Java Thin Clients Revisited: An Architecture for Responsive, Live Wireless Applications

Dragos A. Manolescu and George F. Santamarina
Applied Reasoning
10955 Lowell Ave., Suite 500
Overland Park, KS 66210, USA
{dmanolescu,gsanta}@appliedreasoning.com

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems; D.2.11 [**Software**]: Software Engineering—*Software Architectures*

General Terms

Design

Keywords

Mobile Classic Blend, thin client, thick client, wireless, WAP, HTML, PDA, PalmOS, J9 VM

1. INTRODUCTION

The current programming model of the Web revolves around application servers [5]. The application server architecture offloads most processing to a server, which hosts the application and prefabricates views (e.g., HTML pages) for the Web browser. Clients render HTML pages and transmit to the server events corresponding to user actions, such as pressing a button on an HTML form. They also request new pages from the server and display them.

2. ENVIRONMENT CONSTRAINTS

A successful application for personal wireless devices must deal with an array of constraints. The first set of constraints pertains to pocket-sized devices in general: limited processing power and memory space [6]; reduced screen real estate and limited input capabilities [7]. Another set of constraints pertains to wireless devices: limited bandwidth;[1] disconnected operation; expensive airtime. While the constraints specific to small devices have been dealt with in the past, Web-enabled mobile phones are bringing the second set of issues under the spotlight.

3. WHAT ABOUT WAP?

The above constraints make building Internet applications for mobile phones a hard problem. Several technologies attempt to address these challenges. For example, the Wireless Application Protocol (WAP) provides a suite of markup languages specifically designed for the microbrowsers that run on wireless devices. WAP version 1 introduced WML, the Wireless Markup Language. The newly adopted version 2 adds support for the XHTML Mobile Profile markup language [2]. A scripting language called WMLScript lets applications leverage some of the processing capabilities available on the client side.

However, despite the new features of WAP 2.0 (including support for server push via push proxies), these technologies have an intrinsic flaw as they parallel the current programming model of the Web too closely. For example, WAP involves rendering complete pages, and the live update capabilities are nonexisting or cumbersome. Additionally, users have been reluctant to adopting WAP. So are there any other alternatives for wireless application developers?

4. WHY THIN CLIENT?

We have designed an architecture that provides developers the benefits of server-side programming without the limitations of HTML and WML rendering and lack of live update capabilities. Some researchers have argued that the thick client model will be prevalent on personal wireless devices [8]. We disagree. Instead, we have adopted a thin-client architecture. The thick-client model has fundamental problems that the thin-client avoids. Here are some of the benefits of the thin client model:

Lower time to market and development costs: Building software for personal devices (PDAs) is not the same as building software for desktop computers. Client-side development requires specialized programming languages [9], programming styles [6], and development tools [3, 1]. In contrast, our thin client approach can *minimize and even eliminate client-side development* as it places the application on the server.

Lower bug count at delivery: The thick client approach involves client-server programming. This poses more challenges to developers than writing all the code in one place. For example, the thick client-server involves distributed functionality, which brings into the equation multiple computers, networks, and data sharing and synchronization. These factors increase the operational complexity of client-server software. The additional complexity translates into higher defect potentials. Capers Jones has found that the defect delivery level of client-server applications is almost two times larger than for stand-alone applications [4]. In contrast, the simpler programming model of our thin client *translates into fewer defects* than thick clients.

Simpler programming model: The thick client approach requires designing custom protocols and APIs for each application.

[1]Even in a 3G wide area network the wireless bandwidth remains well below the wired bandwidth, and it will remain that way for the forseeable future.

Copyright is held by the author/owner.
OOPSLA'02, November 4-8, 2002, Seattle, Washington, USA.
ACM 1-58113-626-9/02/0011.

When one side sends a message to the other, there are a wide variety of responses that can happen. This is error prone. In contrast, our thin client approach *involves programming to a relatively fixed, widget API*. This API changes only when the device adds support for new widgets. Once debugged, developers can build applications without worrying about bugs outside the application code.

Better integration with existing infrastructure: Corporations have invested in wireline technology for their Web-enabled applications. Thick clients require new applications, leading to limited reuse and integration problems. In contrast, our thin client approach *promotes reuse and facilitates integration* with current infrastructure as it leverages the server side of corporate applications.

