plot of figure 7.7 uses a HPC Linux cluster as the NaradaBrokering server. Of course we are only using one node and so this is not a parallel computing application. It does illustrate that this type of HPC engine can be used in Web Server mode with each node running different services.

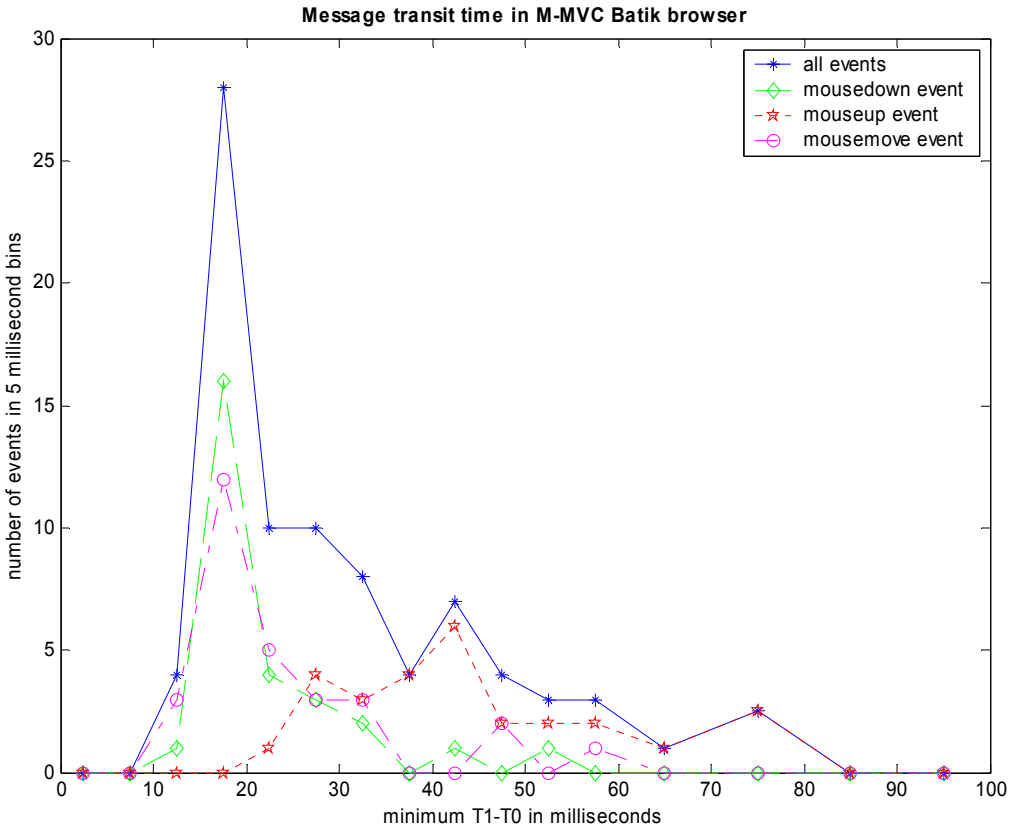


Figure 7.8 Histograms of the elapsed time T1(first event to return)-T0 for three types of mouse events and the set of all mouse events which is just the sum of the first three histograms. This data corresponds to test case 6 of Table 7.1 and the row labeled 6 in table 7.3. The configuration is in detail: NB on 8-processor Solaris server; 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.

The configurations of figures 7.5 and 7.8 are similar with the server "complexity" of the last figure tending to be more heavily used than "ripvanwinkle" used if figure 7.5. The differences between these figures are most pronounced for the mouse up event which has a very broad distribution in figure 7.8. This probably stems from poor performance of the desktop holding the model for this run.

7.4 Summary

These first 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.

The performance results we've presented through experiments with SVG applications however do not suggest that general M-MVC application implementations would generate exactly the same performance data. Nor do we try to achieve the most optimized performance for these particular experiments at this stage. Rather, our approach has been focused on investigating viability of the M-MVC approach and factors that impact overall system functionality and performance when the architecture changes from tight-coupled to loose-coupled model. The aim is to identify general issues for the design and implementation of service-oriented message-based applications, particularly embracing areas of messaging correlated technology, human-machine interaction, and environment. As a systematic experimental approach, it helps to clarify the ambiguities or subtle differences between method-based MVC and message-based MVC approaches and provides resources with preliminary data that ca be used in design decisions.

Chapter 8

Architecture of Collaborative Message-based MVC

8.1 Lessons learnt

We have studied both standalone and collaborative applications in both traditional MVC and Message-based MVC architectures. Our results suggest that the performance of M-MVC applications is acceptable and that M-MVC becomes very attractive due to its great advantage in developing cross-platform and collaborative capabilities. We learnt several important lessons from studying the Batik Java code.

We think it is reasonably clear that although many modern applications adopt MVC (Model-View-Control) paradigm, they may not strictly follow the principles in implementation. By not faithfully following modularized design principle the applications do not have a clear "control" mechanism which includes a well defined explicit communication channel between "model" and "view". Usually, there exits direct linkage (not through a well defined "*control*") between the *Model* and the *View* in the MVC paradigm. For example, some functional modules in Batik involve information from many stages across the pipeline of Section 3.3, rather than just providing the linkage between the two modules. Often capabilities are instantiated in GUI ("*View*") but execute functions in the backend "model" through direct methods calls. We suggest that MVC is popular and the role of the model and view relatively well defined. However the architecture and implementation of the control mechanism between these components is

less standard. This thesis has explored and advocates use of a publish/subscribe (listener) architecture for the control between loosely coupled (service oriented architecture) model and view. We recommend designing the model-view linkage as messaged based to get M-MVC. Note that a message based linkage can be implemented with method calls if one wishes to get greater efficiency when model and view share a common address space. This is familiar from MPI in parallel computing that can use explicit messaging on distributed memory and pointer manipulation and method calls on shared memory.

Further there are difficulties from the Batik software not providing "Object Serialization". Object Serialization (the process of reading and writing objects) has many uses, including remote method invocation (RMI). In addition to the object streams, `java.io` has other classes and interfaces that define the API to help classes perform serialization for its instances. However existing java codes like Batik, use data structures that are not easily serializable and this also makes it hard to split the application into a Web service. We expect other software has similar problems and recommend the interaction between model and view use serializable classes.

We found good performance in a message-based version of Batik described in chapters 6 and 7. However there were important issues that needed attention. We found that the “natural” delay of one to two milliseconds in the messaging infrastructure (NaradaBrokering) was often impacted by the thread scheduling in the operating system. This was apparent in our Batik case as NaradaBrokering was running in same machine as some very compute intensive rendering. This effect had not been seen in previous uses of NaradaBrokering which had run this software standalone in its “own CPU”. It is not yet

clear how important this effect is. It affects the *performance measurement* a lot; however the impact on *user-perceived performance* is less. One is attributing to NaradaBrokering a delay that is inevitable anyway and present in the original Java code. However we think this effect could be important in new generations of desktop machines and their operating systems. For instance multi-CPU and multi-core CPU desktops are of growing important. We expect a growing use of explicit messaging and suggest that this could be assigned to a dedicated CPU (core).

8.2 Proposed architecture (how would one code from scratch)

We combine the M-MVC ideas with those of service-oriented architectures [Atkinson et. al.] and the concept of "Grids of Grids of simple services" [G. Fox]. The latter describes the principle of simple services which are "as small as possible" subject to the communication overhead of breaking capability out as a service being acceptable. This concept is another analogy with parallel computing where we know that acceptable efficiency requires that problems be large enough so that the decomposed parts are large enough that computational work within each node is large compared to the inter-node communication [Fox94]. Usually communication compared computing for a system component is a "surface" effect and the ratio of communication to computation decreases like (system complexity)$^{1/d}$ for an effective dimensionality d. As analyzed by Fox, this concept is more general than the geometric structure of science and engineering simulations with d defined as an information dimension which may or may not be the same as the geometric dimension [Fox94]. In our case it says that are usually many ways of defining model and view and as seen in figures 3.2(b), 3.3 and 5.6, possibilities of breaking up the model and view into multiple components. In the simple service concept,

one breaks up the full system into simple services defined as above. This is not well defined for the size of the communication between simple services is obviously dependent on the implementation. So we somewhat quantify this concept by requiring that the communication be implemented by publish-subscribe mechanisms and preferably with serializable exchanged objects. We leave moot the implementation of the service linkage which need not be explicit messages but some optimized "shared memory" model. Further we do not want these ideas to replace existing software engineering principles and component models. Rather they are principles to be defined for components that can exploit the M-MVC model. That is for components that need to be isolated so they can be re-implemented on different clients and servers, run in distributed fashion, or whose events must be exposed to support collaboration. Traditional software engineering techniques should be used inside the simple services. We note that our work suggests it would be interesting to research publish/subscribe architectures that can be interoperably run in efficient fashion on shared and distributed memory (system) fashion.

Thus we propose building all applications as simple services with M-MVC corresponding to the *View* being one or more such services. This then unifies the web and desktop models for applications.

8.3 Comparison of Batik SVG Browser with proposed architecture

This application has an MVC architecture but was not designed in a way to make it easy to break it into simple services linked by messaging. In particular there was one particular Java class (corresponding to the Bridge capability) that "ran throughout the application" like a piece of Spaghetti. As well the primary signature, it had many different

entry points implementing interfaces reflecting capabilities spread throughout Batik. This Spaghetti Java structure made it very much harder to decompose Batik in an elegant way. As described in chapter 6, we successfully implemented an M-MVC architecture but this could have been much easier! We see no reason why an SVG browser could not be designed in a fashion where for example the modular structure of fig 6.2 is properly reflected in the Java code. The resultant system would be easier to maintain as well as having the M-MVC advantages of supporting collaboration and easier portability to different platforms.

Chapter 9

Conclusions and Future Research Issues

9.1 Thesis Summary

The concept of Service Oriented Architectures and Web Service technology in particular provide a general platform that promises to maximize interoperability and reusability for next generation of software applications. However, the issue of software system integration and convergence is only addressed for applications that are intrinsically loosely coupled from their geographic distribution. This thesis explores an approach that builds a message centered application architecture spanning desktop to Internet applications. By doing so, it prepares a service-enabled application ready to be either conveniently integrated into Web Service platform or to run on a desktop or even with a handheld interface. This approach could make universal deployment of and access to heterogeneous software assets a reality.

We believe our prototype showed directly how a message-based MVC (three-stage pipeline) model can generate a powerful application paradigm suitable for SVG and other presentation style applications. As SVG is an application of the W3C DOM, we can generalize the approach for other W3C or similar DOM based applications. Essentially all “office” (document oriented) applications can and perhaps should be developed with the W3C DOM. Thus our work applies straightforwardly to OpenOffice (StarOffice) and Microsoft Office suites and we believe its applicability is much broader. Further our

approach suggests that one need not develop special "collaborative" applications. Rather any application developed as an M-MVC style service can be made collaborative using the tools and architectural principles discussed in this paper.

Note that Moore's law implies that computer performance will continue to improve while networks will also continue to increase in bandwidth with however latency for long distance linkage remaining higher than that needed for interactive use. Thus inevitable infrastructure improvements will tend to make our approach more attractive in the future.

These ideas can also suggest a uniform approach to user interface design with desktop and web applications sharing a common portlet (WSRP, JSR168)-based architecture. This could motivate the development of new desktop applications with many capabilities not present in today's systems such as OpenOffice [OPENOFFICE] and Microsoft Office. The CGL research laboratory currently looking at extending these ideas to OpenOffice [Wang+Fox] while a limited implementation is possible using the rather crude event interface exposed for PowerPoint [WFP+04]. These ideas can unify PDA and desktop, as well as Linux, MacOS, Windows and PalmOS applications.

We've demonstrated a new way of building universal applications centered around explicit messaging. At meanwhile, we've also realized that this is just a start of a big topic with many issues need further study which could lead to important research results.

9.2 Answer to Initial Research Questions

Here we summarize answers to the questions given in the introduction of Chapter 1.

9.2.1 Can MVC be implemented in a message-based fashion?

We suggest the answer is yes and that a major contribution of this thesis is to demonstrate this as shown in Chapters 3-7. In fact we think the message based model MVC makes it clearer how to implement the control part which is essentially the negotiation between model and view embodied in the messages. Recent Web Service standards WSRP Web Services for Remote Portlets [WSRP] and WS-Management [WS-MANAGEMENT] provide interesting protocols to implement control. WSRP is specifically aimed at Web Service user facing ports while WS-Management is a general protocol for communication and negotiation between services.

9.2.2 What principles are there to govern the decomposition of a given application into MVC components?

This is discussed in detail in section 8.2 which is based on implementation described for SVG in chapter 6. We propose building applications as simple services whose size is as large enough to ensure acceptable communication but as small as is useful to achieve modular development with loosely coupled message-linked services.

9.2.3 What is the performance of the message-based MVC and what factors influence it?

The performance of M-MVC was studied in detail in chapter 7 and we found that the overheads of the explicit messaging was 20-40 ms for our modification of SVG and about 10-20% of time needed by SVG for typical action measured. We found in this study that thread scheduling generates major influence on this performance which had not been anticipated; we see operating system support of such messaging as an interesting future research area. The measured value will be reduced for cleaner decompositions as would

be possible for M-MVC applications built from scratch. Table 7.5 shows that the raw overhead of the messaging system was just 15 ms and either less CPU intensive or cleaner implementations would see our measured overhead reduced to this. In simpler scenarios in fact we measure NaradaBrokering overheads of 2 ms per hop or a total of 8 ms for the round trip from model to view by way of the broker. Our laboratory expects to develop a special in memory version of NaradaBrokering for use in Web Service containers and peer-to-peer interactions. That will substantially reduce overheads. Further continued improvement in CPU performance will reduce this overhead and in 3-5 years it will be essentially negligible on a desktop with a factor of 4-8 better performance. We performed one important optimization; namely we used "vector events" that transmitted multiple mouse move events that were generated every pixel traversed by the user's mouse. This type of optimization will always be important and can be extended to take account of other lightweight events such as mouse over. We emphasize that the current overhead is small compared to that generated by geographical distribution and so is already unimportant in collaborative applications.

9.2.4 How does M-MVC depend on the operating system, the application, machines and network?

We didn't investigate the operating system dependence of our model although standard NaradaBrokering benchmarks [NARADABROKERING][show better performance on Linux than Windows operating systems. Our SVG desktop application when decomposed was not sensitive to the network overhead over the university campus net with its high speed backbone as shown in the tables of chapter 7. The 100 ms or more typical network delay between institutions would be noticeable. As discussed above, we

identified features of the SVG application (clumsy decomposition, CPU intensive rendering) that increased the overhead. Cleaner simpler applications will less lower overheads although quite possibly a larger ratio of overhead to intrinsic rendering time.

9.2.5 What is the relationship of collaboration and Web services with MVC paradigm?

As we discussed in sections 9.2.1 and 9.2.2 above, we suggest that M-MVC provides a uniform service-oriented architecture that supports collaboration for Web services and desktop applications. Chapter 5 provides a detailed discussion of our SMMV and MMMV collaboration modes that capture the shared output port and shared input models of Web service collaboration.

9.2.6 What is the way to define state and state changes in collaborative applications?

Defining the state and identifying its changes is a critical and often very difficult part of the construction of collaborative applications. Fox noted [FOX03] this could be simplified for Web services by sharing the messages on input or output (user-facing) ports. We have extended this with the M-MVC concept by sharing the control and model-view interactions of an MVC application. We have discussed the structure of the state change events in section 4.4 and chapter 5.

9.2.7 How easy is it to convert an existing application to message-based MVC?

This is described in section 8.3 and was difficult for Batik due to the existence of large classes with many non serializable interfaces that cut cross many different parts of

the program. Several problems associated with subclassing and inheritance has been identified by some researchers. Lieberman asserts that inheritance is disadvantageous for highly interactive, incremental software development [H. Lieberman] while the smalltalk work [J. Bennett] also suggested it was not suitable for distributed applications. Our experiments with Batik support these conclusions.

9.2.8 What are the architectural and implementation principles to be used in building applications from scratch in a message-based MVC paradigm?

This is described in detail in Section 8.2 and involves breaking the application into simple service-oriented modules that can be efficiently implemented linked by messages. We explained there the principles to be used in decomposing the application.

9.3 Future Research

There are obviously many interesting deployment projects using the ideas in the thesis to build practical applications. More over, M-MVC provides fertile ground for important research issues that raised by our work. We give a few ideas below.

The performance data suggests the need to understand scheduling overheads and if these can or should be reduced. We only have limited data on the differences in operating system dependence and the comparison of Linux and Windows is a clear short term research project. Handheld devices with Symbian and Palm O/S need to be included. We should also study the uPortal and Jetspeed portals and see if their design and their associated standards (JSR-168, WSRP) need changes to support desktop applications.

This seems likely as the portals themselves are not themselves yet built according to the "simple service" principle. Some of these issues are being researched by Pierce's group in CGL which leads the OGCE (Open Grid Computing Environment) which is a major portlet research and development project [OGCE]. We suggest that this research could usefully address M-MVC in this architecture and see how standards like WSRP and WS-Management could be used to standardize control interfaces in MVC.

We introduced vector events to reduce message traffic and this needs further research; we also think a more powerful optimization could involve an in-memory operation mode for message systems like NaradaBrokering. A similar idea was originally proposed in CGL to allow NaradaBrokering to support directly Web Service containers like Apache Axis. This raises the general issues of how one builds "simple services" designed for both distributed and shared memory. In chapter 7, we also noted the possible importance of optimized transport with the use of UDP with application level fault tolerance and this is another research area. Chapter 6 emphasized the importance of application level queues and yet these are currently distinct from queues managed by NaradaBrokering. A careful study of the different ways of supporting multiple queues at different levels of the system could be fruitful.

Our performance results in chapter 7 highlighted the importance of the client performance. A useful short term project would look at this more systematically (extending the results exemplified by figure 7.4 on faster more modern desktops). It could be interesting to project M-MVC performance to the future by studying the differences we can see today with clients that span a 4 year vintage.

Our study of collaboration in chapter 5 emphasized the importance of a general message based protocol to support applications. CGL has proposed XGSP for this [WUBF] [WBUF] but only implemented a version for audio-video conferencing. It would be interesting to extend this work to general applications supporting the different modes SMMV and MMMV with different types of participants described in section 4.4.1.

Since Batik implements scalable vector graphics (SVG) [SVG] ─ an open standard for interactive graphics interface, such experience has general significance in helping us to understand of similar commercial tools such as Microsoft PowerPoint [POWERPOINT], Adobe Illustrator [ILLUSTRATOR] and PhotoShop [PHOTOSHOP], Corel Draw [CORELDRAW], and Macromedia Flash [FLASH], which has proprietary implementations. CGL has investigated extending these ideas to OpenOffice [Wang+Fox] while a partial implementation is possible using the limited event interface exposed by Microsoft for PowerPoint [WFP+04].

Appendix

Appendix A

Computer-based computing

The history of computing originated from ancient time, when human beings endeavor to stretch their knowledge and creativity from counting numbers. This effort ─ including the creation of device facilitated computing tools ─ has been continued through out the whole process of human civilization.

In 1945, the first electronic computer ENIAC [ENIAC1945] was built. It was originally intended for solving artillery calculation problem. But its general-purpose design demonstrated a new approach of building digital machines in supporting of large-scale computing at the time.

Over the decades, these machines have emerged as an imposing force that affect our society in every aspects of civilization including science, social science, commerce, the arts, and the humanities. The automation of our lives has involved subtle alterations in knowledge acquisition and information dissemination. As a brief and cursory introduction, these features are shown in a clear trace of the transition for generations of computer technology ─ from small computer systems, large-scale processors integration and supercomputers, to connect geographically dispersed computing powers into a global information infrastructure, the Internet.

Small computer systems started with the revolution of primitive general purpose programmable electronic computer with advancement including improvement of I/O devices and semiconductor technologies; development of programming languages such

as FORTRAN [FORTRAN1956], ALGOL [ALGOL1958], COBOL [COBOL1959], and C [C1972]; introduction to time-sharing operating system like UNIX [UNIX].

Supercomputers targeted for high performance computing and communication, which is dominated by massive parallel systems with development from pipelining, vector processing, which are limited to use of single-user workstations, to "Hypercube" [HYPERCUBE] with distributed memory architecture that aggregated 128 processors at the time. Derivative parallel algorithms and programming techniques includes widely used SMMV [SMMV] and MMMV [MMMV] programming model, and message passing interface (MPI) [MPI] as communication mechanism between processors.

Development of the Internet begins with tremendous gains from the existing computing powers by providing availability and pervasive accessibility to these distributed resources, which form highlighted spots in the interwoven and linked global information infrastructure. The effort has also inspired spawning of distributed computing technologies and Web technologies which including the development of object-oriented style programming with representative language such as C++ [C++] and Java [JAVA]; software development environment including CORBA [CORBA], COM [COM], J2EE [J2EE], and .NET [DOTNET]; structured data descriptive language XML [XML]; dynamic connection to database through JDBC [JDBC] and ODBC [ODBC].

