An Alternative to Thick Clients for Personal Wireless Devices

Dragos A. Manolescu
Electrical Engineering and Computer Science
415 Snow Hall,
University of Kansas,
Lawrence, KS 66045
dragos.manolescu@acm.org

Draft of February 20, 2002

1 Introduction

Thad Starner's article in the January 2002 issue of IEEE Computer focuses on improving the functionality of software for personal wireless devices [12]. Starner proposes a shift from thin to thick clients as an easy route to achieving this goal. He argues that if the technology trends continue to evolve as they have done throughout the last decade, wireless devices will eventually have disk storage, RAM, and CPU power comparable with primary general-purpose computing devices.

While technology will undoubtedly continue to advance, Starner's analysis doesn't take into account important factors like software development costs, integration with existing infrastructure, application deployment, security, and total cost of ownership. In this article I revisit the topic and examine the characteristics of thin and thick clients in the context of personal wireless devices. I argue that the thick client approach won't cause thin clients to go extinct. Neither approach is perfect, and I'll point out several characteristics of thin clients that make them more attractive than thick clients. As an example, I present an existing software architecture that exploits the characteristics of the thin client approach while dealing with the constraints imposed by current wireless technology.

If you're planning to wait for technology to replace thin clients with thick clients as Starner suggests, this paper may be useful to you. It will show you some of the benefits of the thin client approach that you will have to give up, and you may want to reconsider. In contrast, if you can't wait for technology, you should read this paper. The software architecture described here lets you use and benefit from the thin client approach on current personal wireless devices.

2 Background

Thin and thick clients are different approaches of dealing with the partitioning of functionality between the client and the server. The server side provides services and "serves" requests from clients. A thin client relies on the functionality of the server; a thick client handles much of the functionality itself.

The thin client approach dates back from the days of mainframes. In those days hardware was locked behind doors in air-conditioned rooms and were accessed via "dumb" terminals. All processing took place on the mainframe, and the terminals simply handled paper (and later on CRT) output and keyboard input. Because so little processing was carried out by these terminals, they were called *thin clients*.

The personal computer (PC) era offered new possibilities. The CPUs, memory, and storage capabilities of PCs enabled them to run their own programs, independently from the server. As PCs' processing capabilities

increased, software moved from the server to the client, thus creating thicker clients. In the thick client environment, the bulk of the software runs on the client and interacts with dedicated servers only for special services, e.g., enterprise-wide applications and databases.

3 Thin Clients for Personal Wireless Devices

Affordable and powerful personal computing convinced some people that thin clients were gone. This was a mistake. The Web has resurrected the thin client approach. Server-side programming is back in style. It could be argued that a Web browser application is closer to the 3270 architecture than to a fat client. The Web is making inroads in the wireless market. However, as long as the Web remains server-centric, our personal wireless devices are going to act like 21st century 3270s.

After decades of experience with thick clients we are now in a better position to evaluate the two approaches. While the thick client approach has opened new doors, it has fundamental problems that the thin client avoids. We cannot dismiss the thin client approach and wait for technology to catch up. Instead, we should rethink the architecture to incorporate the lessons learned from both approaches.

Let's consider the following characteristics:

Development cost Building software for personal devices (PDAs) is not the same as building software for desktop computers. The differences include programming languages [13], the programming style [8], and the development tools [5, 3]. This translates into developer retraining and retooling—an expensive proposition. In contrast, the thin client approach can *minimize client-side development* as it places the application on the server.

Integration with existing infrastructure Corporations have invested large amounts of money in wireline technology for their Web-enabled applications. Acquiring new infrastructure to add support for wireless devices represents an expensive endeavor. Businesses would rather amortize their current technology investments and gradually expand their market to wireless customers. Thick clients require new applications, causing limited reuse and integration problems. In contrast, thin clients *promote reuse and facilitate integration* as they reuse the server side of applications.

Instantaneous deployment 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, the thin client approach allows developers to *deploy new applications as well as update existing applications instantaneously* as applications reside in a single location.

Improved security Starner suggests that in the future users may replace their general-purpose computers with mobile systems. In fact, this depends on the type of data. Corporations won't allow their employees to carry sensitive enterprise data on mobile devices. Instead, by maintaining this data on the enterprise servers, the corporation has direct control over regulating and monitoring the access. Should the device fall into the wrong hands, the liability is higher if it contains full-fledged applications or confidential records. Even with encryption, the chances of it being broken while in the possession of a cracker are much higher [11]. In contrast, thin clients pose *lower security risks* because they only contain remote GUIs instead of enterprise data and applications.