Lower administration costs: Software changes in time. Developers add new functionality or fix security holes. Updating thick client software generally requires user involvement and poses security risks. In contrast, our thin client approach allows developers to *deploy new applications* as well as *update existing applications seamlessly* as applications reside in a single location.

Improved application and data security: Some people suggest that in the future users may replace their general-purpose computers with mobile systems [8]. In fact, this depends on the type of data they manipulate. Corporations frown on their employees carrying sensitive enterprise data on mobile devices. In contrast, our thin client approach poses *lower security risks* because it only contains remote GUIs instead of storing enterprise data and applications on the device.

However, the idiosyncrasies of wireless connectivity and the limitations of technology employed by current wireless devices make building wireless thin clients a hard problem.

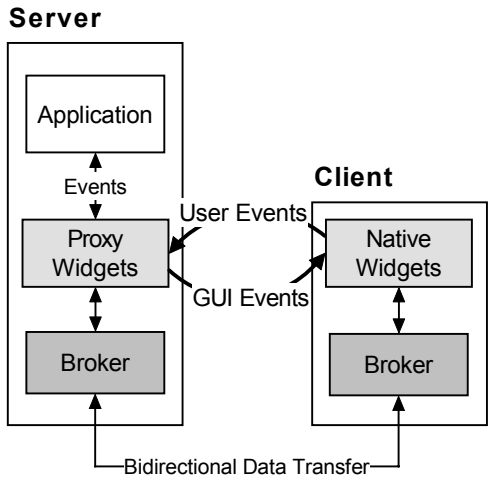

Figure 1: The Mobile Classic Blend Architecture

5. MOBILE CLASSIC BLEND

Mobile Classic Blend (MCB) is a unique patent-pending technology aimed at developers building thin-client applications for Java-powered personal wireless devices. MCB represents a radical departure from the increasingly popular way of building thick clients on portable devices supporting Java 2, Micro Edition (J2ME).

In this demonstration we show how MCB addresses device independent programming; software development under the constraints specific to current wireless technology (devices and connectivity); client- and server-side extensibility; responsiveness of networked applications in constrained environment; and server-side push in a 2G-world. Our approach leverages object-oriented frameworks, patterns, object serialization, message sending, and current state of the art in object technology to achieve high performance on a constrained platform.

We present a live, server-centric application running on a PalmOS-powered phone, explain its design, and show what it takes to build MCB applications. We've built the MCB architecture from the ground up for live, server-centric applications. The architecture revolves around high-performance ORBs and a communication protocol optimized for low-bandwidth and medium-latency networks (i.e., less than 600 milliseconds round trip time)—see Figure 1. MCB applications support transparent software updates, server-side push, wake up, and auto connect over wireless connections.

MCB is a version of Classic Blend (CB) that leverages IBM's J9 VM and PalmOS on the client, and J2SE/J2EE on the server. MCB accommodates the constraints of current technologies by using on the client techniques commonly found in high-performance enterprise servers, as well as techniques specific to small memory systems.

6. REFERENCES

[1] Borland Software Corporation. jBuilder MobileSet. On the Web at http://www.borland.com/jbuilder/mobileset/.

[2] Wireless Application Protocol Forum. Wap 2.0 technical white paper. Available on the Web from `http://www.wapforum.org`.

[3] IBM. VisualAge Micro Edition. On the Web at http://www.embedded.oti.com/.

[4] Capers Jones. *Applied Software Measurement*. McGraw-Hill, New York, NY, second edition, 1997.

[5] Dragos A. Manolescu and Adrian E. Kunzle. Several patterns for ebusiness applications. In *Proc. 8th Conference on Pattern Languages of Programs*, Monticello, IL, September 2001. The Hillside Group, Inc. Available on the Web from `http://micro-workflow.com/PDF/ebp.pdf`.

[6] James Noble and Charles Weir. *Small Memory Software: Patterns for Systems with Limited Memory*. Software Patterns Series. Addison-Wesley, 2000.

[7] James Noble and Charles Weir. A window in your pocket: Some small patterns for user interfaces. In *Proc. European Pattern Languages of Programs*. The Hillside Group, Inc., 2001.

[8] Thad Starner. Thick clients for personal wireless devices. *IEEE Computer*, 35(1):133–135, January 2002.

[9] Java 2 platform, micro edition. On the Web at http://java.sun.com/j2me/.