Appendix B

Internet and Web applications

Computer network, distributed system, Internet [INTERNET 1969], Net and World-Wide Web (WWW or Web) [WWW 1991] all refer to inter-connected computer systems,

with focus from different aspects. As one of the greatest successful examples of technology advancing human civilization, Internet started with research innovations of computer network technologies and has transformed and revolutionized as a global information infrastructure. Its influence goes beyond technology sector and evolves into a unique, complex, diversified, and synergistic world-wide system combined with technological, organizational, and community involvements. With the increasing of online population, we expect Internet and Web applications to generate more-than-ever impact in depth and broadness on our society in information acquisition, community operations and electronic commerce. An ever-ongoing effort of world communication will, in turn, motivate spawning of new innovative Web technologies. To get a whole picture of the evolution of Web applications, we first trace back to the beginning of Internet.

In December 1969, Internet was born ─ ARPANET [ARPANET 1969] connected four host computers together through Interface Message Processors (IMP) and formed the first true computer network, as shown in fig. B.1(a). Although fledging, it allows computers from four university campuses in the U.S. to "talk" to each other. In these times, one starts off sending request of exchanging data by typing commands from a computer keyboard. The data is fragmented into "packages", delivered to the destination host over high speed telephone line, and then reassembled in original order at arrival. Email (or electronic mail) was an initial "hot" application that developed on top of ARPANET (see fig. B.1(b)), which provided early person-to-person communication by sending messages over the Internet.

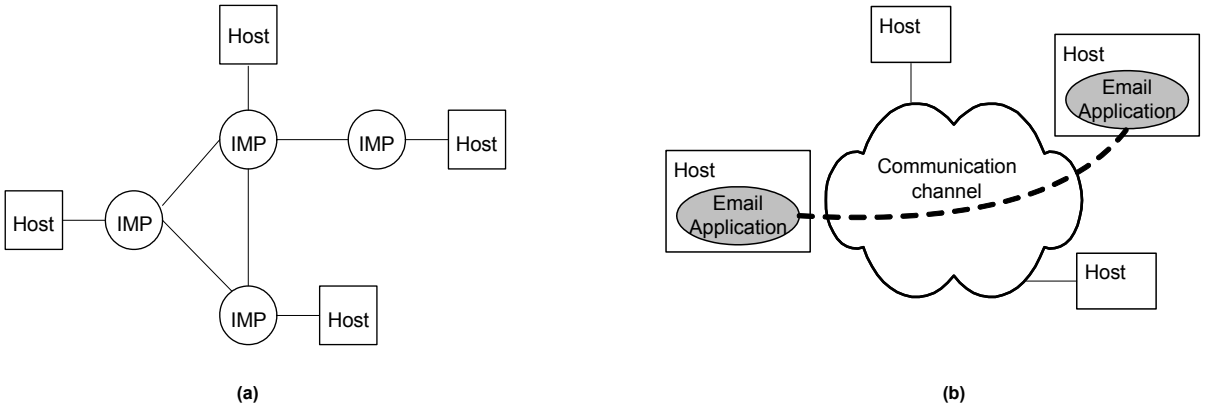


Figure B.1 ARPANET and its application: (a) ARPANET's four node network; (b) Email application on ARPANET over an abstract communication channel

Appendix C

Network Infrastructure

Layered network

A typical layered network system can be illustrated in a stack as in Figure 2.2. The lower level is more close to computer and telecommunication hardware; the upper level is application software for user interaction; in between them is network infrastructure. Open Systems Interconnection (OSI) [OSI] reference model and Internet are two widely used network architecture. The former is considered the primary logical architectural model for inter-computer communications with standard seven-layer architecture; the latter, which comes from experience of building ARPANET, bundles several functional units into one layer and forms a four-layer model instead. As a general principal, both are designed for connectivity among a large number of computers with high-performance, low-cost, robustness and expandable linkage. The central idea is a layered approach to modularity design ─ decomposition of the system into components/objects stack from low to high level.

As a contrast, fig. C.1 depicts a graph that reflects the general design scheme of network system and fig. C.2 shows a specific example of Internet application ─ email. There are two types of communication interfaces: within local system, each lower level object provides *service interface* to adjacent higher level object; between counterparts (or peers) of two systems, a *peer interface* of communication service is defined. The abstraction of object (or layer) and its two service interfaces are referred as *protocol*. For the case, starting with user input of email message on host computer 1, Process-to-Process Channel Protocol (PPCP) wraps email data and forwards it down to Host-to-Host Protocol (HHP). After adding its own information to the header of the datagram, HHP forwards it further into the physical network. On the destination host, opposite operations are executed and original email data is recovered in application interface with attached protocol information removed along each bypassed level.

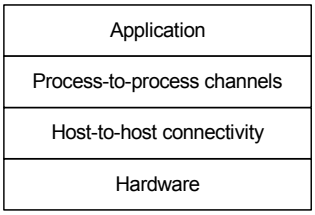


Figure C.1 Network system in layered stack

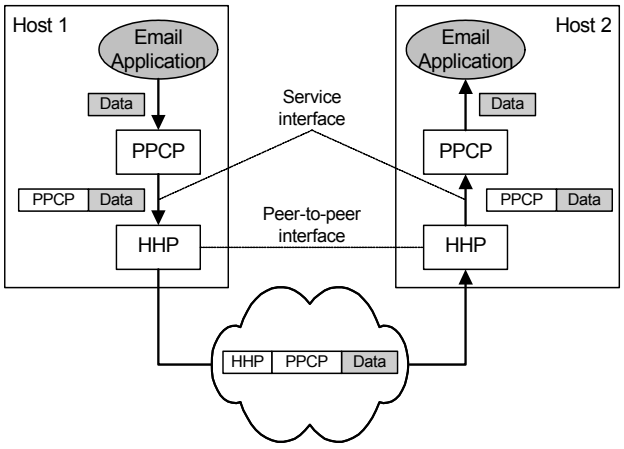


Figure C.2 Email application on ARPANET over an abstract communication channel

While there're many ways of decomposition on a system, layering abstraction has general prominent advantages ─ it reduces the complexity of network topology in two dimensions: it forms manageable modules in vertical thus avoids spaghetti code; it

confines protocol specification in horizon within each peer level with clear interpretation, which supports upper-layer protocol transparency ─ encapsulation (high-level message wrapped inside low-level message) and delivery of the high-level message only operate at the same layer without acquisition of knowing its details. Modularized design facilitates reuse and replacement of existing program components, which helps system maintenance. Moreover, layering provides a service model that allows flexible bundle and expansion of layers (or components) in an arbitrarily complex network system. In section 3.6, we elaborate utilizing the expandable feature of layered service in our design of M-MVC for Web applications, which has separate application level architecture from underlying messaging infrastructure. Here, we describe an example of bundling multiple functions into one layer in a case of TCP/IP protocol, which is the distinction of Internet architecture from that of OSI model.

TCP/IP protocol

Around middle 70's, TCP/IP protocol [TCP/IP 1974], short for Transmission control protocol/internet protocol, was developed as a standard of communication between different networks in a common language. IP defines how to route data frames or packages from host to host and supports multiple networks interconnected into a single, logical network; TCP, built on top of IP, provides detection and correction of erroneous data during transmission. In 1984, Domain Name System (DNS) [DNS 1984] set up a tier structure that marks every online host computer with a unique IP address for access identification. Internet was originally limited to the access of large research organizations. Apace the development of micro-computers, TCP/IP was modified to support for personal computers which typically connected through unreliable telephone lines. The

accommodating changes helped it being adopted as major Internet protocol by national governments around middle 80's. TCP and UDP [UDP 1980], an unreliable datagram delivery protocol proposed in 1980, form the so-called *end-to-end protocol* that provides service to variety of application protocols. Fig. C.3 shows the mapping of TCP in Internet and Transport layer in OSI [OSI]; both are equivalent in function to *Process-to-Process Channel* in the stack of a typical layered network system in fig. C.1.

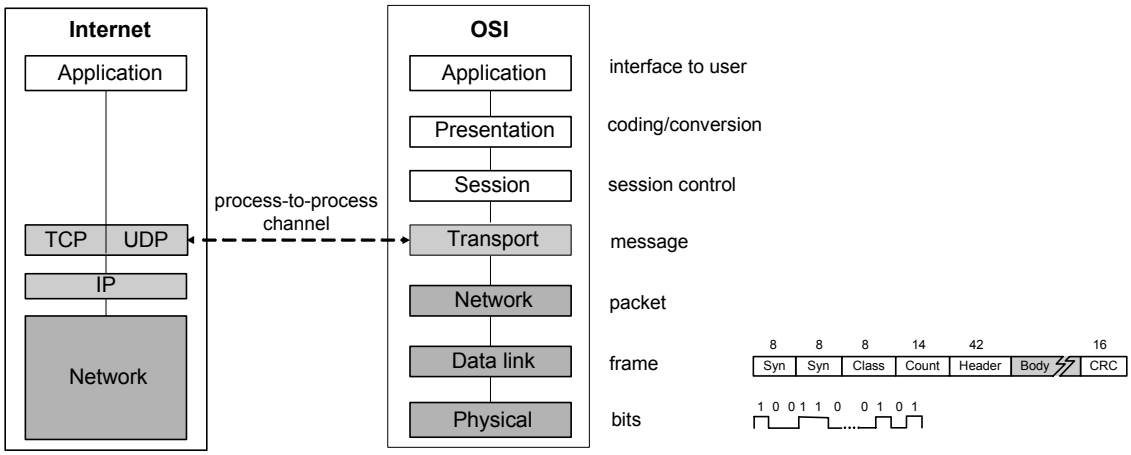


Figure C.3 Internet versus OSI Architecture

Internet — an internetwork of networks in an open architecture

It worth mentioning that between mid 70 and early 90's, many innovative technologies advanced computer and computer network, and generated great impact on the rapid growth of Internet.

- Main stream computers running in university mostly use UNIX [UNIX] operating system. Telnet [TELNET 1972], a remote connection service for controlling a computer; and FTP [FTP 1973], file transfer protocol that allows data being sent in chunks between computers provide application tools for network access. UNIX-to-UNIX protocol [UUCP 1976] opened up networking to the broader

academic community, on top of which Usenet newsgroup [USENET 1979] supplies a system that archives news "articles" in hierachy and exchanges them between writers and readers.

- Parallel computing highlighted supercomputer by advancing computer in capacities and computation speeds, and stimulated the expansion of computer network in serving for high performance computing. In 1986, NSF establishes 5 super-computing centers to provide high-computing power for all This allows an explosion of connections, especially from universities.
- Personal computer [PC 1981] was introduced by IBM in 1981. Incorporating personal computers into Internet encourages expansion of online population of individual users. The potential market attracts entering of commercial exploitation of Internet from 90’s.

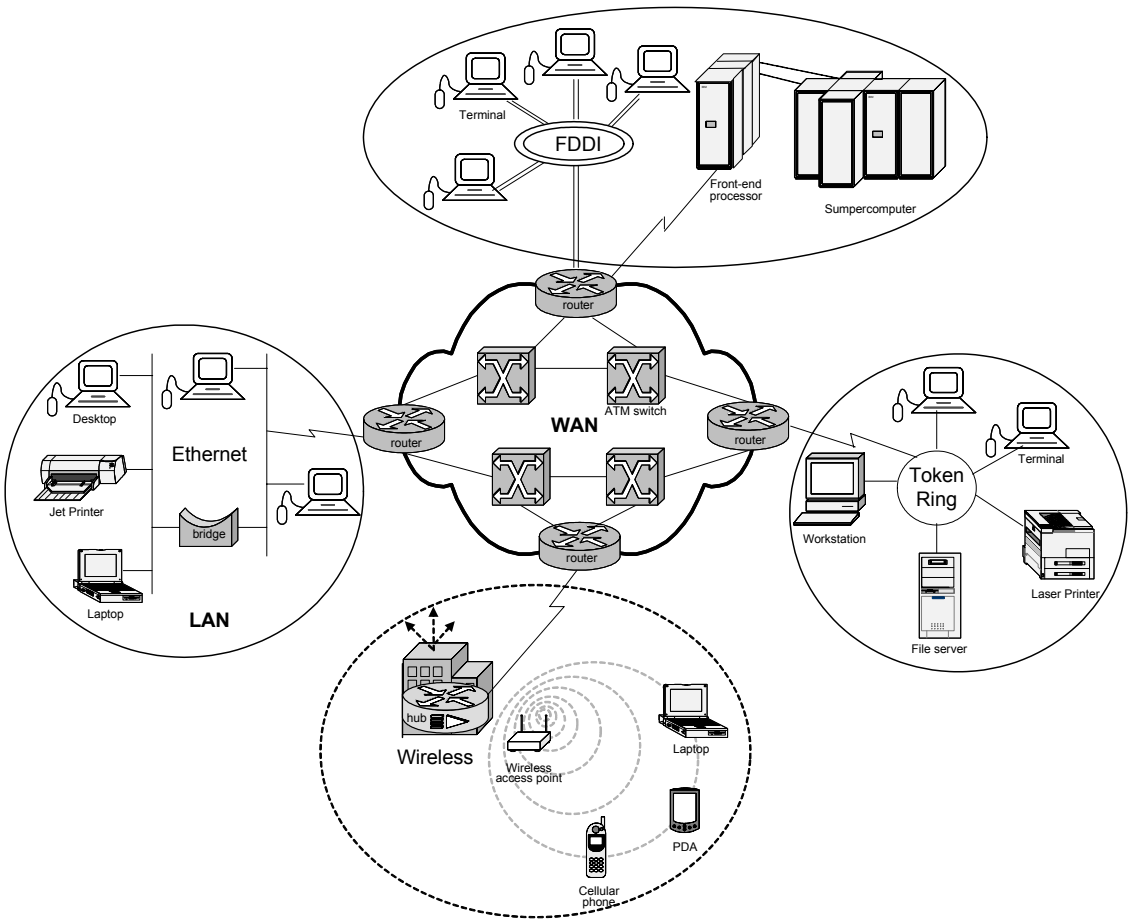


Figure C.4 Internet topology as network of networks

Internet represents a design concept of openness. It implies an open architecture ─ a Wide Area Network (WAN) [WAN] freely available to interconnected autonomous Local Area Network (LAN) [LAN]. The sub-networks and their linkage packages, which routes through routers, are equally treated. This concept greatly facilitated subsequent advance of network technologies. Today, Internet has evolved as a ubiquitous network accessible from heterogeneous subnets building on wired (e.g. Ethernet [ETHERNET], FDDI [FDDI], and Token-ring [TOKENRING]) and wireless [WIRELESS 1997] deployment at *Data link* and *Physical* layers. The topology, which is enriched with diverse subsystems (see fig. C.4), provides a powerful global infrastructure that facilitates development of

growingly sophisticated information services for academia, organization, business and community.

Appendix D

Overview of Web Application Architecture

World-Wide Web (WWW) [WWW 1991] marks a big leap forward from Internet to Web. WWW is developed by Tim Berners-Lee and scientists at CERN (Geneva), the European centre for High Energy Physics. Lee made clear comments on Internet and Web [TIM B.L.]:

> "*The Web is an abstract (imaginary) space of information. On the Net, you find computers — on the Web, you find document, sounds, videos, ... information. On the Net, the connections are cables between computers; on the Web, connections are hypertext links. The Web exists because of programs which communicate between computers on the Net. The Web could not be without the Net. The Web made the net useful because people are really interested in information (not to mention knowledge and wisdom!) and don't really want to have know about computers and cables.*"

WWW was originally implemented using a non-GUI browser for retrieving and viewing documents from Internet in a multi-platform environment. It was based on three key technologies: Hypertext Transfer Protocol (HTTP) [HTTP] which provides a mechanism of accessing online information as file document; Hypertext Markup Language (HTML) [HTML] that defines document format in text tags; an important concept of building consistent client user interface so that users could access information

from many types of computers. Mosaic [MOSAIC], developed at National Center for Supercomputing Applications (NCSA), was the first browser with Graphical User Interface (GUI). Its commercialized version, Netscape browser, became immediately popular as it frees personal computer users from the narrowed desktop to an open cyberspace by navigating the Web with easy point and click. Basic structure of World Wide Web as a Web application is illustrated in fig. D.1. By clicking on a hyperlink, a client can request for a Web page that is indicated by URL [URL] from a remote server through HTTP in a request and response cycle.

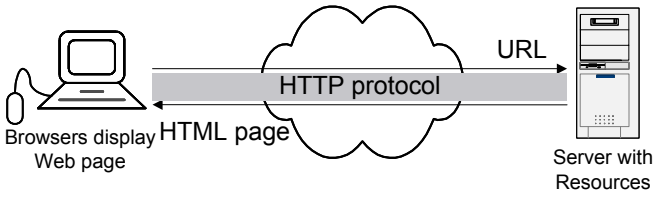


Figure D.1 Basic structure of World Wide Web

Web applications are typically built on top of TCP/IP. In early days, client-server architecture is widely used in Internet applications including FTP, Email, NFS [NSF1989] and variety of Web browsers using HTTP protocol. Apace Web applications becoming more sophisticated, the exploration of their architectures and technologies reflects an in depth deployment approach from the intrinsic views of Internet and Web. Meanwhile, this approach is greatly influenced by the advance of ad hoc technologies including computer hardware, telecommunication and software engineering. Our discussion of the evolution implies two themes: one is the timeline; the other is different perspective of views of Web applications. Based on architectural topology, they can be classified into centralized versus decentralized system, which further evolve into three categories: client/server, multi-tier, and peer-to-peer (P2P) [P2P]. With respect to forming a service

model over existing Internet infrastructure, there are grids [GRIDS] and overlay network [OVERLAYNETWORK]. Both of them are originally designed to provide middleware services for distributed system: the former aims to support massive computational applications; the latter offers a software routing mechanism that simulates multicast over IP at *transportation* level, which effectively supports P2P applications such as video/audio conferencing. JXTA [JXTA] is an example of P2P overlay network. From the point of view of interoperable relationship between distributed components, Web applications are built with Web services [WEBSERVICE] that communicate through SOAP [SOAP] with messages typically in XML [XML] format.

Client/server model

In a two-tier client-server model, the first tier is client side interface; the second tier is the Web server, which is usually associated with file system or database. Web browsers such as Internet Explorer (IE) [IE] or Netscape Navigator (Netscape) [NETSCAPE] are dominant client side interfaces to Internet. Technologies such as Java [JAVA 1995], JavaScript [JAVASCRIPT 1995] and ActiveX [ACTIVEX 1995] enhances browser environment by bringing dynamic interaction with embedded code that runs in HTML browser. Through Common Gateway Interface (CGI) [CGI], tier two server interfaces with external applications. CGI scripts, including Perl [PERL], C/C++, Tcl [TCL] and VisualBasic [VB], are typical server side technologies that provide interactions with many different systems like graphics renderer and database. Client/server model, as request/response architecture, is suitable for applications of single service supporting multiple clients.

Multi-tier model

To accommodate the need of scalability for large-scale applications with complex services that support potentially hundreds of thousands of users, a more sophisticated approach is adopted as multi-tier architecture. In the case of common three-tier application, a system is explicitly separated into functional modules including client interface, broker/server and file system (or database storage). Apart from benefits of modularized software engineering as a general principle, this approach is perceived to allow a lasting architecture while detailed technologies for individual component change over the time. For instance, in building customizable applications, the client interface used to run on Windows and UNIX may need to upgrade to Linux operating system. Such modification will only affect the first tier without causing changes to full life cycle of the software development. A comparison of architectures of two, three, and four tier model is depicted in fig. D.2. Servlet [SERVLET] and Java Server Pages (JSP) [JSP] eventually usurped CGI as the prevalent server side technologies. JDBC [JDBC] provides a bridge between client and backend database, which allows SQL [SQL] queries to be issued at middle tier.

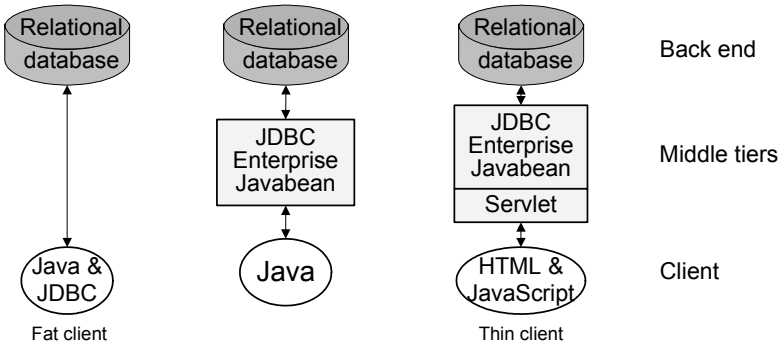


Figure D.2 Comparison of 2, 3 and 4 tier model

Middleware layer

The concept of "middleware" has emerged and plays an important role in the development of large-scale distributed applications such as business application deployment. Taking the view that Internet is consists of distributed objects; a middleware system comprises a layer in between application and network infrastructure (or operating system). The platform supplies a rich set of services to distributed system by fostering application portability and interoperability: CORBA [CORBA], DCOM [DCOM], DCE [DCE], Java RMI [RMI] forms the RPC/RMI-systems; transaction processing systems forms the business logic, MQSeries [MQSeries] and MSMQ [MSMQ] provides message queuing service; LDAP [LDAP] offers directory access protocols, ODBC [ODBC], JDBC and mediators supplies database access and integration; COM+ [COM+], CORBA, Enterprise Java Beans (EJB) [EJB] provides component models. Middleware technologies keep evolving ─ nowadays, J2EE [J2EE] plus XML solution and .NET [DOTNET] already replace CORBA and DCOM as new generation platforms for building Web applications.

