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

Xiaohong Qiu[1, 2]

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

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

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

1. Introduction

Web Services are becoming an increasingly important feature of Internet and Grid systems. They support a loosely coupled service oriented architecture that builds on previous distributed object architectures like CORBA, Java RMI, and COM to provide scalable interoperable systems. The broad applicability of this approach includes enterprise software, e-Science and e-Business. Correspondingly there are a growing number of powerful tools that are available for building, maintaining and accessing Web Service-based systems. These tools include portals that allow user frontends to Web Services. This model for user interaction has new standards like portlets with WSRP (Web Services for Remote Portlets) and the Java Specification Request JSR168 supporting lightweight interfaces to the backend resources. This architecture shown in fig.1 implements the Model-View-Controller or MVC [1] architecture with a clean message based interface partially specified by the portlet standard.

We have a general approach of building Web applications centered around messages. The key challenge is to exploit this concept in design and implementation to provide scalable interoperable systems. Specifically we investigate a universal modular design with messaging linkage service model that converge desktop applications, Web applications, and Internet collaboration to achieve reusability, scalability, interoperability and pervasive accessibility. We have systematically carried out our work by proposing an "*explicit message-based MVC*" paradigm (MMVC) as the general architecture for Web applications [2]; presenting an approach of building "*collaboration as a Web Service*" by decomposition of MMVC in a three-stage pipeline architecture for three collaboration types ─ monolithic, thin client and interactive client [3]; bridging the gap between desktop and Web application by leveraging the existing desktop application with a Web service interface through "*MMVC in a publish/subscribe scheme*", which is the focus of this paper; identifying some key issues that influence message-based Web applications with a detailed analysis of performance in a presentational style application experiment with rich Web content and high client interactivity [4]. Elsewhere we discuss SIMD and MIMD collaboration as the general architecture of "*collaboration as a Web service*" model [5].

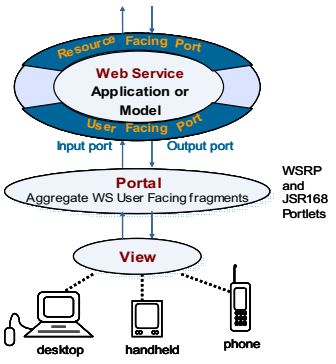

Figure1 Portlet Approach to Web Services and their user interfaces

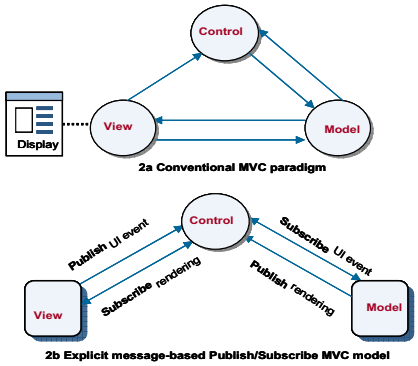

Figure 2 MVC in Publish/Subscribe

The general MVC approach of fig. 2a) is a well-established paradigm, which has been used for many years. As we describe later in more detail, traditional MVC applications employ method-based interactions between the components with this approach giving the needed high performance for interactive applications. In this paper, we explore "*explicit message-based MVC*" (MMVC) ─ a different approach with MVC being used systematically but with message based interactions (shown in fig. 2b), between the model and the view components. We suggest that modern computers and networks are fast enough that this approach will give adequate performance for desktop applications such as those in the Microsoft Office Suite. We embody this idea as the "*MVC rule of the Millisecond*" [6]. This asserts that message based interactions between "nearby" components have an intrinsic delay of a few milliseconds and so this linkage approach is possible whenever such a delay is acceptable. Simple non-optimized Java messaging gives such a delay whenever the components are either on the same computer or on machines with a local area connection.