Peer-to-Peer model

The classification of two-tier and multi-tier architecture is originated from two dimensional layering view of a system between interface and service, with evolution from one-to-one to multi-to-multi model. However, both two-tier and multi-tier architecture defines fixed clients and servers with server provide centralized service. As a contrast, Peer-to-peer (P2P) [P2P] architecture has emerged as a self-organizing decentralized network system with a dynamic and adaptive paradigm ─ it does not distinguish client and server in a static way; peer nodes (or "servents") hold equal

position to other peers and dynamically join or leave Internet on the edge. In real implementation, P2P application could be fully distributed or semi-centralized. File sharing systems such as Gnutella [GNUTELLA 2000], Napster [NAPSTER], and Freenet [FREENET] are examples of the above two types. P2P application commonly provides locater function using controlled-flooding mechanisms, where a query is forwarded from a node to its neighbor's recursively. The query would end up either with the querying node receiving a reply or its propagation stopping at end of Time-To-Live (TTL). P2P presents a stabilized model, which unlike client/server architecture, does not sensitive to individual server's availability. However, flooding-based system does not scale well. It is mainly because flooding looking up mechanism is build on top of stateless multicast over IP and it imposes communication overheads.

Web Services

WWW marks the beginning of the Web by using HTTP to transport HTML document across the Internet while Web Services are poised for a new generation of Web by providing development environment that enables highly dynamic program-to-program interaction [CKB]. Compared with conventional monolithic systems, Web Services extend the idea of traditional Web application servers to make an application framework, which have fundamental characteristics embracing dynamic bound components and loosely coupling message linkage that maximize the flexibility, interoperability, and reusability of software assets. As a platform neutral and programming language independent framework of building distributed systems, it aims to address the issues with existing distributed computing models such as CORBA, JAVA RMI, distributed Smalltalk contains [GOTTSCHALK]. The limitations of these systems are mainly due to

tight coupling of system components, which reduces the possibility of cross-platform interoperability and just-in-time integration of services.

Service oriented architecture (SOA) [SOA] is a new generation of web-based distributed application infrastructure. The impetus of adopting SOA is the realization that interoperability and loose coupling are essential features that can greatly simplified the tasks of building, integrating and extending distributed systems such as enterprise applications.

Web Services [WEBSERVICE] provide universal APIs that support the general framework and are becoming an increasingly important feature of Internet and Grid systems. They support a loosely coupled service oriented architecture that builds on previous distributed object architectures like CORBA, Java RMI, and DCOM to provide dynamic, scalable, and interoperable systems. The broad applicability of this approach includes enterprise software, e-Science and e-Business. Correspondingly there are a growing number of powerful tools that are available for building, maintaining and accessing Web Service-based systems. These tools include portals that allow user front-ends to Web Services. This model for user interaction has new standards like portlets with WSRP (Web Services for Remote Portlets) [WSRP] and the Java Specification Request JSR168 [JSR168] supporting lightweight interfaces to the backend resources.

There are variants of Web Services architecture depending on the details of Web Services stack from different organizations [MYERSON]. Examples include IBM [WSCA] [WSFL], Microsoft [CKB], W3C [BOOTH], Sun, BEA, Hewlett-Packard, Oracle, and ebXML [EBXML]. The Web Services architecture defines the relationship

between *Service Provider*, *Service Requester*, and *Service Broker*, and how to "engage" a Web Service between the requester and the provider via intermediate broker by *Publish*, *Find*, and *Bind* operations using WSDL [WSDL] and UDDI [UDDI] protocols.

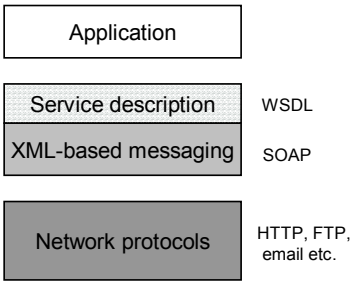


Figure D.3 A Web service stack

A Web service stack is depicted in fig. D.3. It shows that Web Services interact by exchanging of messages in SOAP format while WSDL is used for describing a service as contract. According to IBM Web Services Conceptual Architecture [WSCA], XML messaging layer is "the most fundamental underpinnings" of the Web Services architecture that provides network accessibility of services. Implementations of Web Services include open source project Axis [AXIS] from Apache.

Appendix E

DOM

Definition of *Node* interface in Independent Definition Language (IDL) as specified in Document Object Model (DOM) Core Level 1 Specification by W3C.

IDL Definition

```
interface Node {
  // NodeType
  const unsigned short   ELEMENT_NODE                = 1;
  const unsigned short   ATTRIBUTE_NODE              = 2;
  const unsigned short   TEXT_NODE                   = 3;
  const unsigned short   CDATA_SECTION_NODE          = 4;
  const unsigned short   ENTITY_REFERENCE_NODE = 5;
  const unsigned short   ENTITY_NODE                 = 6;
  const unsigned short   PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short   COMMENT_NODE                = 8;
  const unsigned short   DOCUMENT_NODE               = 9;
  const unsigned short   DOCUMENT_TYPE_NODE     = 10;
  const unsigned short   DOCUMENT_FRAGMENT_NODE = 11;
  const unsigned short   NOTATION_NODE               = 12;

  readonly attribute DOMString          nodeName;
           attribute DOMString          nodeValue;
                                           // raises(DOMException) on setting
                                           // raises(DOMException) on retrieval
  readonly attribute unsigned short     nodeType;
  readonly attribute Node               parentNode;
  readonly attribute NodeList           childNodes;
  readonly attribute Node               firstChild;
  readonly attribute Node               lastChild;
  readonly attribute Node               previousSibling;
  readonly attribute Node               nextSibling;
  readonly attribute NamedNodeMap       attributes;
  readonly attribute Document           ownerDocument;
  Node                  insertBefore(in Node newChild,
                                     in Node refChild)
                                     raises(DOMException);
  Node                  replaceChild(in Node newChild,
                                     in Node oldChild)
                                     raises(DOMException);
  Node                  removeChild(in Node oldChild)
                                     raises(DOMException);
  Node                  appendChild(in Node newChild)
                                     raises(DOMException);
  boolean               hasChildNodes();
  Node                  cloneNode(in boolean deep);
};
```

Figure E.1 IDL definition of Node Interface

Appendix F

Overview of SVG

F.1 Two types of computer graphics

There are two types of computer graphics: raster and vector. The raster graphics is made up of pixels. The vector graphics is composed of paths.

Raster images are commonly referred as bitmap images. Examples include JPEG (Joint Photographic Experts Group), GIF (Graphics Interchange Format), TIFF (Tagged-Image File Format), PNG (Portable Network Graphics) and PICT (Macintosh graphics) format. Since a bitmap uses an array of pixels with values indicating the color or shade,

bitmap images have a fixed resolution and cannot be resized without losing image quality. They tend to have much large file sizes than vector graphics because of pixel-based storage. Individual raster technology often employs different compression approaches to reduce the file size for storage and fast download.

Vector graphics use mathematical relationships between points and the paths connecting them to describe an image. Vector formats can also integrate raster images thereby represent a combination of lines, curves, and bitmaps. Common vector formats include AI (Adobe Illustrator), CDR (CorelDraw), CGM (Computer Graphics Metafile), SWF (Shockwave Flash), DXF (AutoCAD and other CAD software), and SVG (Scalable Vector Graphics). Typically Vector-based images are not translated into bitmaps until the last possible moment. Therefore they have much more flexibility for transformation (e.g. scaling) while keep high resolution. In addition, vector images describe graphics object at metadata level (e.g. text-based description of properties), which tend to have much smaller file sizes than raster-based bitmaps for storage and transmission.

In general, bitmaps are suitable for photographs and images with subtle shading whilst vector graphics are best used for page layout, interaction and distribution of applications. However, most modern output devices, including computer monitors, dot-matrix printers, and laser printers, are raster devices while almost all sophisticated graphics systems, including CAD systems and animation software, use vector graphics. The reconciliation of the difference between raster graphics and vector graphics comes down to when rasterizing occurs during the process of implementation.

In fact, vector font technology has been viewed as an important feature of operating system. Many printers (e.g. PostScript printers) have been used vector graphics with

internal converter to raster images. With all the merits of vector graphics, future software applications would primarily use vector-based technology to create, import, display and print graphics objects alongside with bitmaps.

F.2 SVG and the thesis project

The impetus for the thesis project occurred in spring 2000, when our research group conducted a survey of new technologies for building Web based applications for education. At that time, Macromedia Flash [FLASH] had become a popular toolkit for Web designers ─ its fascinating movie like web interface immediately attracted our attention since developers already got used to common HTML web pages. Flash provides a fresh vector-based authoring platform with graphical user interface (GUI) that allows one to easily manipulate document layout, mix sound and animation, ultimately generate an interactive multimedia movie as web content. Then it can be exported to a SWF file, the binary format of Flash content, and played by Flash Player plug-in in Internet Explorer (IE) [IE] or Netscape Navigator (Netscape) [NETSCAPE] web browser.

We gained initial experience by building a research presentation web site [QIU-10-2000] using Flash 5.0 [FLASH5] as the snapshot displayed in fig. F.1; and combined with Macromedia Generator 2.0 [GENERATOR2], presenting a simple example of dynamic delivery of contents provided by a text source file. The former demonstrates how a stunning animated internet would look like with rich media web content such as vector graphics (with ActionScript [ACTIONSCRIPT] interaction); the latter shows an integrated data driven authoring case with potential supporting architecture that consists of client, server and database ─ a typical three-tier web application model of the time.

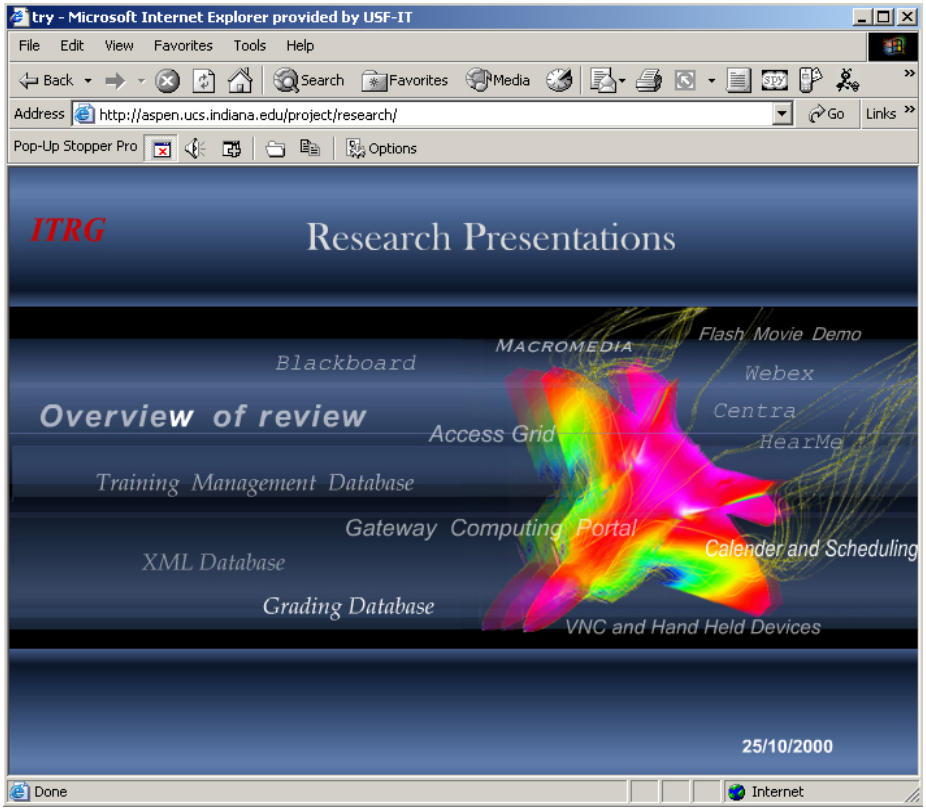


Figure F.1 Research presentation web site using Macromedia Flash

Macromedia was not the first one providing vector-based animation authoring tools. However, it uses vectors to pack graphics and animation objects, adding a bit of sound, and creatively coded to use SWF format for delivery of the rich media over to Web. However, with Flash and Generator integration, it supplied an all-in-one package. This model seemed quite successful in offering a platform that reconciled the gap between artistic vector graphics designing and graphical content authoring in web browsers, which appeared to split users into artist and programmer group.

At the mean time, we noticed that W3C proposed a series of multimedia related specification in 2000 including Scalable Vector Graphics (SVG) 1.0 specification [SVG] as a candidate recommendation and Synchronized Multimedia Integration Language (SMIL) 2.0 specification [SMIL2.0] as a working draft. After substantial investigation

and assessment of the core technologies, a technical report about Flash 5.0 and Generator 2.0 was completed [QIU-01-2001] in January of year 2001. As a conclusion, we found that SVG provides a better programmable environment than Flash for our research purpose. In contrast to other vector graphics technologies such as Macromedia Flash, the most attractive features of SVG are its openness and compatibility of existing standards, which can be further outlined in following aspects:

- open standard versus proprietary product;
- XML file format versus binary SWF format;
- conformation to DOM (core architecture and event model) versus internal data structure and event handling;
- JavaScript versus ActionScript programming language for scripting and animation applications.

Although as a technology, SVG hasn't yet come up with a Flash like authoring platform, it has been successfully deployed in various application areas. SVG is well accepted by the geography community in developing Geographical Information System (GIS) (e.g. interactive mapping system) [BFF2003] [ADCOCK2004]. Its vector graphics feature is especially suitable for zooming to fit various display devices including small screen such as PDA and mobile phone [KALLIO] [ZM2004]. To meet industry demands, W3C has proposed two mobile profiles [SVGMOBILE]: SVGTiny (SVGT) and SVG Basic (SVGB) that entail SVG specification 1.0 with special customization for mobile devices based on constraints of low memory, CPU, and small display settings. SVG-based commercial or non-commercial tools provide viewing (Adobe SVG viewer

[ADOBESVG], Corel SVG Viewer [CORELSVG]), editing, presentation, data visualization, conversion capabilities [SVGIMPLEMENTATIONS].

In retrospect, we've made a wise decision in carrying out our investigation and experiments based on open standard SVG [SVG] and open source implementation ─ Batik SVG viewer project [BATIK1.5]. The reason is two-folds. New generation of Web applications are composed of rich media content and client interface. SVG contains technical and implementation details that allow us to have an in depth/complete evaluation of the tactics of graphical rendering and the mechanism of event-based interactive relationship between the visual constituent and its data structure. Further more/More over, as a standalone client application, Batik SVG provides an ideal case for our experiments to analysis different architectural principles, which facilitates the investigation of a uniform software design strategy for distributed applications that bridge the gap so as to maximally leverage existing components and incorporate collaboration capability.

Based on our progress with SVG experiments, in 2002, the original effort of building an advanced Web-based education system supporting rich media content ─ collaborative application with SVG ─ has been extended and emerged the idea for designing a generic architecture of message-based Web applications that justified for emerging loosely coupled Web Services oriented software model. Message-based MVC (or M-MVC) was developed as a paradigm of messaging linkage service model converging desktop application and Web application with automatic collaboration advantage. In subsequent sections and chapters, we elaborate how exploration of SVG along its multiple

dimensions service as a coherent approach to the M-MVC solution that is adaptable for emerging trends of the Internet and Web evolution.

F.3 Essential features of SVG

SVG is the acronym of Scalable Vector Graphics. W3C defines SVG as "*a language for describing two-dimensional graphics and graphical applications in XML*" in specification version 1.0 [SVG]. Compared with HTML content, SVG has richer Web graphics flavor, which can be viewed as a specification for "graphics of HTML". The feature of an open standard for interactive vector graphics as well as those inherited from XML and DOM makes SVG a unique technology for Web applications.

Vector graphics feature

As a rich graphical content, SVG includes three types of graphical objects (vector shape, text and image) that can be nested, grouped, transformed and styled, in addition to graphical processing (clipping, masking and filtering). SVG content can be dynamically updated (zoom, rotate and translate) without loss of rendering resolution.

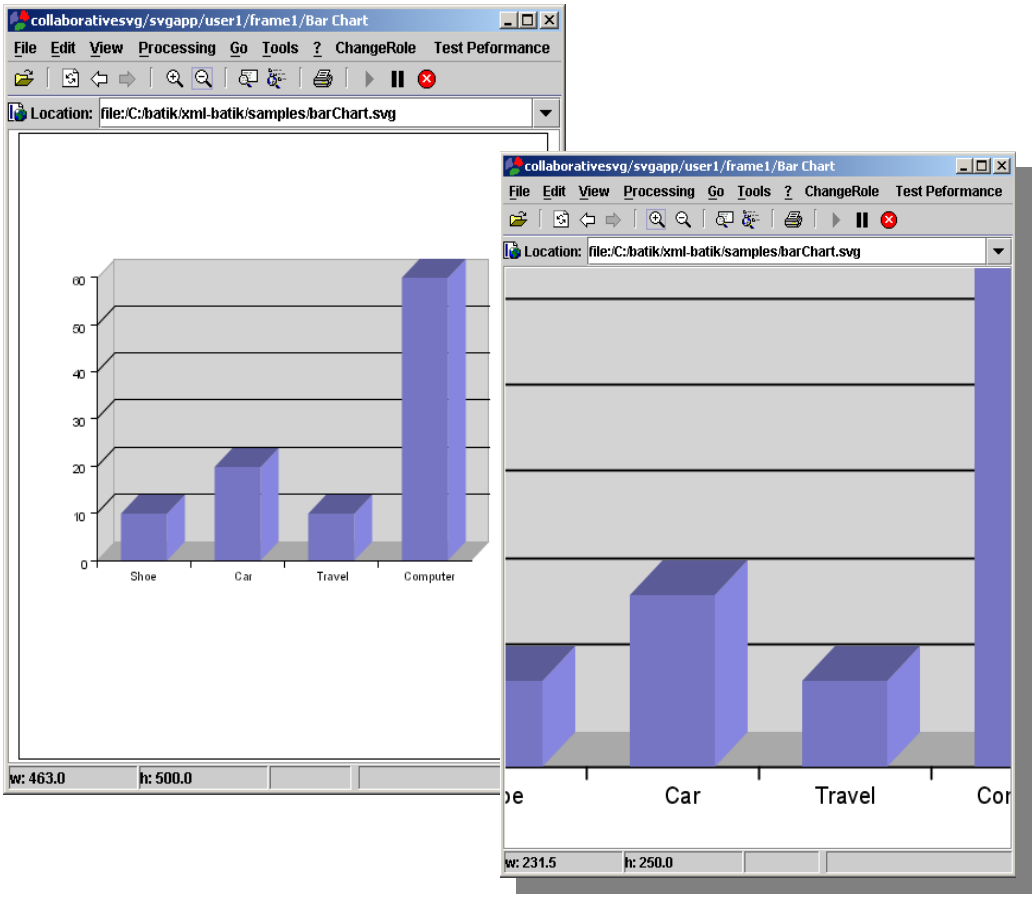


Figure F.2 Scaling of SVG document

Anybody who has experienced the frustration of trying to zoom/enlarge a bitmap image but it turned out to be/came up with an unrecognizable rough and smoggy picture would probably agree that scaling is one of the most desirable vector graphics features that SVG can bring to the table. As shown in fig. F.2, a few snapshots captured the rendering of a Batik SVG example barChat.svg in original size as a comparison to its 2x enlarged size. The merit is that high quality visual results are always guaranteed no matter at what scale level. This is an important feature for interactive systems. As often one needs to magnify a picture (e.g. a photo or map) for a closer look or detailed information. Likewise, it helps to zoom out a graph for a panorama view on a small hand held device so as to keep the same visual feel as on regular PC display. In this sense,

SVG provides a vector graphics technology that enriches the solutions designed for support of ubiquity and hypermedia for next generation of client interface.

XML feature

SVG is an open standard that defines graphical objects in XML [XML]. Because XML has been widely used for structured information exchange, SVG gains many advantages such as scalable, accessible, structured, and rich format by building on the existing standard. Through supporting of XLINK [XLINK] and SVG view specification (or XPointer [XPOINTER]), SVG provides an effective way for referencing both remote external document object (e.g. the ' href ' attribute on the 'a' element) and internal document fragment (specified on attribute of 'view' or 'g' element). Apart from being used as a standalone graphical content, SVG is also a usable XML namespace that is accessible from other namespace document (e.g. XTHML) or third party application like JDBC. It is an attractive portable intermediate format for exporting (e.g. from Illustrator and PowerPoint); transcoding between vector graphics (e.g. pdf and PowerPoint) and from vector to rasterized graphics (e.g. PNG).

For simplicity and visual convenience, we use a rectangle graphical object to illustrate SVG composition. In the canvas area of Batik SVG viewer, one blue rectangle (with two frame borders to indicate positions) is displayed within “Position A” frame border in fig F.3.

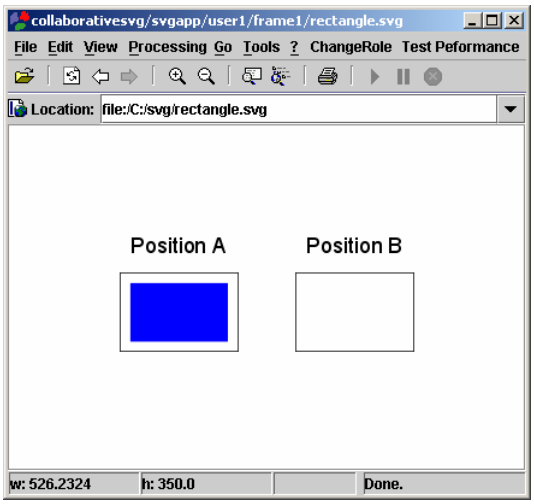


Figure F.3 Graphical representation of rectangle.svg document

The visual effect of the above example would be equivalent to the following XML representation (see fig. F.4), where SVG content is embedded within a well-formed and valid XML document. In line 1, XML declaration indicates version 1.0 of XML is being used. In line 2, the document type declaration (DTD) imposes constraints on logic structure of containing SVG elements via an external specification “svg-20000802.dtd”. The outmost 'svg' tag in line 4 marks the beginning of root SVG element with an ending tag in line 28. A total of five SVG elements are included: five rectangles and two texts, which are highlighted by arrow markers. The grouping element 'g' (with “content” identification) provides a container for three embedded subsets: grouping element “PositionA” and “PositionB”, as well as 'rect' shape element. Each of the grouping position elements is composed of 'text' and nested border group containing black 'rect' outline and 'rect' white fill.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
"http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd">

<svg id="body" width="300" height="350" viewBox="0 0 300 350">

   <g id="content" transform="translate(0,120)">
      <g id="PositionA" transform="translate(0,0)">
         <text class="legend" x="10" y="10" style="font-family: Arial; font-size:22">Position A</text>

         <g id="borderA">
            <rect id="originalPlaceShadowBorder" x="0" y="30" width="120" height="80" style="fill:none; stroke:black" />
            <rect id="origianlInterior" x="1" y="31" width="118" height="78" style="fill:white; stroke:none" />
         </g>
      </g>

      <g id="PositionB" transform="translate(180,0)">
         <text class="legend" x="10" y="10" style="font-family: Arial; font-size:22">Position B</text>

         <g id="borderB">
            <rect id="targetPlaceShadowBorder" x="0" y="30" width="120" height="80" style="fill:none; stroke:black" />
            <rect id="targetInterior" x="1" y="31" width="118" height="78" style="fill:white; stroke:none" />
         </g>
      </g>

      <rect id="targetRect" x="10" y="40" width="100" height="60" style="fill:blue;" />
   </g>

</svg>
```

Figure F.4 A SVG file rectangle.svg in XML format

As shown by the simple example, SVG entails powerful structuring capabilities of document by using XML in describing graphical object. Internally, a svg document fragment is organized with elements grouped and nested in a very flexible manner. Attribute fields of a given element contains miscellaneous commonly-used properties, such as identification (“id”), coordinate system information (e.g. “width”, “height”, and “viewBox”), utility methods such as matrix operations (“transform/translate”), and the ability to control various graphics painting and text styling options (e.g. “style/fill” and “style/font-family”).

The combination of text-oriented format and URI referencing features of SVG greatly increase distribution and re-use capabilities of existing graphics object. Particularly, XML-based object model allows SVG element or SVG document to be embedded inline as a component for building complex application. Given the gap between visual and

XML-based SVG presentation, in subsequent sections, we explain how it is filled through DOM, the programmable interface.

DOM feature

SVG is an application of DOM [DOM2CORE] [DOM2EVENT]. The importance of SVG DOM is two folds: one is a DOM tree like structure with nodes of parsed graphics objects provides an effective programmable interface for complete access and manipulate of graphical elements and their properties; the other is inherent DOM event model plus language binding (e.g. JavaScript) and/or Cascading Style Sheets (CSS) [CSS] enables sophisticated application including interactivity, scripting and animation, which distinct SVG as a general-purpose presentational technology.

SVG DOM represents a hierarchical data structure of vector graphics. The static feature is depicted in fig. F.6. In implementation, the SVG DOM tree can be built upon parsing corresponding SVG document such as “rectangle.svg”, which has visual and XML presentations as shown in fig. F.5 and fig. F.6 respectively. The SVG document fragment ─ 'svg' element (with “body” identification) is a child node of XML document root node and contains multiple grouping elements whilst the leaf nodes are composed of basic shape element and text element such as 'rect' and 'text'.

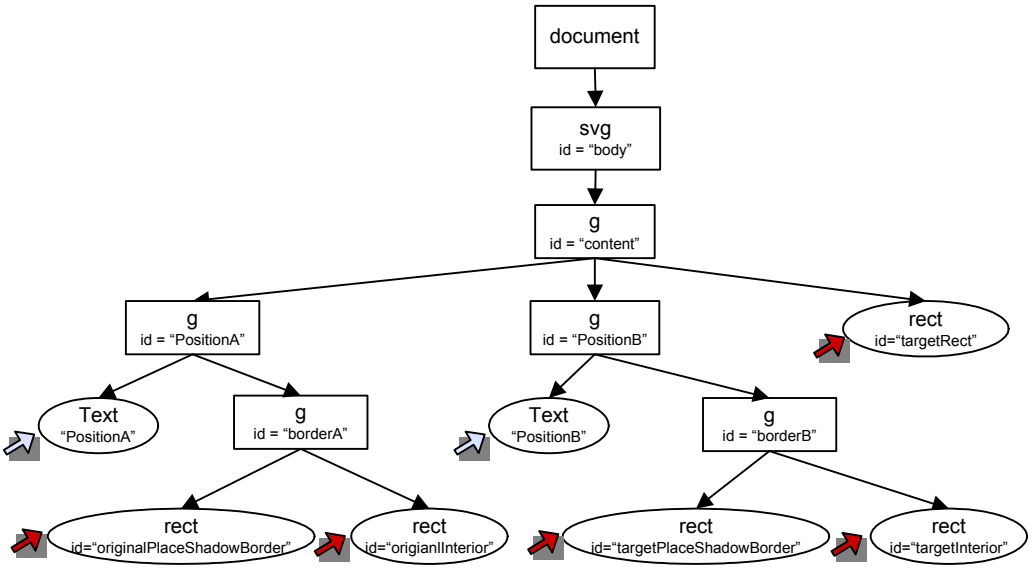


Figure F.5 SVG DOM tree representation of rectangle.svg document

Each SVG element node contains common attributes (e.g. id), element boundaries, as well as those belonged to a specific type such as styling and processing instructions. The tree structure makes it convenient for traversal and locating an element by means of canonical searching algorithms. The binding language provides *get* and *set* methods for applications to retrieve and modify element properties. These high level method calls are executed through SVG element interfaces (e.g. *getAttribute* and *setAttribute*) and generic DOM node interfaces (e.g. *insertBefore*, *replaceChild*, *removeChild*, and *appendChild*).

Rendering

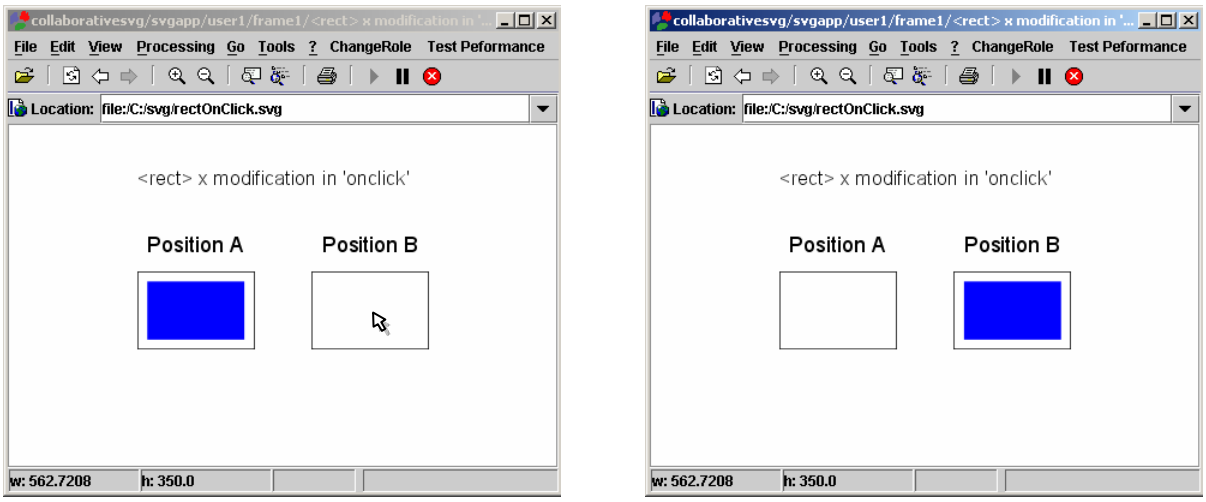
The rendering of SVG content is a process of converting text-oriented description of graphical objects to pixel-based bitmap image for output device. A *canvas* is regarded as an infinite virtual space to draw graphics on. In reality, SVG content is rendered and viewed in a finite rectangular area of canvas, so called *viewport*. SVG uses pixel unit in coordinate systems and top/left corner is defined as the origin of root viewport.

SVG employs “painters model” to render a target graphical object on *canvas* by conducting successive operations. For simple put, a graphical element is initially painted onto a temporary canvas, applied with all filtering and painting operations including clipping, masking and object opacity (alpha channel) before being merged into the background as a whole onto the output device.

There’re several issues that complicate SVG rendering process. First, coordinate system and transformation between canvas (viewport) space and user space; local (current node) and global (root node) coordinates within the user space. Second, three types of fundamental SVG graphical contents, which embraces shape, text, and raster image, are rendered differently. Third, graphical elements are commonly grouped and nested. SVG imposes implicit drawing order (with first parsed element getting “painted” first on bottom of rendering stack) in absence of *z-order* attributed in SVG 1.0 specification. Of course, graphics rendering consists of many algorithmic, technical, and optimization details that beyond what covers here.

F.4 An example of interactive SVG application

We use a simple example rectOnClick.svg to explain the interactive mechanism of SVG applications. A blue rectangle is located in frame holder A (see fig. F.4a). When a mouse click occurs over frame hold B, the rectangle toggles to position B (see fig. F.4b).



(a) ***before "mouse click" on*** **(b)** ***after "mouse click" on***

Figure F.6 A simple interactive SVG application of toggling rectangle

The example is defined in rectOnClick document shown in fig. F.7. Line 4 to 41 constitutes the main body of SVG document. It defines key elements and actions that can operate on them. Major graphical elements are defined in group from line 19 to 39, which consist of subgroups "Position A" and "Position B", and a blue rectangle. The subgroups are themselves composite as well, each containing text and border elements. These elements are parsed as SVG DOM nodes in memory. Note that event listeners are added to element nodes in border A and border B elements. A user input event (e.g. mouse click) would invoke call back method that is defined in JavaScript code segment from line 8 to 15 — that is, switching the blue rectangle from position A to position B. Graphical rendering engine produces visual reflection that accommodates to the change.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd">

<svg id="body" width="300" height="350" viewBox="0 0 300 350">

   <title>&lt;rect&gt; x modification in 'onclick'</title>

   <script type="text/ecmascript">
      function moveToX(evt, target, val){
         var r = evt.target;
         var doc = r.ownerDocument;
         var t = doc.getElementById(target);
         t.setAttribute('x', val);
      }
    </script>

   <text x="0" y="60" class="title" style="font-family: Arial; font-size:20">&lt;rect&gt; x modification in 'onclick'</text>

   <g id="content" transform="translate(0,120)">
      <g id="PositionA"  transform="translate(0,0)">
         <text class="legend" x="10" y="10" style="font-family: Arial; font-size:22">Position A</text>

         <g id="borderA" onclick="moveToX(evt, 'targetRect', '10')">
            <rect id="originalPlaceShadowBorder" x="0" y="30" width="120" height="80" style="fill:none; stroke:black" />
            <rect id="origianlInterior" x="1" y="31" width="118" height="78" style="fill:white; stroke:none" />
         </g>
      </g>

      <g id="PositionB" transform="translate(180,0)">
         <text class="legend" x="10" y="10" style="font-family: Arial; font-size:22">Position B</text>

         <g id="borderB" onclick="moveToX(evt, 'targetRect', '190')">
            <rect id="targetPlaceShadowBorder" x="0" y="30" width="120" height="80" style="fill:none; stroke:black" />
            <rect id="targetInterior" x="1" y="31" width="118" height="78" style="fill:white; stroke:none" />
         </g>
      </g>

      <rect id="targetRect" x="10" y="40" width="100" height="60" style="fill:blue;" />
   </g>

</svg>
```

Figure F.7 An interactive SVG example with scripting in rectOnClick.svg document

F.5 Summary

In summary, SVG provides a rich, structured description of vector and mixed vector/raster graphics. Different concepts of SVG in terms of vector graphics, XML conformation, and DOM structure represents important aspects of visual, XML-based format, URI reference, and hierarchical data structure that promise SVG more than a rudimentary graphical rich format but comprises of fundamental elements for a programmable environment. Conversion between different presentations of SVG: flattened text-oriented XML document, SVG DOM tree, and pixel-based bitmap image for output device.

Substantial implementations that based on SVG specification and SVG content have been developed in support of viewing, generating, transcoding, editing, and drawing [SVGIMPLEMENTATIONS]. Among SVG viewers that are developed for rendering SVG format content, Adobe [ADOBESVG] and Corel [CORELSVG] implemented SVG as a plug-in of a conventional browser; Apache Batik SVG is a stand-alone client application, which is written in Java apart from a few native classes; there are SVG Tiny implementations that are customized for handheld devices as well.

Appendix G

Overall architecture of Batik

The following Batik architecture [BATIK] illustrates three levels of modules: low level, core, and application that support SVG application deployment.