In this paper, we explore this area and make two contributions. Firstly we look at existing method-based MVC application – the Batik SVG browser from Apache [7] – and convert it into a message-based approach as contrasted in figures 2a) and 2b). We discuss some of the issues that came up in this conversion. Secondly we use this message-based version of SVG to explore the overhead in the message-based approach. We find it represents about a 20% overhead for "*model*" and "*view*" distributed in different sites on Indiana University's Bloomington campus and the user does not distinguish the interactive experience in switching from method to message-based interactions.

2. Technology Background: DOM and SVG

W3C Document Object Model (DOM) [8] defines "*a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents*" and a generic event model, which implemented by version 5 browsers (e.g. Mozilla and Internet Explorer). DOM can reflect structure of Meta data abstracted by XML schema. DOM also allows any language bindings; therefore it can be used by variety of applications. Scalable Vector Graphics (SVG) [9], as defined by W3C, is "*a language for describing two-dimensional graphics and graphical applications in XML*". SVG is an example of DOM application. Compared with HTML content, SVG has richer web graphics flavor with additional features inherited from XML and DOM which makes it a unique technology that has been widely used by graphical authoring tools, GIS system and data visualization. Apache Batik SVG browser is a stand-alone client application, which is written in Java apart from a few native classes; there are also customized SVG implementations for handheld devices.

3 Message-based MVC and SVG

3.1 Introduction

We choose an existing system ─ Batik SVG browser [7], and modify its architecture from a method-based desktop application to a message-based one. This has several implications. The message-based architecture allows one to build desktop applications as web services and so unify traditional desktop and web service plus portal approaches. This unification makes collaborative applications straightforward to build as described in section 4. Further the separation of *model* and *view* makes it easier to support diverse client devices and operating systems. This could be significant with the growing interest in PDA and Linux clients. Note our strategy allows "long distance" linkage between the "*model*" (business logic of application) and view as well as their cooperation on local networks as within a campus. However transcontinental latencies are hundreds to thousands of milliseconds and so this cannot be used for interactive experience. As we describe later, we will use the same messaging infrastructure that has been used to support large Grid applications. This unification of Grid and client applications into a single message-based architecture is key to our paper. We can use our approach for interactive applications when model and view are nearby

and allow collaboration and traditional Web portal use for remote access.

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

Secondly, our approach allows building collaborative SVG as a special case of our general Collaboration as Web Service architecture. This work has been discussed in our earlier papers in Internet Computing 2003 [2] and SVG Open 2003 [3]. We presented a multiplayer chess game as a test case of our collaborative SVG infrastructure without decomposition of the client application.

Thirdly, we discuss the conversion of a client application into a distributed system and identify the difference in design principles. This approach allows maximum reusability of existing components in Web application deployment and also unifies desktop and Web applications architecture in a message-based service model. MVC is a frequently used paradigm in modern architecture design (e.g. Microsoft Windows). In a "conventional" MVC, "*controller*" executes its tasks through method calls since messages are hidden in system level. We make a critical observation, namely "conventional" MVC has to be replaced by an "*explicit message-based*" MVC in order to enable components of the application to be distributed. In our approach, we use "explicit control messages" to abstract the semantic meanings of "*controller*" so that messages of the original system are exposed and 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 network.

c) Our approach requires us to support our model view linkage with a high performance messaging middleware infrastructure. We use NaradaBrokering [10], which has been separately developed and provides a variety of publish/subscribe models including peer-to-peer and Java Message Service (JMS) emulation. Our use for collaborative SVG would exploit these latter Grid messaging capabilities of NaradaBrokering. The changes bring up issues that cause a challenge to the system:

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

3.2 Message-based Event model

The basic idea is illustrated in fig. 3. Traditional event-based programming is used extensively in the Batik SVG browser and most modern applications. Different parts of a program are linked asynchronously with one part producing events that are passed to listeners whose callback method has been passed to the producer. As shown in fig. 3b) this can also be implemented with explicit messages where listeners subscribe