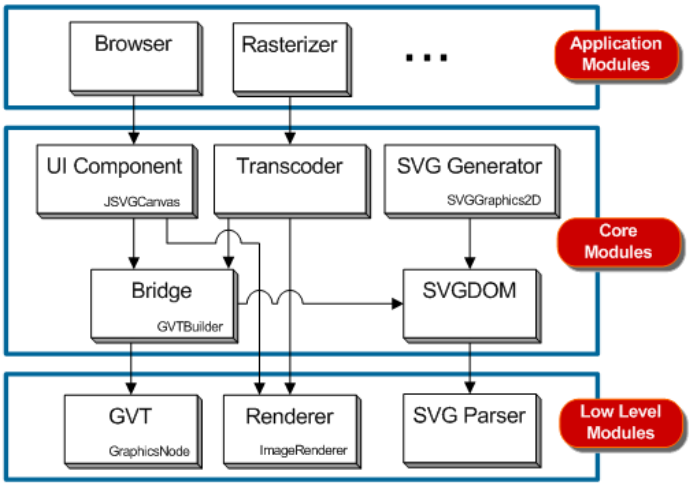


Figure E.1 Batik Architecture

Appendix H

Detail Analysis of Batik

Figure H.1 **JSVGViewerFrame**

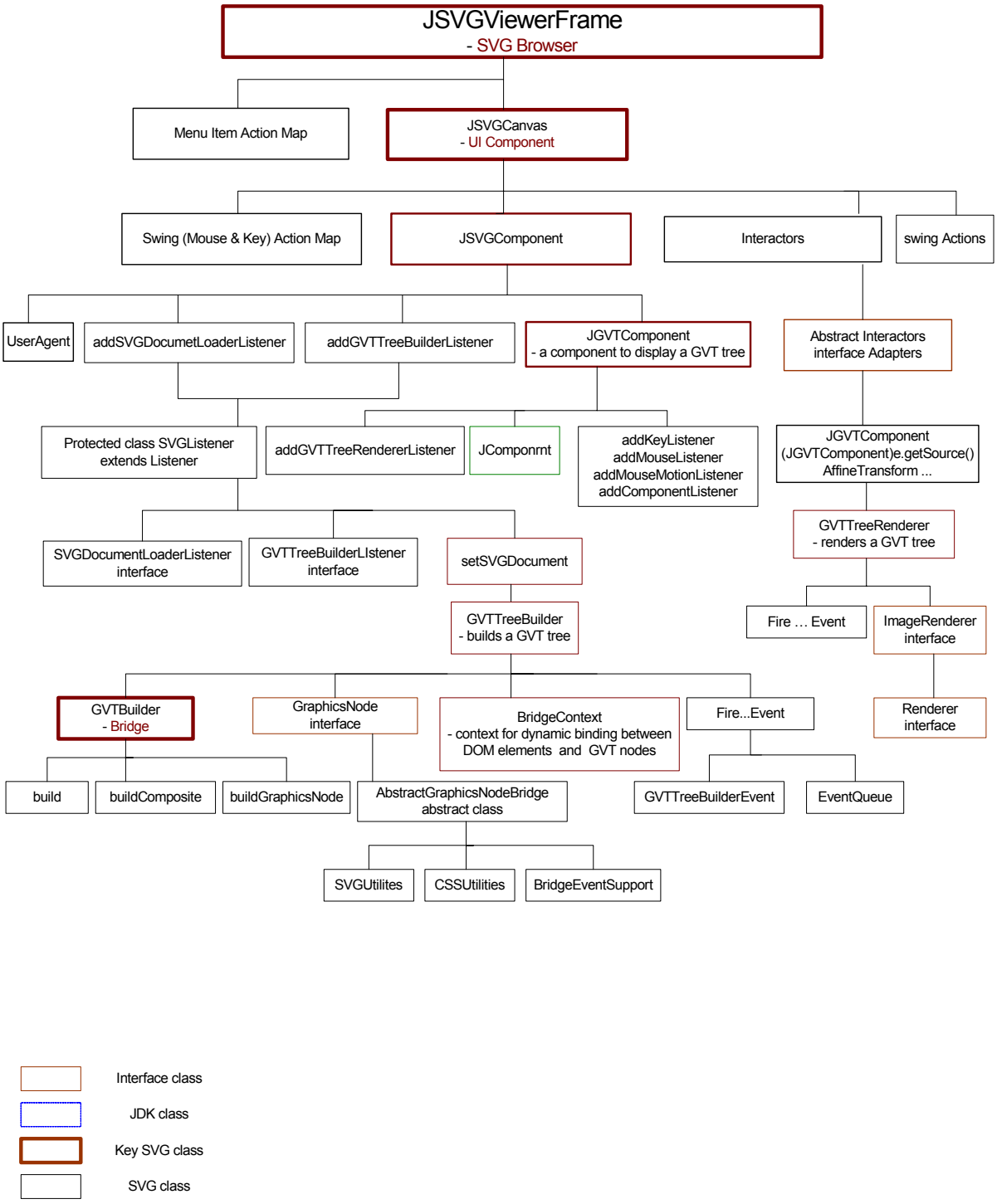


Figure H.2 **Data flow**

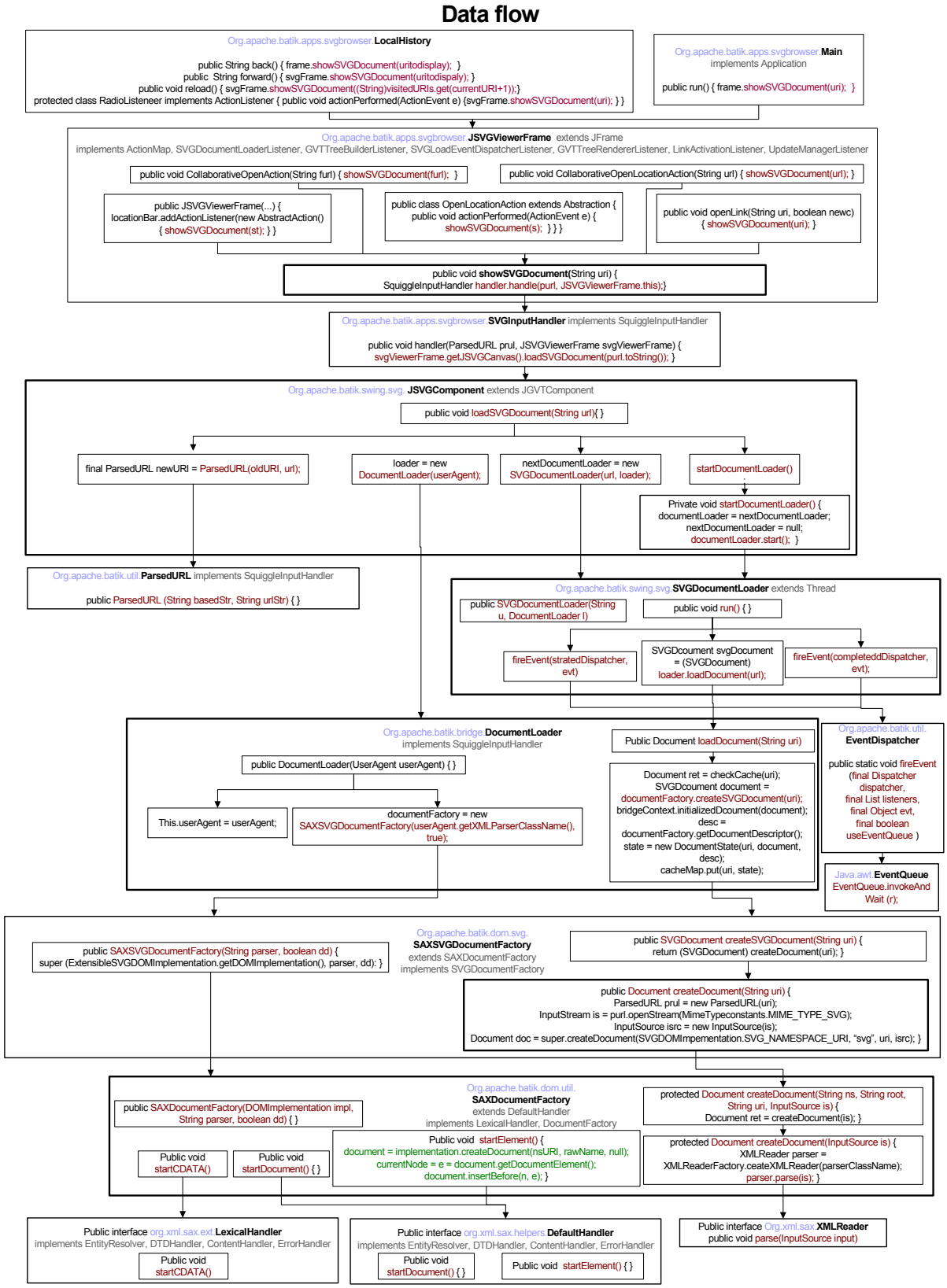


Figure H.3 **openLink**

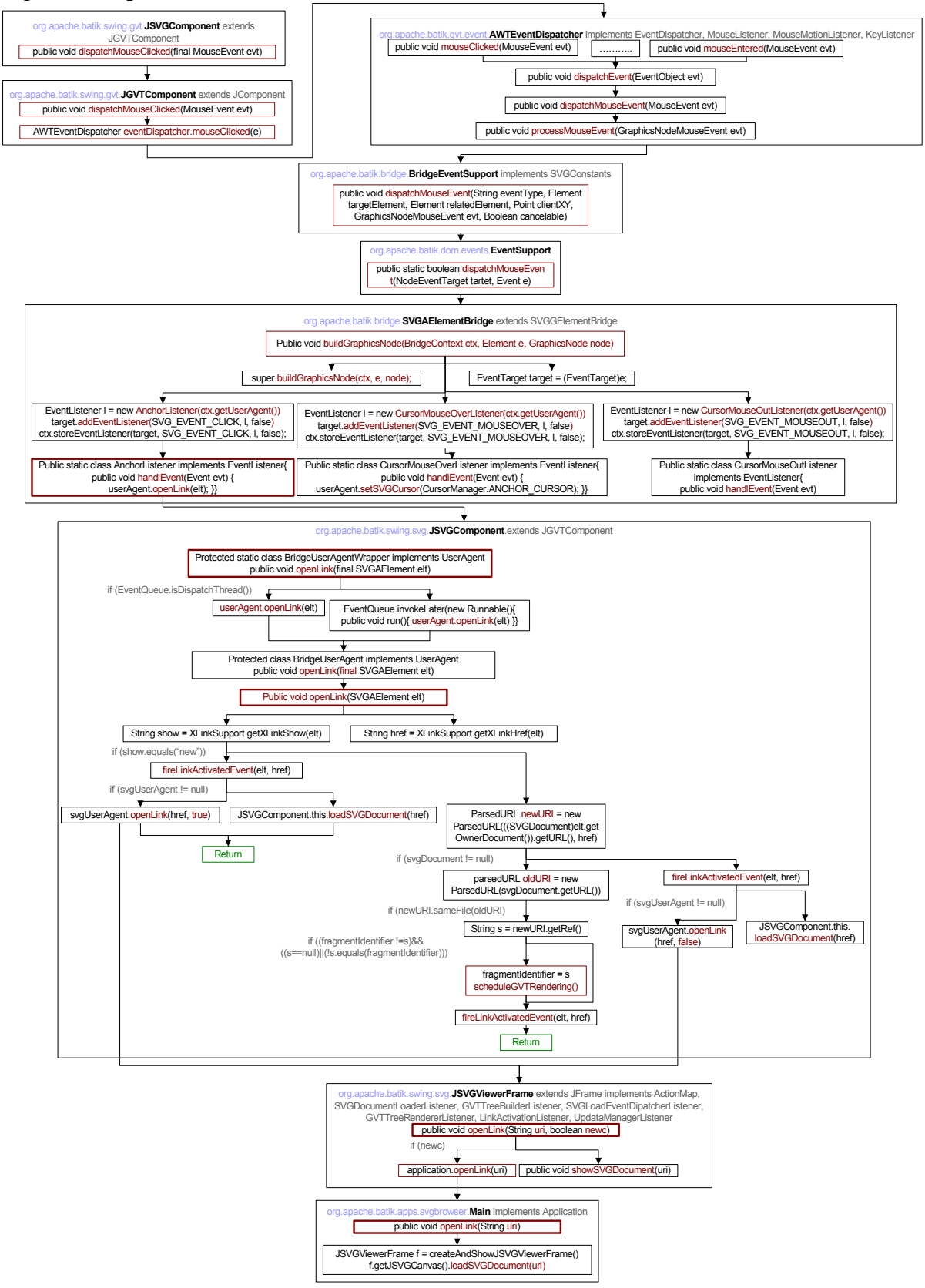


Figure H.4 **openLink (rendering)**

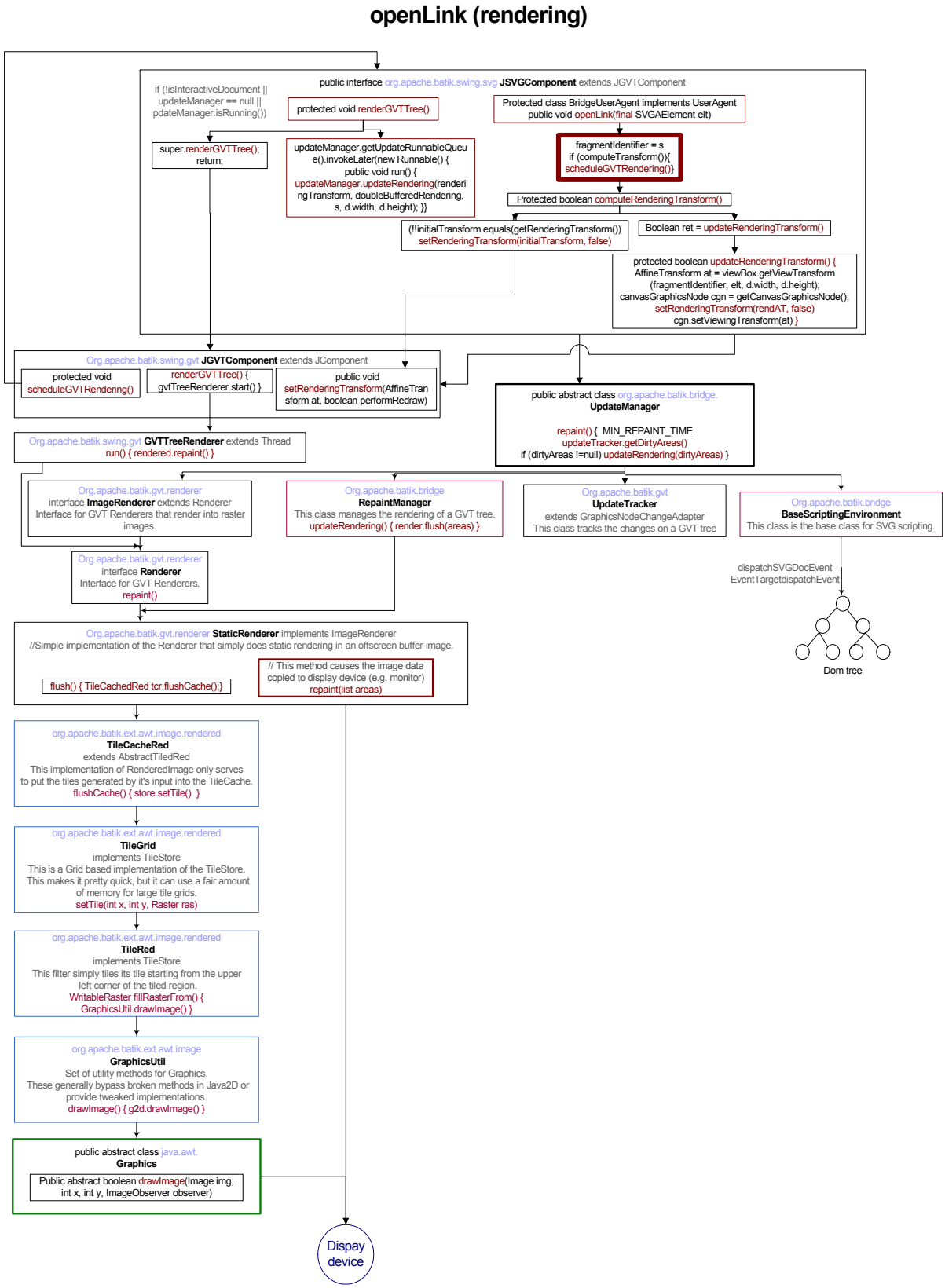


Figure H.5 **GraphicsNode Event (DOM-Bridge-GVT event flow)**

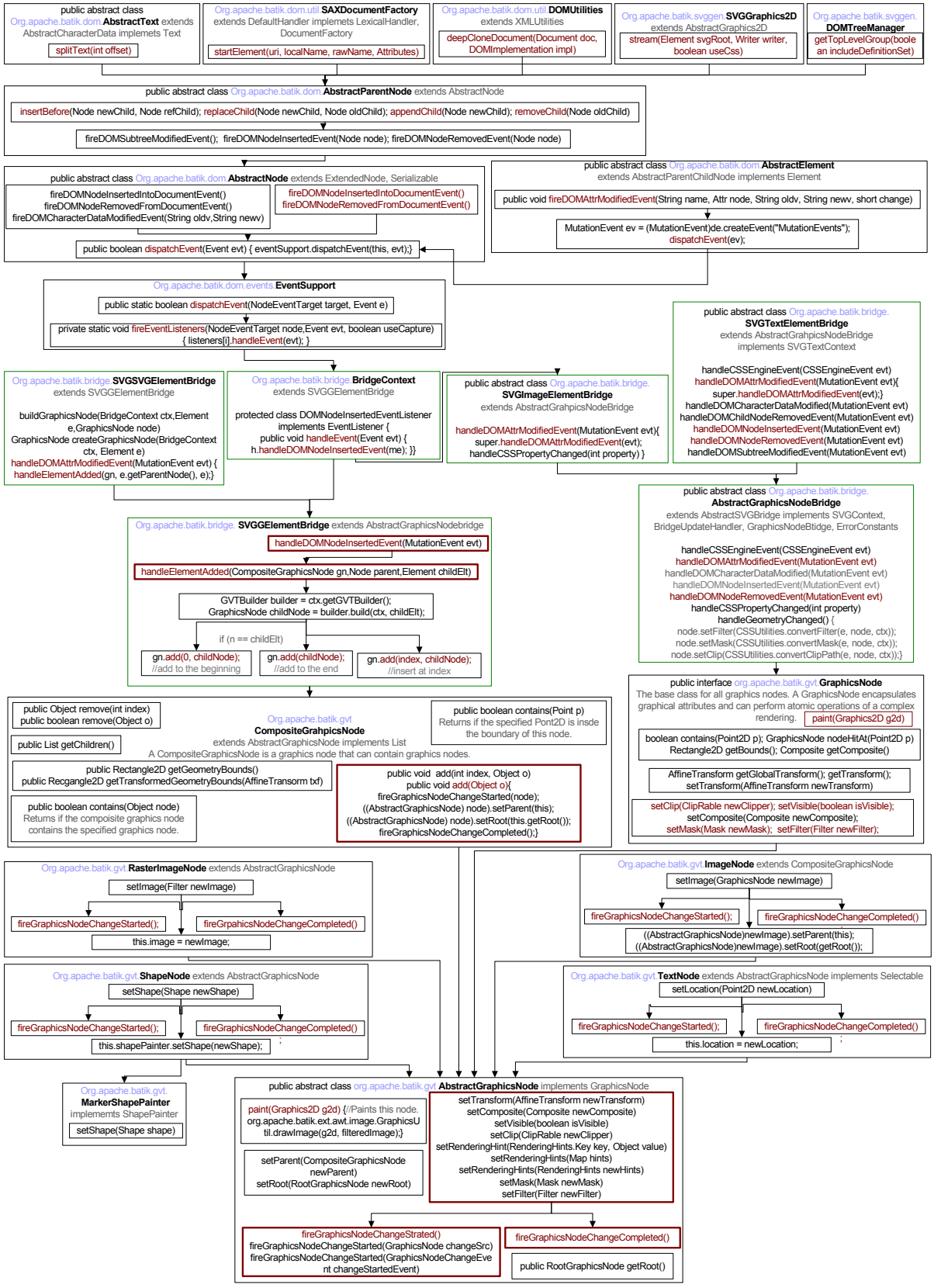


Figure H.6 **EventDispatcher**

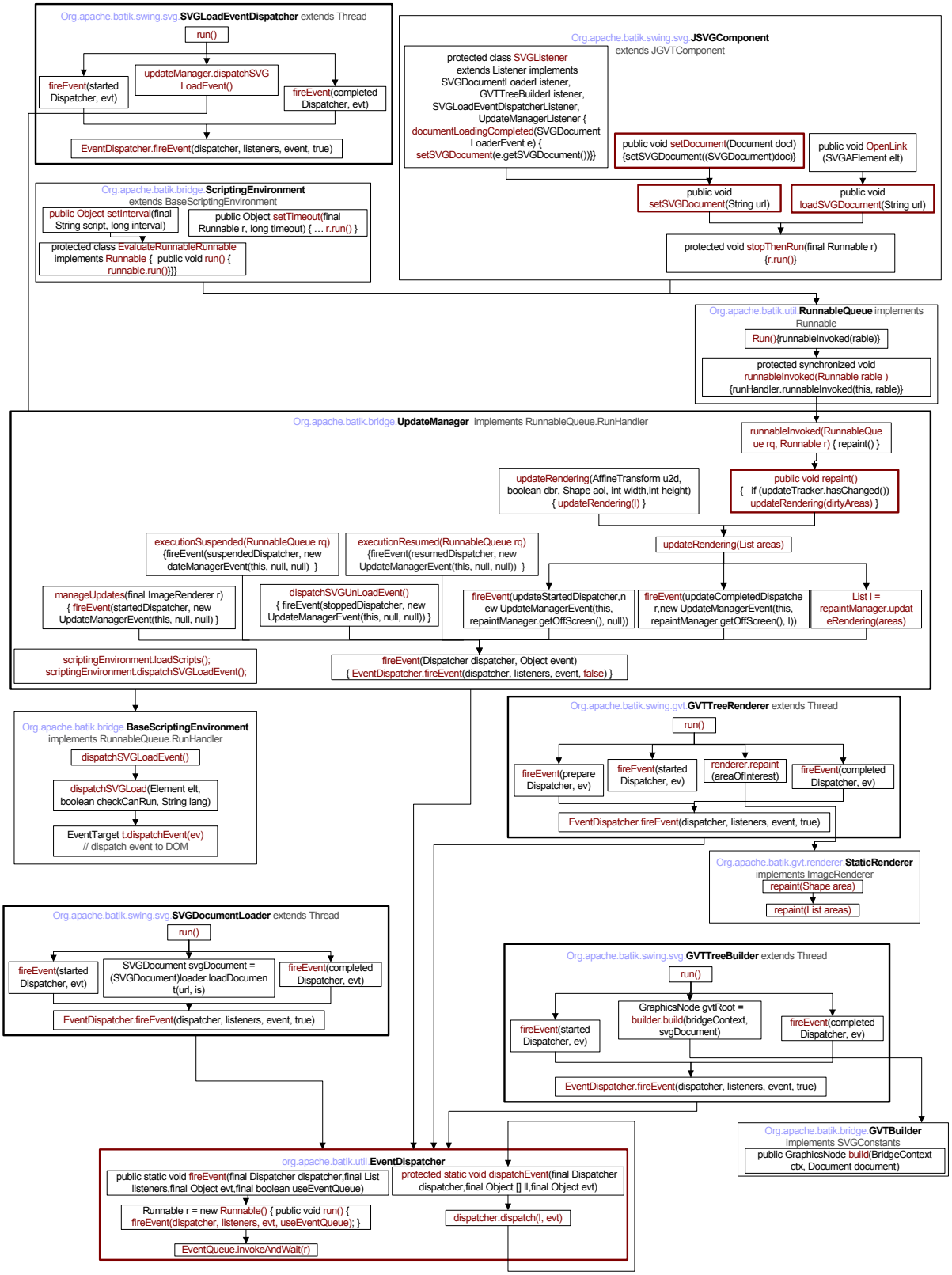


Figure H.7 **Handling of DOM Event**

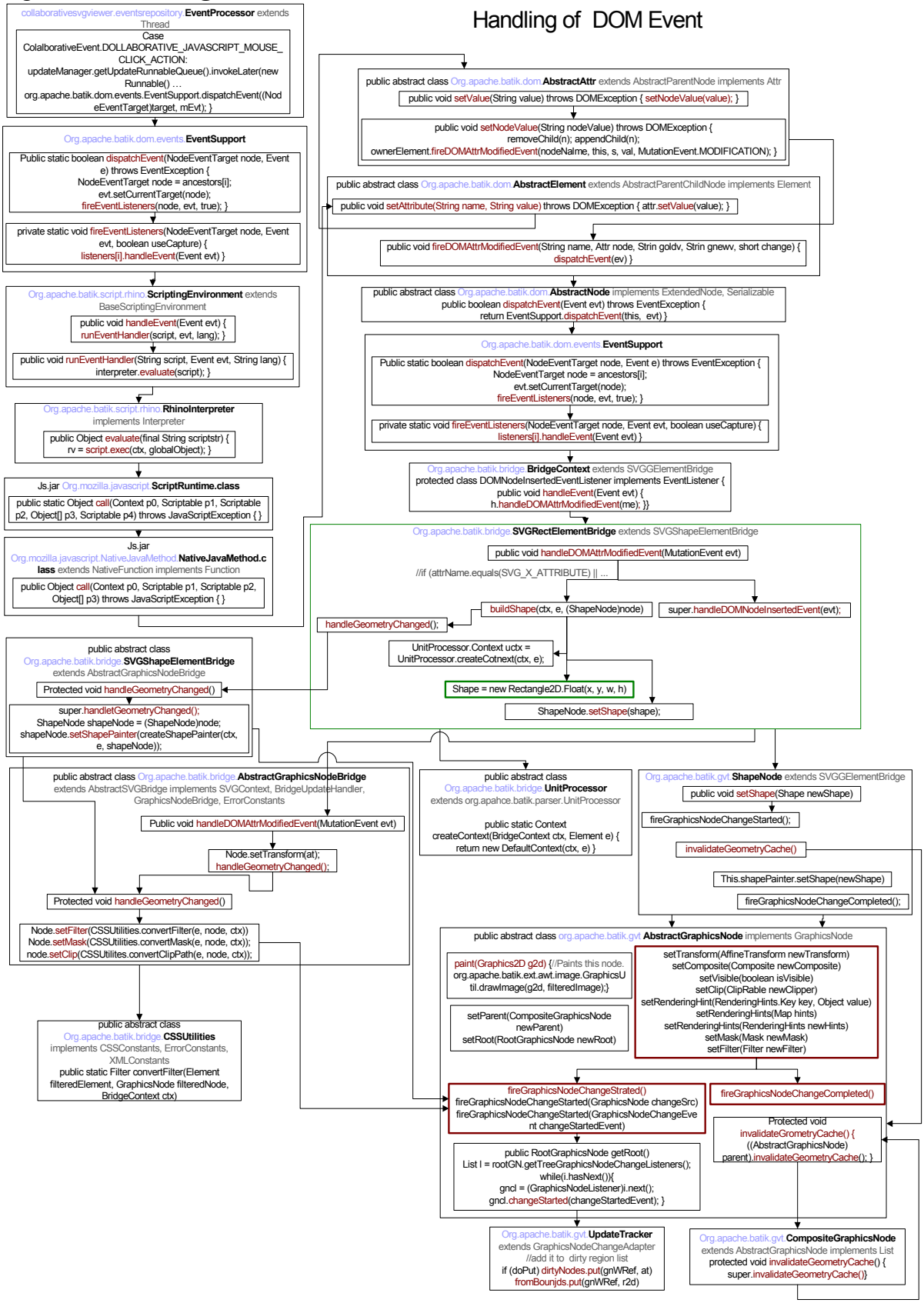


Figure H.8 **Batik component paint1**

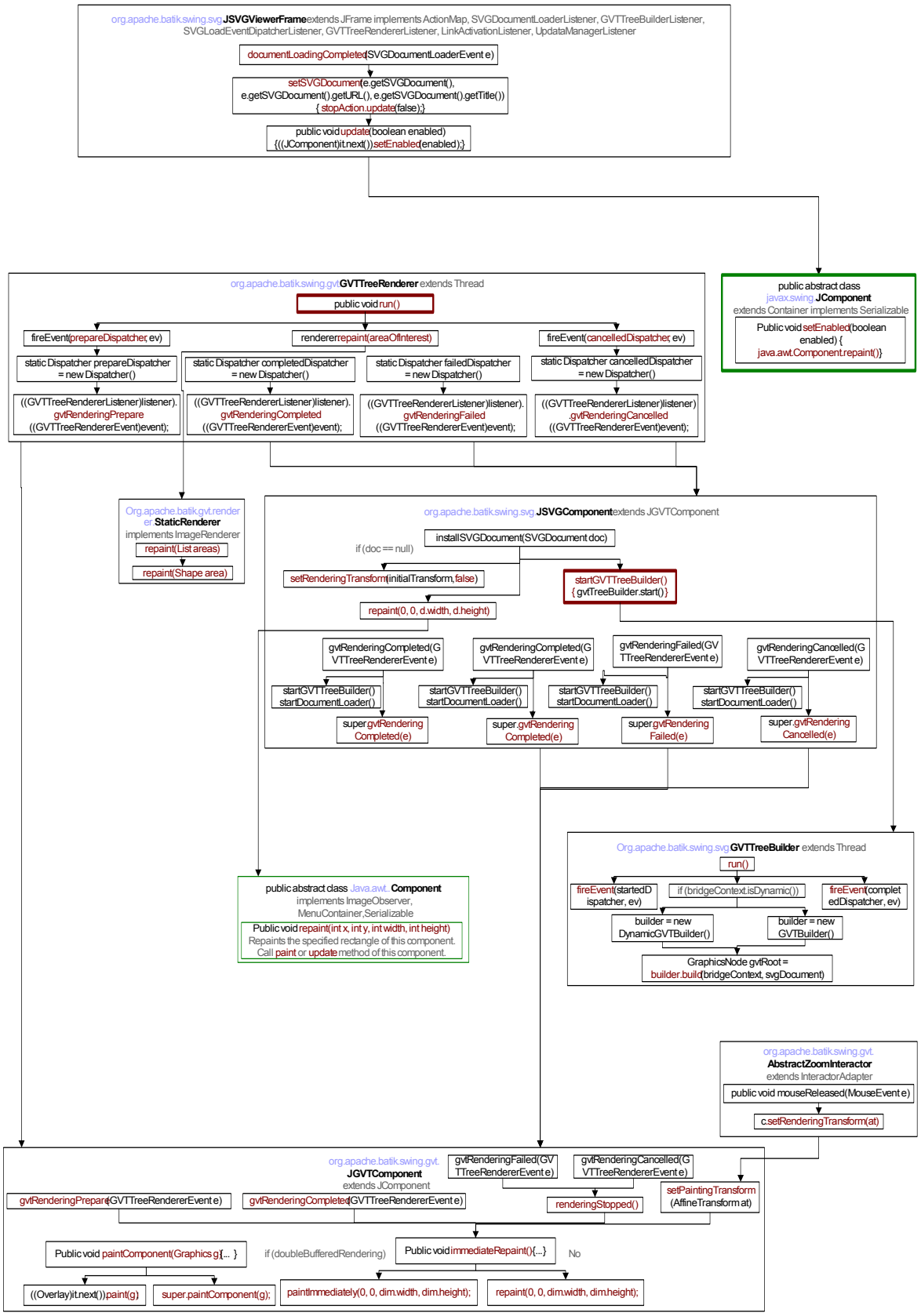


Figure H.9 **Batik component paint2**

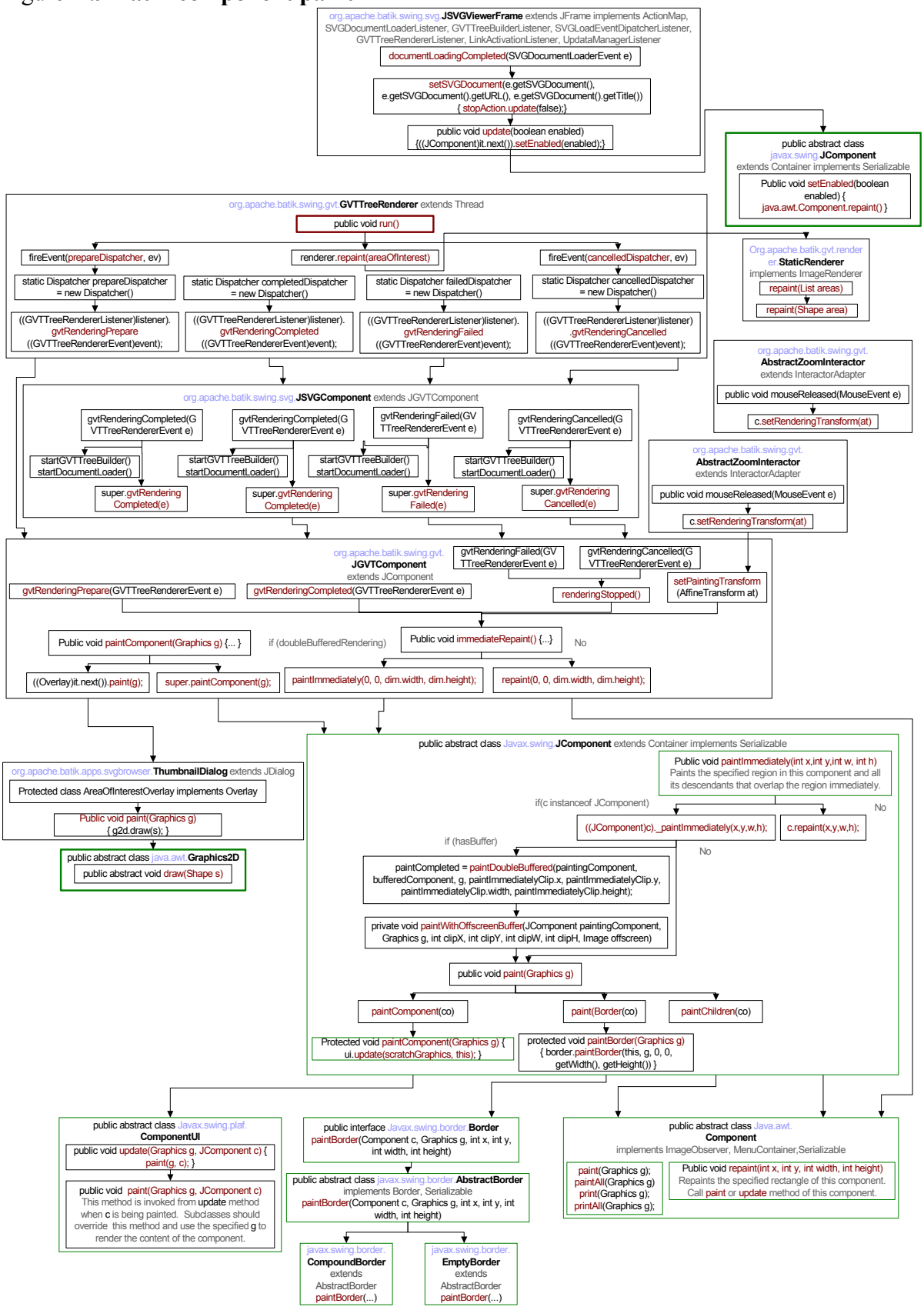


Figure H.10 **Batik Component Paint**

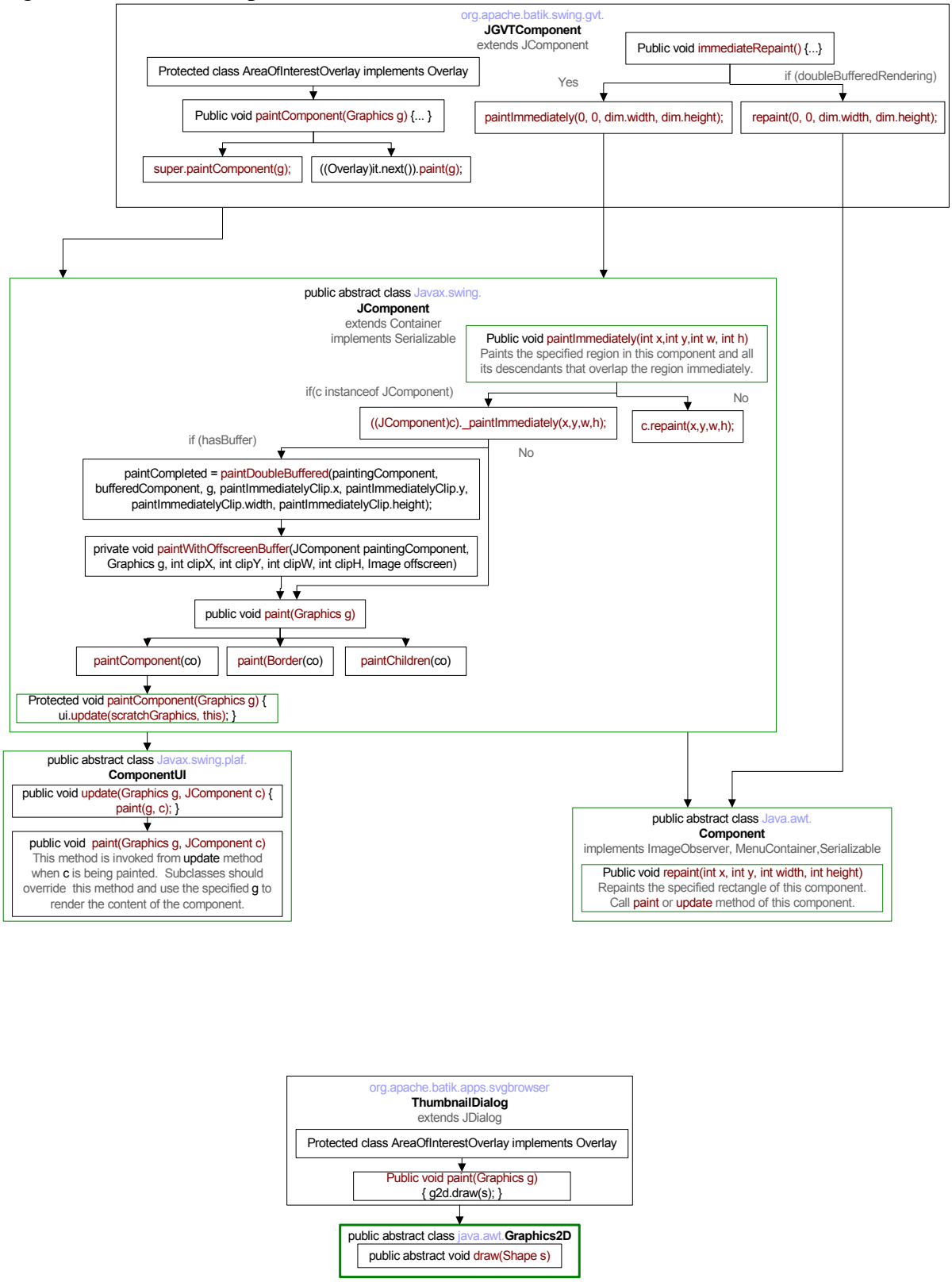


Figure H.11 **Graphics Node**

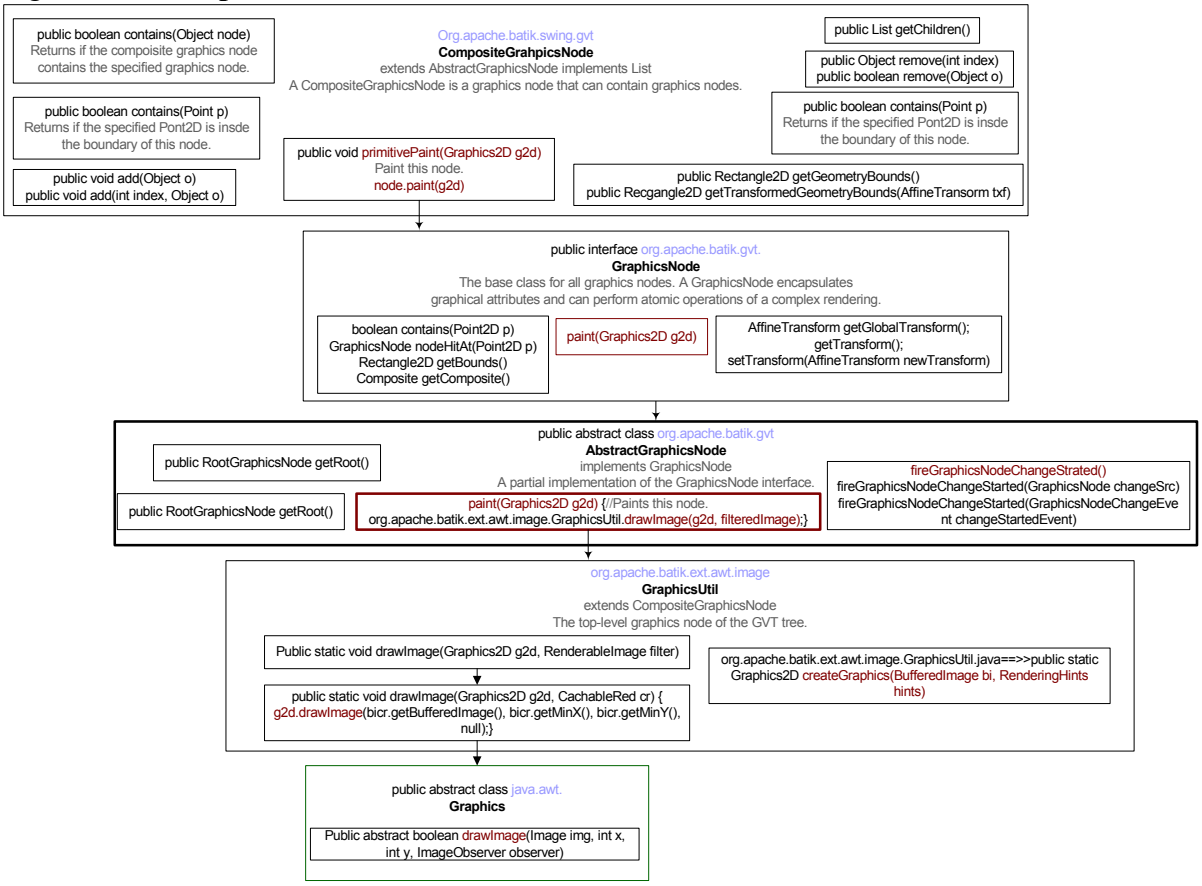


Figure H.12 **Graphics Node Paint**

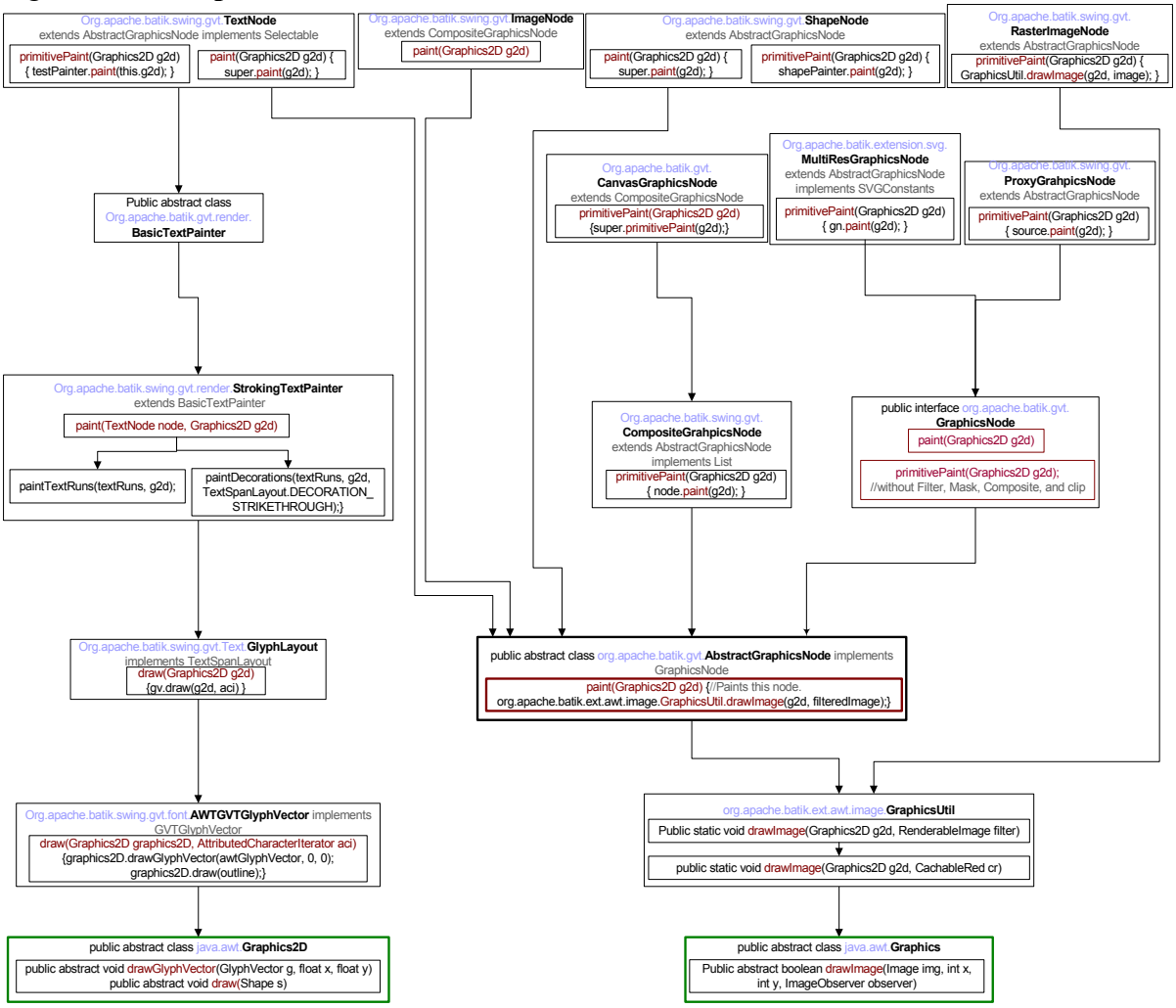
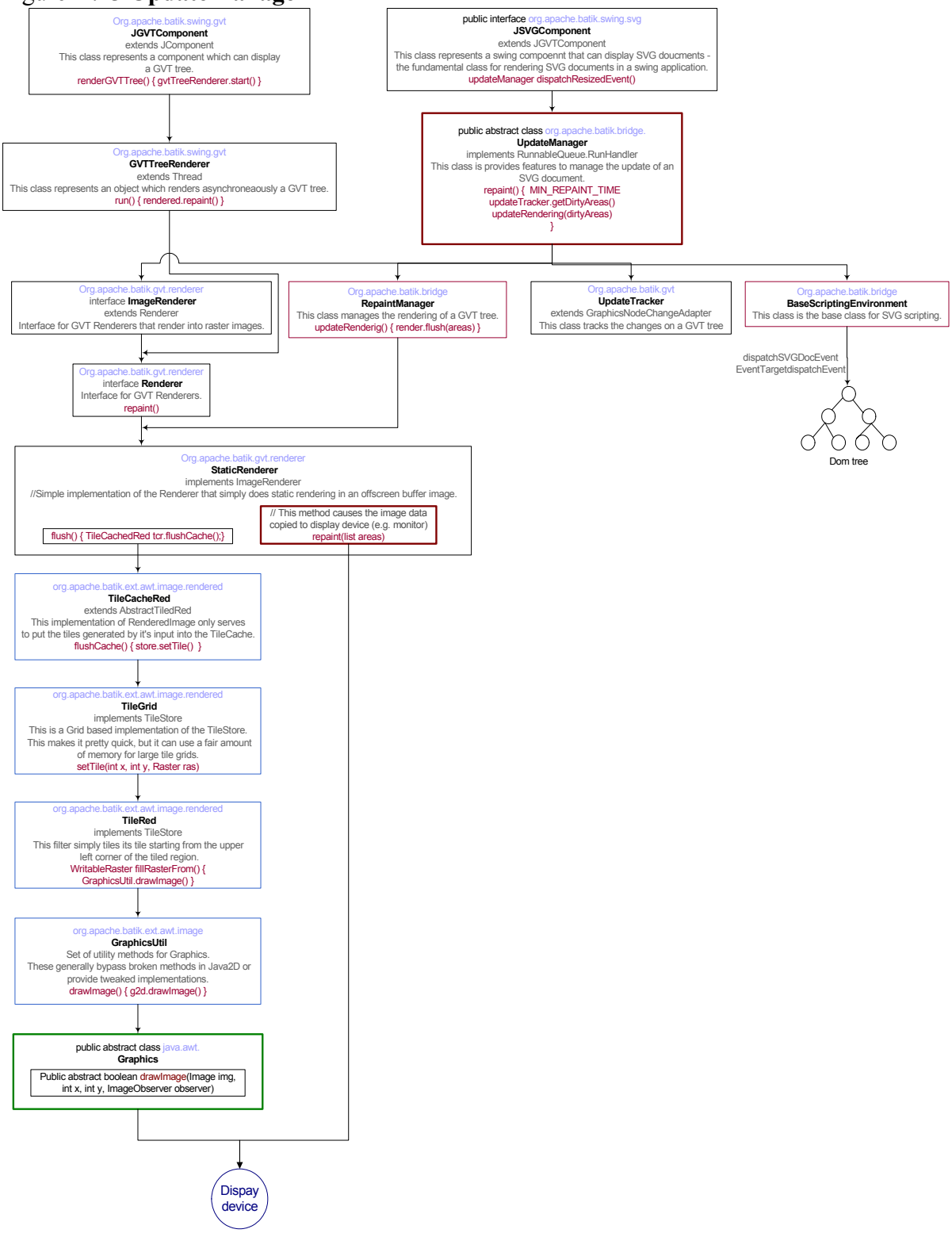


Figure H.13 **UpdateManager**



Appendix I
JavaScript event vs. AWT event

Table I.1 JavaScript event vs. AWT event

JavaScript Event	**AWT Event**
onclick	MOUSE_CLICKED
onmousedown	MOUSE_PRESSED
onmouseup	MOUSE_RELEASED
onmousemove	MOUSE_MOVED
onmouseover	MOUSE_ENTERED
onmouseout	MOUSE_EXITED

Bibliography

[A. EINSTEIN] Albert Einstein, “Everything should be made as simple as possible, but not simpler” in *Reader's Digest*. Oct. 1977.