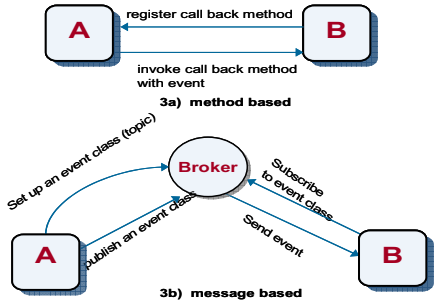

Figure 3 Message or method based Publish/Subscribe

to an event class (topic) and events producers publish them to this topic. Our strategy is to replace the listener model of fig. 3a) by the publish/subscribe broker model of fig. 3b). Note that either approach can use explicit queues (maintained on a broker in the message case) or alternatively integrate the broker into the producer as in most simple method-based event models. 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 MMVC architecture is shown in figure 4. 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 more restructuring of the existing software.

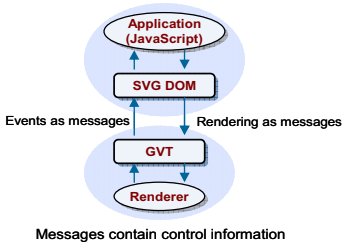

Figure 4 SVG browser derived from message-based MVC

4. Performance

We have started an extensive series of performance measurements to demonstrate the viability of our approach. The main issues are the algorithmic change from breaking the code into two and the two-way transit time is only 20% of the total processing time in the case of local brokers. We separately measured the overhead in NaradaBrokering itself which consisting of forming message objects, serialization and network transit time with four hops. This overhead is 5-15 milliseconds depending on the operating mode of the Broker in simple stand-alone measurements. The contribution of NaradaBrokering to the overhead is larger than this (about 30 milliseconds in preliminary measurements) due to thread scheduling overhead interfacing with the complex SVG application. We will discuss this and optimizations to the system performance in future papers [4].

5. Conclusions

We believe our prototype shows 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. Our approach suggests that one need not develop special "collaborative" applications. Rather any application developed as a Web service can be made collaborative using the tools and architectural principles discussed in this paper. Note that Moore's law implies that computer performance will continue to improve while networks will also continue to increase in bandwidth with however latency for long distance linkage remaining higher than that needed for interactive use. Thus inevitable infrastructure improvements will tend to make our approach more attractive in the future. These ideas can also suggest a uniform approach to user interface design with desktop and web applications sharing a common portlet (WSRP, JSR168)-based architecture.

This could motivate the development of new desktop applications with many capabilities not present in today's systems such as OpenOffice and Microsoft Office. Other research in our laboratory is looking at extending our ideas to OpenOffice while a limited implementation is possible using the rather crude event interface exposed for PowerPoint. These ideas can unify PDA and desktop, as well as Linux, MacOS, Windows and PalmOS applications.

References

1) G. Lee, *Object oriented GUI application development*. Prentice Hall, 1994. ISBN: 0-13-363086-2.
2) Xiaohong Qiu et. al. *Internet Collaboration using the W3C Document Object Model* in Proceedings of the 2003 International Conference on Internet Computing, Las Vegas June 2003.
3) Xiaohong Qiu et. al. *Collaborative SVG as A Web Service* in Proceedings of SVG Open, Vancouver, Canada, July 2003.
4) Xiaohong Qiu, Shrideep Pallickara, and Ahmet Uyar *Making SVG a Web Service in a Message-based MVC Architecture* submitted manuscript for SVG Open Conferences, Tokyo, Japan, September 2004.
5) Xiaohong Qiu and Anumit Jooloor, *Message-based Web Services Architecture for e-Learning* to appear in International Conference on Education and Information Systems: Technologies and Applications, Orlando, Florida, July 2004.
6) Geoffrey C. Fox, *Software Development around a Millisecond* in CISE Magazine.
7) Apache Batik SVG Toolkit.
8) W3C Document Object Model (DOM) Level 2 Core Specification.
9) W3C Scalable Vector Graphics (SVG) version 1.0 Specification.
10) Community Grids Lab NaradaBrokering system.