[Abdullah+Gay] Abdul Hanan Abdullah and Brian Gay. Implementing an Interface to Networked Services. Proceedings of the 12th annual international conference on Systems documentation: technical communications at the greate divide. October 1994. http://delivery.acm.org/10.1145/200000/192532/p25-abdullah.pdf?key1=192532&key2=7906545011&coll=Portal&dl=ACM&CFID=36221945&CFTOKEN=19327503

[A.CHUANG] Alfred Chuang is CEO of EBA, a leading enterprise infrastructure software company. His remarks were made in a keynote address on the future of software at the Harvard University Business School and cited by *The Age of Convergence*, in BEA Features, March 2003. http://www.bea.com/framework.jsp?CNT=fea00006.htm&FP=/content/news_events/features_news/features

[ACTIONSCRIPT] Macromedia ActionScript Dictionary at http://www.macromedia.com/support/flash/action_scripts/actionscript_dictionary/actionscript_dictionary000.html

[ACTIVEX] ActiveX introduced in early 1996 based on Internet OLE components http://en.wikipedia.org/wiki/Component_object_model

[ADCOCK2004] Vincent T. Adcock, *Implementing an integrated SVG application for real time dynamically generated Internet mapping*, Proceedings of SVGOpen 2004, Tokyo, Japan. http://www.svgopen.org/2004/papers/SVG_Open_Abstract/

[ADOBEVIEWER] Adobe SVG Zone at http://www.adobe.com/svg/ featuring the version 3 viewer 2004

[ADOBESVG] Adobe Systems Incorporated, *SVG Viewer 3.0*. http://www.adobe.com/svg/main.html

[A. Huk] Alex Huk. *Seeing Motion: Lecture Notes*. The material comes from course text book: Sensation and Perception (5th Edition) by E. Bruce Goldstein. http://www-psych.stanford.edu/~lera/psych115s/notes/lecture7/

[ALGOL60] ALGOL is a computer programming language originally developed in 1958 and released as ALGOL 60 in 1960.

[ANABAS]Anabas, Inc. provides the next-generation of collaboration software platforms enabling a new class of interactive, rich-media capable collaborative applications. http://www.anabas.com/

[ANSI SMALLTALK] National Committee of Information Technology Standard (NCITS). *ANSI Smalltalk Standard version 1.9*. NCITS J20 Draft. December 1997. http://www.smalltalk.org/versions/ANSIStandardSmalltalk.html

[APACHE] Apache Software Foundation at http://www.apache.org/

[ARPANET] The ARPANET from the US Defense Department Advanced Research Projects Agency (ARPA) initiated in 1969 http://www.funet.fi/index/FUNET/history/internet/en/arpanet.html

[ASP] Active Server Pages **(**ASP**)** is Microsoft's server-side technology for dynamically-generated web pages that is marketed as an adjunct to Internet Information Server (IIS). It has evolved to ASP.NET and competes with technologies like PHP, Python, CGI. http://en.wikipedia.org/wiki/Active_Server_Pages

[Atkinson et. al.] Malcolm Atkinson, David DeRoure, Alistair Dunlop, Geoffrey Fox, Peter Henderson, Tony Hey, Norman Paton, Steven Newhouse, Savas Parastatidis, Anne Trefethen and Paul Watson. *Web Service Grids: An Evolutionary Approach*. UK e-Science Technical Report. July 13, 2004. To be published in a special issue of Concurrency & Computation: Practive & Expereince Magazine 2005.

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

[AXIS] Axis is an Apache project that implements SOAP protocol by W3C. http://ws.apache.org/index.html

[BAKKEN] David E. Bakken. *Middleware*. Encyclopedia of Distributed Computing. Kluwer Academic Press, 2003.

[BATIK1.5] Batik Scalable Vector Graphics SVG browser (version 1.5 release) at http://xml.apache.org/batik/index.html

[BCW1986] Francois Bodin, Francois Charot, and Charles Wagner, *Overview of a high-performance programmable pipeline structure*, Proceedings of the 3rd international conference on Supercomputing Crete 1986 at http://delivery.acm.org/10.1145/320000/318868/p398-bodin.pdf?key1=318868&key2=3930276801&coll=portal&dl=ACM&CFID=22251036&CFTOKEN=35504884

[BFF2003] Luca Piazza Bonati , Luciano Fortunati, and Giuseppe Fresta, *SVG Explorer of GML Data*, Proceedings of SVGOpen 2003, Vancouver, Canada. http://www.svgopen.org/2003/papers/SvgExplorerOfGmlData/index.html

[BLOG] Blog is a shared online journal or frequently updated personal web page. http://www.blogger.com/start

[BOOTH] David Booth et al. *Web Services Architecture*. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/

[CARROLL] J.M. Carroll (Ed.) *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. The MIT Press, Cambridge, MA, pp. 80-111.

[CGI] Common Gateway Interface or CGI was an early World Wide Web technology that specified how a client web browser passed data to and from a program executed on a Web server. http://en.wikipedia.org/wiki/Common_Gateway_Interface

[CGL] Community Grids Lab (CGL) of Indiana University. CGL publications Web site at http://grids.ucs.indiana.edu/ptliupages/publications/.

[CKB] Luis Felipe Cabrera, Christopher Kurt, and Don Box. *An introduction to the Web Services Architecture and Its Specifications*. Microsoft Technical White Paper archived in MSDN library. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/introwsa.asp

[COM+] In 1993, Microsoft released OLE 2, and created COM as the underlying object model for OLE 2 ; in 1996 DCOM was introduced in response to challenge of CORBA. Finally with Windows 2000, a significant extension to COM named COM+ was introduced. At the same time, Microsoft de-emphasized DCOM as a separate entity. .NET is replacing all these technologies. http://en.wikipedia.org/wiki/Component_object_model

[CORELSVG] Corel Corporation, *Corel SVG Viewer*. http://www.smartgraphics.com/Viewer_prod_info.shtml

[CORBA] Common Object Request Broker Architecture or CORBA**,** is a standard for software components created and controlled by the Object Management Group (OMG). I http://en.wikipedia.org/wiki/CORBA

[CORBAEVENTSERVICE] Object Management Group Inc., Event Service Specification (version 1.1), Object Management Group Inc., available at http://www.omg.org/docs/formal/01-03-01.pdf

[CORBANOTIFICATIONERVICE] Object Management Group Inc., *Notification Service Specification*, Object Management Group Inc., available at http://www.omg.org/docs/formal/00-06-20.pdf

[CORELDRAW] Corel Draw at http://www.corel.com/servlet/Satellite?pagename=Corel2/Products/AllProducts

[Cox+Novobilski] Brad J. Cox and Andrew J. Novobilski. *Object-Oriented Programming, An Evolutionary Approach*. Second Edition, May 1991. ISBN: 0201548348.

[C. Yu] Colin Yu. *Migrating a Struts application to Webshpere Portal*. IBM WebSphere Developer Technical Journal. March, 2004. http://www-900.ibm.com/developerworks/cn/wsdd/techjournal/0403_yu/0403_yu_eng.shtml

[DCE] Distributed Computing Environment or DCE was produced by the Open Software Foundation OSF and was an early suite of distributed computing technologies such as RPC, DNS, Kerberos Security and the Andrew file system. http://www.linktionary.com/d/dce.html

[DCOM] See [COM+]

[DHTML] Dynamic HTML in Netscape Navigator at http://developer.netscape.com/docs/manuals/communicator/dynhtml/index.htm

[DNS] The Domain Name System or DNS, first invented in 1983 by Paul Mockapetris, is a distributed database that maps between host names and the numerical IP address. http://www.yourencyclopedia.net/DNS.html

[DOM] W3C, Document Object Model (DOM) at http://www.w3.org/DOM/Activity.html

[DOM1] W3C, Document Object Model (DOM) Level 1 Specification at http://www.w3.org/TR/REC-DOM-Level-1/

[DOM2CORE] Document Object Model (DOM) Level 2 Core Specification at http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/Overview.html

[DOM3CORE] Document Object Model (DOM) Level 3 Core Specification at http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022/Overview.html

[DOM2EVENT] Document Object Model (DOM) Level 2 Events Specification at http://www.w3.org/TR/DOM-Level-2-Events/Overview.html

[DOM3EVENT] Document Object Model (DOM) Level 3 Events Specification at http://www.w3.org/TR/DOM-Level-3-Events/Overview.html

[DOTNET] Microsoft's distributed software and Web Service platform introduced in 2002 and replacing COM http://en.wikipedia.org/wiki/Microsoft_.NET

[EBXML] Paul Levine et al. *ebXML Business Process Specification Schema (version 1.01)*. http://www.ebxml.org/specs/ebBPSS.pdf

[EJB] Java Server side Component Technology with support for transactions http://java.sun.com/products/ejb/

[EMAIL] Electronic mail specification at http://www.faqs.org/rfcs/rfc733.html

[ENIAC] ENIAC or the Electronic Numerical Integrator And Computer**,** was the earliest programmable electronic computer and was unveiled on February 14, 1946 at the University of Pennsylvania and was transferred to the US Army Aberdeen Proving Grounds, Maryland in 1947. http://www.seas.upenn.edu/~museum/

[ETHERNET] Dominant Data Link Protocol introduced in 1973 http://en.wikipedia.org/wiki/Ethernet

[FDDI] Fiber Distributed Data Interface (FDDI) is a token ring based data link layer transmission protocol originally aimed at fiber but generalized to other media. It featured 100 mbits/sec performance and has been largely made obsolete by fast(gigabit) Ethernet http://en.wikipedia.org/wiki/Fiber_distributed_data_interface

[FLASH] Macromedia Flash at http://www.macromedia.com/

[FLASHWHITEPAPER] *Flash Player for Developers and Publishers*, Macromedia FLASH white paper 2003. http://www.macromedia.com/software/flash/survey/whitepaper_jul03.pdf

[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.

[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

[Fox94] G. Fox, R. Williams, and P. Messina. *Parallel Computing Works*. Morgan Kaufmann Publishers Inc., 1994. Section 3.4. http://www.npac.syr.edu/copywrite/pcw/

[Fox98] Fox, G., Scavo, T., Bernholdt, D., Markowski, R., McCracken, N., Podgorny, M., Mitra, D., and Malluhi, Q., *Synchronous Learning at a Distance: Experiences with TANGO Interactive*. Supercomputing 98 Conference, November 1998. http://www.old-npac.org/projects/training/Papers/sc98/

[FREENET] Freenet at http://freenet.sourceforge.net/

[FTP] File Transfer Protocol (FTP) at http://www.faqs.org/rfcs/rfc454.html

[FWUBP] Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara. *Global Multimedia Collaboration System*. Proceedings of the 1st International Workshop on Middleware for Grid Comoputing co-located with Middleware 2003. June 17, 2003. Rio de Janeiro, Brazil.

[Gao+Yuen] Yaoqing Gao and Chungkwong Yuen. *A Survey of Implementations of Concurrent, Parallel and Distributed Smalltalk*. ACM SIGPLAN Notices. Pages: 29-35. Volume 23, Issue 9. September, 1993. ACM Press, New York. ISSN: 0362-1340

[GENERATOR] *Architecture and Technical Discussion*, Macromedia GENERATOR White Paper February 2000, http://itpapers.zdnet.com/abstract.aspx?&scid=193&x=40&docid=28324

[G. Fox] Geoffrey Fox. *Grids of Grids of Simple Services*. CISE Magazine. July/August 2004.

[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

[Girow+Mitgartz] Andrew Girow and Edik Mitgartz. *SVG Tiny Cartoons on Java Devices*. Proceedings of SVG Open Conference, Tokyo Japan September 2004. http://www.svgopen.org/2004/papers/SVGTinyCartoonsOnJavaDevices/

[G. Krasner] Glenn E. Krasner. *Smalltalk-80, Bits of History, Words of Advice*. Addision-Wesley, Boston, MA, 1983. ISBN:0-201-11669-3.

[GNUTELLA] Gnutella is a simple effective peer-to-peer technology developed in 2000 – originally at AOL http://en.wikipedia.org/wiki/Gnutella

[GOOGLE] GOOGLE is search engine that focuses exclusively on organizing the world's information on the World Wide Web and making it universally accessible and useful. http://www.google.com/about.html

[GOODMAN] Danny Goodman, *Dueling Event Models – A Cross-Platform Look*, column The JavaScript Apostle, View Source online magazine. http://developer.netscape.com/viewsource/goodman_events2/goodman_events2.html

[Goldberg+Robson] Aele Goldberg and David Robson. *Smalltalk-80: The Language and its Implementation.* Addison Wesley, Reading, MA, 1983. ISBN: 0-201-11371-6. Smalltalk, generally released as Smalltalk-80, is a dynamically typed object oriented

programming language designed at Xerox Palo Alto Research Center during the 1970s.

[G. Seshadri] Govind Seshadri. *Understanding JavaServer Pages Model 2 architecture: Exploring the MVC design pattern*. JavaWorld. http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[GOTTSCHALK] Karl Gottschalk et al. *Web Services Architecture Overview*. Archived article of IBM Web Services Architecture Team. September, 2000. http://www-106.ibm.com/developerworks/webservices/library/w-ovr/#author1

[GRIDS] See review *The Grid: Past, Present, Future* Fran Berman Geoffrey Fox Tony Hey 2002 at http://www.grid2002.org/grid2002sample/chapter1.pdf , Chapter 1 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.

[GRYPHON] IBM, GRYPNON is a messaging middleware system using publish/subscribe paradigm for supporting complex distributed applications. Home page is at http://www.research.ibm.com/gryphon/. A brief version of White paper is available at http://www.research.ibm.com/gryphon/papers/Gryphon-Overview.pdf

[HCI] Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank. *ACM SIGCHI Curricula for Human-Computer Interaction*. SIGCHI. http://sigchi.org/cdg/cdg2.html#2_1

[H. Lieberman] Henry Lieberman. *Using Prototypical Objects to Implement Shared Behavior in Object Oriented System*. Proceedings of the First ACM Conference on

Object-Oriented Programming Systems, Languages, and Applications. Pages 214-223. Portland, Oregon. October 1986.

[HTTP] Hypertext Transfer Protocol at http://www.w3.org/Protocols/HTTP/AsImplemented.html

[HTML] W3C, HyperText Markup Language (HTML) verison 4.0 specification at http://www.w3.org/TR/1998/REC-html40-19980424/

[IE] Microsoft Internet Explorer at http://www.microsoft.com/windows/ie/default.asp

[IKMWK] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. *Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself*. Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Volume 32 Issue 10. October 1997. http://delivery.acm.org/10.1145/270000/263754/p318-ingalls.pdf?key1=263754&key2=5855545011&coll=Portal&dl=ACM&CFID=36221945&CFTOKEN=19327503

[ILLUSTRATOR] Adobe Illustrator at http://www.adobe.com/products/illustrator/overview.html

[IM] International Engineering Consortium, *Instant Messaging ─ Definition and Overview*, International Engineering Consortium, 2004. Instant messaging (IM) is an Internet protocol (IP)–based application that provides communication services. An IM platform can offers a portal interface to an integrated real time messaging services like text chat and Voice-over-IP. http://www.iec.org/online/tutorials/instant_msg/

[INTEROPERABILITY] EBA, *Reality Check: Java + .NET -- Interoperability Matters*, in BEA Features, April 2003.

http://www.bea.com/framework.jsp?CNT=fea00007.htm&FP=/content/news_events/features_news/features

[INTERNETWORLDSTATS] Internet World Stats, *Internet Usage Statistics − The Big Picture*, updated on September 1, 2004, http://www.internetworldstats.com/stats.htm

[INTERNET 1969] All About the Internet at http://www.isoc.org/internet/history/

[IP] Internet Protocol is the network layer protocol used by the internet http://en.wikipedia.org/wiki/Internet_protocol

[J2EE] The Server Side Java 2 Enterprise Edition http://java.sun.com/j2ee/

[JAVA] Revolutionary programming language introduced in by Sun in 1995 http://en.wikipedia.org/wiki/Java_programming_language

[JAVA AWT] Java Abstract Windowing Toolkit interfacing to machine native user interface technology for client side Java. http://en.wikipedia.org/wiki/Java_2_Platform,_Standard_Edition

[JAVA EVENT] Sun Microsystems Inc., Java AWT: Delegation Event Model, Sun Microsystems Inc., 1997. http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html

[JAVASCRIPT 1995] JavaScript Web scripting language at http://devedge.netscape.com/central/javascript/

[JAVA SWING] Java graphical user interface technology largely replacing the AWT http://en.wikipedia.org/wiki/Swing_(Java)

[J. Bennett] John K. Bennett. *The design and implementation of distributed Smalltalk*. Proceedings on Object-oriented programming systems, languages and applications. Pages: 318-330. 1987. ACM Press, New York. ISSN: 0362-1340.

http://delivery.acm.org/10.1145/40000/38836/p318-bennett.pdf?key1=38836&key2=0394735011&coll=Portal&dl=ACM&CFID=36221945&CFTOKEN=19327503

[J.COOPER] James W. Cooper. *The Design Patterns ─ Java Companion*. Addison-Wesley Design Patterns Series, MA. 1998

[Jordan+Hendersion] B. Jordan and A. Henderson. *Interaction Analysis: Foundations and Practice*. The Journal of Learning Sciences. Lawrence Erlbaum Associates, Inc., Vol. 4, No. 1, pp.39-103.

[JDBC] The Java Database Connectivity JDBC allows Java programs to query and update relational databases http://encyclopedia.thefreedictionary.com/JDBC

[JETSPEED] Apache open source portal http://portals.apache.org/jetspeed-2/

[J. Josephraj] Jerome Josephraj. *Architect Struts applications for Web Services: Bring the power of the MVC pattern to the Web services domain*. IBM technical article. April, 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-arcstruts

[JLHB] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. *Fine-Grained Mobility in the Emerald System*. ACM Trans. on Computer Systems 6(1), February 1988. http://citeseer.ist.psu.edu/cache/papers/cs/1041/http:zSzzSzwww.cs.washington.eduzSzhomeszSzlevyzSzopalzSzemerald.pdf/jul88finegrained.pdf

[JMS] Sun Microsystems Inc., Java Message Service, Sun Microsystems Inc., 1999 http://java.sun.com/products/jms/docs.html

[JMSTUTORAL] Kim Haase, *Java Message Service Tutorial*, Sun Microsystem Inc., http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/jms_tutorialTOC.html

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

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

[JSR168] Sun Microsystems. *Introduction to JSR 168 ─ The Java Portlet Specification*. Sun Microsystems. http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/pb_whitepaper.pdf

[JXTA] SUN JXTA peer-to-peer project at http://people.jxta.org/servlets/ProjectHome

[KALLIO] Kiia Kallio, Using SVG for graphically rich 2D content in mobile 3D games, Proceedings of SVGOpen 2003, Vancouver, Canada. http://www.svgopen.org/2003/papers/UsingSVGFor2DContentInMobile3DGames/index.html

[KLEINDIENST] Jan Kleindienst, *BeanChannel: Java Distributed Event Model*, Ph.D. Thesis, Charles University Prague, 1998. http://nenya.ms.mff.cuni.cz/publications/pds98.pdf

[Kojarski+Lorenz] Sergei Kojarski and David H. Lorenz. *Domain Driven Web Development With WebJinn*. The 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. October 2003. http://delivery.acm.org/10.1145/950000/949351/p53-kojarski.pdf?key1=949351&key2=6453115011&coll=portal&dl=ACM&CFID=35366781&CFTOKEN=1129766

[Krasner+Pope] Glenn E. Krasner and Stephen T. Pope. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*. Journal of Object Oriented Programming. Volumn 1, Issue 3. Pages: 26-49. Aug/Sept 1998.

[LAN] Local Area Network describes a computer network supporting a "small" localized area http://en.wikipedia.org/wiki/LAN

[LDAP] Lightweight Directory Access Protocol (LDAP) is an IETF protocol for accessing on-line directory services. http://en.wikipedia.org/wiki/LDAP

[LEE94] Lee G., *Object oriented GUI application development,* Prentice Hall 1994, ISBN**:** 0-13-363086-2.

[LFKWQ] Sangmi Lee, Geoffrey Fox, Sunghoon Ko, Minjun Wang, Xiaohong Qiu. *Ubiquitous Access for Collaborative Information System Using SVG*. Proceedings of SVG Open Conference 2002. July 2002. Zurich, Switzerland.

[LWLH] Guanchun Luo, Yanhua Wang, Xianliang Lu, and Hong Han. *A Novel Web Application Frame Developed by MVC. ACM SIGSOFT*. Software Engineering Notes Vol. 28. No.2. March, 2004. http://delivery.acm.org/10.1145/640000/638779/p7-chun.pdf?key1=638779&key2=9833494011&coll=Portal&dl=ACM&CFID=35145809&CFTOKEN=34172557

[MEIER+CAHILL] René Meier, Vinny Cahill, Taxonomy of Distributed Event-Based Programming Systems, 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02).

[MESSENGER] Microsoft, *Instant Messaging Overview*, Microsoft, http://www.microsoft.com/windowsxp/using/helpandsupport/learnmore/remoteassist/viaim.mspx

[MIPS] Microprocessor without interlocked pipeline stages at http://en.wikipedia.org/wiki/MIPS_architecture

[MML] M. Keith Mortensen, Rob McGovern and Charles Liptaak. *ASP.NET and Struts: Web Application Architectures*. Microsoft technical article. December 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/aspnet-aspnet-j2ee-struts.asp

[MOORE] Moore's law on computer performance increasing exponentially in time at http://en.wikipedia.org/wiki/Moore's_Law

[MOSAIC] Mosaic was the first multimedia browser with a good graphical user interface produced by NCSA in 1993 http://en.wikipedia.org/wiki/Mosaic_browser

[MOZILLA] Mozilla Web browser at http://www.mozilla.org/

[MOZILLALAYOUT] Mozilla Layout Engine at http://www.mozilla.org/newlayout/

[MQSeries] Messaging middleware from IBM –WebSphereMQ formerly called MQSeries -- http://www.ibm.com/software/integration/wmq/

[MSMQ] Microsoft Message Queuing (MSMQ) technology http://www.microsoft.com/windows2000/technologies/communications/msmq/default.asp

[MÜHL] Gero Müuhl, *Large-Scale Content-based Publish/Subscribe Systems*, Ph.D. thesis, 2002. http://elib.tu-darmstadt.de/diss/000274/dissFinal.pdf

[MVC] G. Lee, *Object oriented GUI application development*. Prentice Hall, 1994. ISBN: 0-13-363086-2. Model-View-Controller (MVC) is an object oriented architecture that separate presentation from system data structure into triads of Model, View, and Controller.

[MYERSON] Judith M. Myerson. *Web Services Architectures ─ How they stack up*. Web Service Architect at http://www.webservicesarchitect.com/content/articles/myerson01.asp

[NAPSTER] Original application that popularized peer-to-peer file sharing and recently restarted in "legal fashion" http://www.napster.com/

[NARADABROKERING] Open Source Messaging Internet System from the Community Grids Laboratory at http://www.naradabrokering.org

[NETSCAPE] Netscape at http://channels.netscape.com/ns/browsers/

[NFS1989] Network File System Protocol at http://www.ietf.org/rfc/rfc1094.txt?number=1094

[NIELSEN] Nielsen//NetRatings, *Three Out of Four Americans Have Access to the Internet*, report on March 18, 2004, http://www.netratings.com/pr/pr_040318.pdf

[ODBC] Open Database Connectivity or ODBC is a standard software API for connecting to database management systems (DBMS). http://en.wikipedia.org/wiki/ODBC

[OGCE] Open Grid Computing Environments Collaboratory. http://www.collab-ogce.org.

[OGM IDL] OMG IDL Syntax and Semantics defined in http://www.omg.org/technology/documents/formal/corba_2.htm

[OLE] Object Linking and Embedding (OLE) is a Microsoft technology used primarily for copying and pasting data between different applications. It later evolved to become an architecture for software components known as the component object model (COM). http://en.wikipedia.org/wiki/Object_linking_and_embedding

[OMG] Object Management Group, organization website at http://www.omg.org/

[OMG-MESSAGING] OMG, *Data Distribution Services for Real-Time Systems Specification*. OMG. http://www.omg.org/docs/ptc/03-07-07.pdf

[OPENDOC] OpenDoc, introduced by Apple Computer in 1992, was a software component framework standard for compound documents, similar to Microsoft's OLE. http://en.wikipedia.org/wiki/OpenDoc

[OPENINVENTER] Open Inventor, originally IRIS Inventor, is a C++ object oriented 3D graphics API designed by SGI to provide a high level programming abstraction for OpenGL. http://encyclopedia.worldsearch.com/open_inventor.htm

[OPENOFFICE] Open Source cross platform Office Suite http://www.openoffice.org/

[OSI] OSI at http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/introint.htm#xtocid5

[PALLICKARA-06-04] Shrideep Pallickara and Geoffrey Fox, *Efficient Matching of Events in Distributed Middleware Systems*, Journal of Digital Information Management. Volume 2, Issue 2. pp 79-87. June 2004. Special Issue of selected papers from the IEEE ITCC 2004 Track on Modern Grid and Web Systems. http://grids.ucs.indiana.edu/ptliupages/publications/jdim-vol2-num2.pdf

[PARALLELCOMPUTING] Linkage of multiple computers together in a closely coupled fashion to solve a single problem. See The Sourcebook of Parallel Computing edited by Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, Morgan Kaufmann, November 2004.

[P2P] A peer-to-peer (or P2P) computer network refers to any network that does not have fixed clients and servers, but a number of *peer* nodes that function as both clients and servers. http://en.wikipedia.org/wiki/Peer-to-peer

[PERL] Perl, also Practical Extraction and Report Language is a programming language released by Larry Wall on December 18, 1987 that borrows features from C, sed, awk, and the UNIX Shell. http://en.wikipedia.org/wiki/Perl_programming_language

[PC 1981] he IBM PC (Personal Computer) was introduced in August 1981 and is forerunner of current personal computers.

[PHOTOSHOP] Adobe PhotoShop at http://www.adobe.com/products/photoshop/overview.html

[P. McCullough] Paul L. McCullough. *Transparent Forwarding: First Steps*. Proceedings on Object-oriented programming systems, languages and applications. Volume 22, Issue12. December 1987. ACM Press, New York.

[POWERPOINT] Microsoft PowerPoint at http://office.microsoft.com/home/office.aspx?assetid=FX01085797&CTT=6&Origin=ES790020011033

[PUB/SUBNOTIFICATION] IBM, *Publish-Subscribe Notification for Web Services*, IBM. http://www-106.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf

[QIU-10-2000] Xiaohong Qiu, a demo of research presentation web site using Flash 5.0 at http://aspen.ucs.indiana.edu/project/research/

[QIU-01-2001] Xiaohong Qiu, *Macromedia Flash 5.0 and Generator 2.0*, technical report available at http://grids.ucs.indiana.edu/ptliupages/publications/flashjan01.html

[QCF-06-03]Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox, *Internet Collaboration using the W3C Document Object Model* in Proceedings of the 2003 International Conference on Internet Computing, Las Vegas June 2003.

http://grids.ucs.indiana.edu/ptliupages/publications/collaborative_dom_conference_2003_Int_IC_font10_without_title_page.pdf

[QCF-07-03] Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox, *Collaborative SVG as a Web Service* in Proceedings of SVG Open Conference, Vancouver July 2003. http://www.svgopen.org/2003/papers/CollaborativeSVGasAWebService/#S.Bibliography

[Qiu+Jooloor]Xiaohong Qiu and Anumit Jooloor, *Web Service Architecture for e-Learning*, EISTA 2004 International Conference on Education and Information Systems: Technologies and Applications, July 21-25 2004 Orlando. http://grids.ucs.indiana.edu/ptliupages/publications/E388NH.pdf

[QPU]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/

[QOS] Quality of Service usually referencing communication or network http://www.encyclopedia4u.com/q/quality-of-service.html

[R. Fielding] Roy Thmoas Fielding. Architectural Styles and the Design of Network-based Software Architecture. Ph.D. thesis. 2000. University of California, Irvine. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[RISC] Reduced Instruction Set Computer at http://en.wikipedia.org/wiki/RISC

[RMI] Sun Microsystems, Java Remote Method Invocation Technology, Sun Microsystems, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html. RMI implementing RPC style connection between distributed Java objects

[S. Burbeck] Steve Burbeck, *Application Programming in SmallTalk-80: How to use Model-View-Controller (MVC)*. Smalltalk-80 Archive. at http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[SCHMIDT] Douglas C. Schmidt, *An Overview of OMG CORBA Event Services*, available at http://www.cs.wustl.edu/~schmidt/PDF/coss4.pdf

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

[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++.

[SOA] 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.

[SONICMQ] Sonic Software Corp., *Technical Overview: SonicMQ Featuers & Benefits*, Sonic Software Corp., http://www.sonicsoftware.com/products/sonicmq/technical_overview/index.ssp

[SQL] Structured Query Language (SQL) is the most popular query language used with databases. http://en.wikipedia.org/wiki/SQL

[S. Shi] Sherlia Shi. *Design of Overlay Networks for Internet Multicast*. Page 6-7. Ph.D. Thesis. August, 2002. Washington University in St. Louis.

[SSJ] Inderjeet Singh, Beth Steans, and Mark Johnson. *Designing Enterprise Applciations with the J2EE Platform, Second Edition*. Sun technical article. 2002. http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html

[STRUTS] Apache. Struts is an open source project that provides a standard MVC framework to address flow control issues for Java-based Web applications. It is based on JSP model 2 and MVC design. http://struts.apache.org/userGuide/introduction.html

[SVG] W3C Scalable Vector Graphics (SVG) version 1.0 Specification http://www.w3.org/TR/2001/PR-SVG-20010719/

[SVGIMPLEMENTATIONS] The official W3C list of a variety of SVG implementations or applications. http://www.w3.org/Graphics/SVG/SVG-Implementations

[SVGMOBILE] W3C, *Mobile SVG Profiles: SVG Tiny and SVG Basic*, W3C. http://www.w3.org/TR/SVGMobile/

[SYMBIAN] Symbian Ltd., *Symbian OS Technology*, Symbian Ltd., Symbian OS is an operating system for data-enabled mobile phones. http://www.symbian.com/technology/technology.html

[TCL] Tcl ("Tool Command Language" pronounced "tickle") is a scripting language created by John Ousterhout that is commonly used for rapid prototyping, scripted applications, GUIs and testing. http://en.wikipedia.org/wiki/Tcl

[TCP] Transportation Control Protocol or TCP is a connection-oriented, reliable delivery byte-stream transport layer protocol currently documented in IETF RFC 793. http://en.wikipedia.org/wiki/Transmission_Control_Protocol

[TCP/IP] TCP/IP refers to the two most important protocols in the Internet protocol suite with transport layer TCP and Network layer IP: http://en.wikipedia.org/wiki/TCP/IP

[T. DeWeese] Thomas DeWeese. *Java Applications with Apache Batik*. ApacheCon U.S. 2003.

[TELNET1972] Telnet protocol at http://www.faqs.org/rfcs/rfc318.html

[Tim B.L.] Tim Berners-Lee, comments on *new idea which will revolutionize computing*, available at http://www.w3.org/People/Berners-Lee/FAQ.html

[TOKENRING] Token ring is a local area network protocol which resides at the data link layer (DLL) of the OSI model. http://en.wikipedia.org/wiki/Token_ring

[UDDI] OASIS. *Universal Description, Discovery, and Integration (UDDI)*. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[UPORTAL] uPortal is an open-standard effort using Java, XML, JSP and J2EE for a free, sharable institutional portal as an abridged and customized version of the institutional Web presence. http://www.uportal.org/

[UWBF] Ahmet Uyar, Wenjun Wu, Hasan Bulut, Geoffrey Fox. *An Integrated Videoconferencing System for Heterogeneous Multimedia Collaboration*. 7th

IASTED International Conference on INTERNET AND MULTIMEDIA SYSTEMS AND APPLICATIONS ~IMSA 2003~ August 13-15, 2003 Honolulu, Hawaii, USA.

[UDP] The User Datagram Protocol (UDP) is a minimal message-oriented transport layer protocol that is currently documented in IETF RFC 768. http://en.wikipedia.org/wiki/User_Datagram_Protocol

[UNIX] UNIX is a portable, multi-task and multi-user computer operating system originally developed by AT&T Bell Labs. http://en.wikipedia.org/wiki/Unix

[URL] Uniform Resources Locator is a type of URI at http://www.ietf.org/rfc/rfc2396.txt

[USENET] Usenet or Unix User Network was an early (1979) Newsgroup system that initially used UUCP as a communication mechanism. http://en.wikipedia.org/wiki/Usenet

[UUCP 1976] UUCP or Unix to Unix Copy Protocol, was an early (developed at AT&T Bell Labs in 1976 and released a year later) UNIX suite of Shell commands and a protocol allowing remote execution of commands and transfer of data between Unix computers. http://en.wikipedia.org/wiki/UUCP

[VB] Visual Basic (VB) is an event driven programming language derived from BASIC and supporting ActiveX. VBScript, Visual Basic for Applications and VisualBasic.NET were built on VB. http://msdn.microsoft.com/vbasic/

[Veit+Herrmann] Matthias Veit and Stephan Herrmann. *Model-View-Controller and Object Teams: A Perfect Match of Paradigms*. Proceedings of the 2nd international conference on Aspect-oriented software development. 2003. Pages: 140-149. ACM Press, New York. ISBN: 1-58113-660-9.

[Viller+Sommerville] S. Viller and I. Sommerville. Social analysis in the requirements engineering process: *from ethnography to method*. Proceedings of International Symposium on Requirements Engineering. Limerick 1999, IEEE Computer Soc. Press, pp. 6-13.

[VIRGILLITO] Antonino Virgillito, *Publish/Subscribe Communication Systems: from Models to Applications*, Ph.D. thesis, University of Rome, 2003. http://www.dis.uniroma1.it/~virgi/virgillito-thesis.pdf

[VONNEUMANN] Von Neumman Computer at http://encyclopedia.thefreedictionary.com/Von%20Neumann%20computer

[VPN] A Virtual Private Network**,** or VPN, is a private communications network used within an enterprise built as an overlay network on the public internet http://en.wikipedia.org/wiki/Virtual_private_network

[W3C] World Wide Web Consortium at http://www.w3c.org/

[WAN] Wide Area Network covering a broad geographical region http://encyclopedia.thefreedictionary.com/WAN

[Wang+Fox] Minjun Wang and Geoffrey Fox. *Design of a Collaborative System for Open Office*. Proceedings of IASTED KSCE Conference Virgin Islands. November 2004.

[WBUF] Wenjun Wu, Hasan Bulut, Ahmet Uyar, Geoffrey C. Fox. *A Web-Services Based Conference Control Framework For Heterogenous A/V Collaboration*. 7th IASTED International Conference on Internet and Multimedia Systems and Applications ~ IMSA 2003. August 13-15, 2003. Honolulu, Hawaii, USA.

[WEBSERVICE] A web service is a collection of interoperable protocols and standards used for exchanging data between applications running on heterogeneous platforms that have been accepted by essentially the entire computer community. According to W3C, Web services provide "a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks." http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[WEBSITEINFLASH] Research Presentation Web site using Macromedia Flash at http://aspen.ucs.indiana.edu/project/research/

[WEBSPHEREMQ] IBM, *WebSphere MQ*, IBM, http://www-306.ibm.com/software/integration/wmq/v521/ [WIRELESS] IEEE 802.11 at http://grouper.ieee.org/groups/802/11/

[WFP+04] Minjun Wang, Geoffrey Fox and Shrideep Pallickara. *Demonstrations of Collaborative Web Services and Peer-to-Peer Grids*. Journal of Digital Information Management. Volume 2, Issue 2. June 2004. Special Issue of selected papers from the IEEE ITCC 2004 Track on Mordern Grid and Web Systems.

[WFP+05] Minjun Wang, Geoffrey Fox and Marlon Pierce. *Grid-based Collaboration in Interactive Data Language Applications*. To appear in Proceedings of IEEE International Conference on Information Technology April 11-13, 2005, Las Vegas.

[WSCA] Heather Kreger et al. *Web Services Conceptual Architecture (WSCA 1.0)*. IBM Technical White Paper. May 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[WSEVENT] Box et al., Web Services Eventing (WS-Eventing) is a standard that enables Web Services to be interoperable through publish/subscribe for event

notification messages. http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-eventing.asp [WSRP] 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 [WSFL] Frank Leymann et al. *Web Services Flow Language (WSFL 1.0)*. IBM Technical White Paper. May 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[WSDL] W3C. *Web Services Description Language (WSDL 1.1)*. W3C. http://www.w3.org/TR/wsdl

[WS-MANAGEMENT] Sun Microsystems. *Web Services Management (WS-Management)*. It is a Web Services specification for systems management. October 2004. Sun Microsystems. http://developers.sun.com/techtopics/webservices/management/WS-Management.pdf

[WSNOTIFICATION] IBM, *Publish-Subscribe Notification for Web Services*, IBM, available at http://www-106.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf

[WSRF] OASIS, Web Services Resource Framework (WSRF) TC is open standard for modeling and accessing stateful resources using Web services. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[WUBF] Wenjun Wu, Ahmet Uyar, Hasan Bulut, Geoffrey Fox. *Integration of SIP VoIP and Messaging with the AccessGrid and H.323 Systems*. Proceedings of 1st International Conference on Web Services. Las Vegas, June 2003.

[WWW 1991] A Little History About World Wide Web at http://www.w3.org/History.html or http://www.w3.org/WWW/

[XLINK] W3C, XML Link Language (XLink) Version 1.0 uses XML syntax to create and define generic links to indicate internal or external resources. http://www.w3.org/TR/xlink/

[XML] W3C, XML (Extensible Markup Language) version 1.0 is a W3C Recommendation for creating special-purpose markup languages. It has become the standard way to define interoperable data structures. http://www.w3.org/TR/REC-xml/

[X.Qiu] Xiaohong Qiu, *Building Desktop Application with Web Services in a Message-based MVC Paradigm*, IEEE 2nd International Conference on Web Services (ICWS 2004), San Diego July 2004. http://grids.ucs.indiana.edu/ptliupages/publications/ICWS04_BuildingDesktopApplicaitonwithWebServicesinaMessageBasedMVCParadigm.pdf

[YBSJRRM] Naveen Yajaman, Josh Brown, Shanmugam Subramaniam, Tony John, Narsimha Reddy, Venkataraman R and Andrew Mason. *Web Service Façade for Legacy Applications*. June 2003. Microsoft technical article. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/wsfacadelegacyapp.asp

[Yokote+Tokoro] Yasuhiko Yokote and Mario Tokoro. *The design and implementation of Concurrent Smalltalk*. ACM SIGPLAN Notices, Conference proceedings on Object-oriented programming systems, languages and applications. Volume 21 Issue 11. June 1986.

[Y. Shan] Yen-Ping Shan. *MoDE: AUIMS for Smalltalk*. ACM SIGPLAN Notices, Proceedings of the European conference on object-oriented programming systems, languages, and applications. Volume 25, Issue 10.

[Zaslavsky+Memon] IIya Zaslavsky and Ashraf Memon. *Web services for generating SVG-Tiny maps on mobile phones*. Proceedings of SVG Open Conference, Tokyo Japan September 2004. http://www.svgopen.org/2004/papers/WebServicesForSVGTinyMapsOnMobile/

[ZM04] Iiya Zaslavsky and Ashraf Memon, *Web Services for Generating SVG Tiny Maps on Mobile Phones*, Proceedings of SVGOpen 2004, Tokyo, Japan. http://www.svgopen.org/2004/papers/WebServicesForSVGTinyMapsOnMobile/