Lower total cost of ownership (TCO) Typically software licenses charge a per-CPU fee. Thick clients require a license for each client. For large installations this may yield an expensive bottom line. In contrast, the thin client approach *reduces the TCO* as users can share a single server-side applications. Note, however, that this is bound to change as companies adapt the licensing models to the architectural trend.

The above characteristics show that the thin client approach has several advantages over the thick client approach. In time (i.e., once personal devices with processing power and storage capabilities comparable with general purpose computers become viable) the development cost for personal devices will eventually drop, and the integration with existing infrastructure will be available out of the box. However, the thin client approach will continue to provide instantaneous deployment, improved security, and lower TCO even after thick clients become technologically viable.

The key factor in choosing between a thick or thin client lies in striking the right balance between the type of processing that takes place on the client, and the type of processing that takes place on the server. In the context of personal wireless devices additional challenges stem from the peculiarities of wireless connectivity and the limitations of current technology. Next I present a software architecture designed for developing thin clients on current wireless platforms.

4 A Software Architecture for Wireless Thin Clients

A successful thin client for personal wireless applications must deal with an array of constraints. A first set of constraints pertains to pocket-sized devices in general: limited processing power and memory space [8]; reduced screen real estate and limited input capabilities [9]. Another set of constraints pertains to wireless devices: limited bandwidth; 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.

These constraints make building Internet applications for mobile phones a challenging task. 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 [4]. 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 (which include support for server push via push proxies), these technologies have an intrinsic flaw: they parallel too closely the current programming model of the Web, which I sketch in the next section.

4.1 Application Server

The programming model of the Web revolves around application servers [7]. This architecture borrows heavily from the early days of the thin client approach. As such it offloads most processing to the server, which hosts the application and prefabricates views (i.e., 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. Thus, the server carries out all processing, including the client-specific page generation.

Figure 1 shows a sketch of the application server architecture. On the server side, the arrows between the application and the Web server show the amount of data flowing in each direction. The application receives requests (low volume) and generates complete HTML pages in response (high volume). The dashed arrow from the server to the client shows the data flow triggered by clients' requests. Since HTTP is a unidirectional protocol, the server can move data to the client only in response to a client-initiated request.

Several characteristics of this architecture make it unsuitable for personal wireless devices.

- The application server architecture takes little advantage of the client's processing capabilities. The server-side handles everything, from updating the application's state through generating HTML. Additionally, the application server must maintain knowledge about the markup language recognized by microbrowsers running on the wireless clients (e.g., WML or one of its successors), as well as for the large number of existing thin clients on popular browsers.

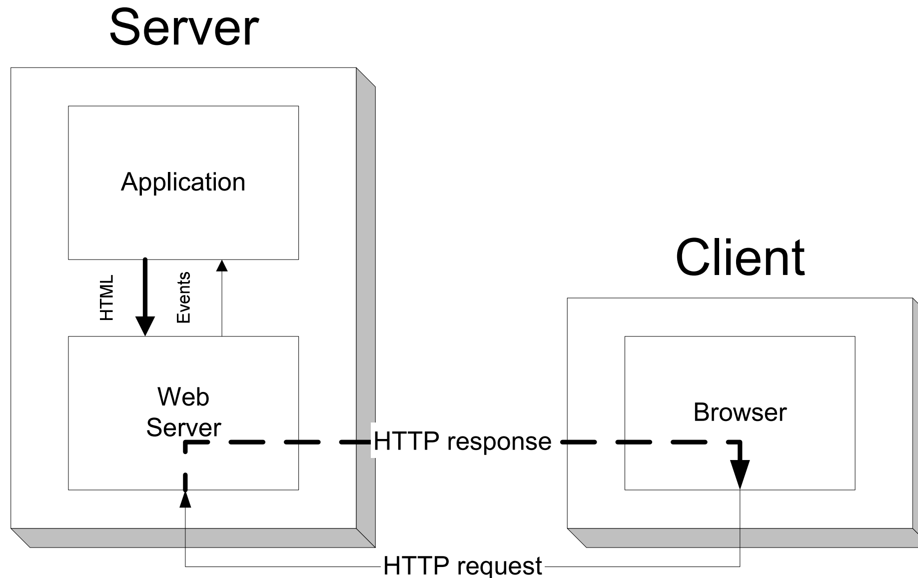

Figure 1: Application server, the architecture widely used on the Web; the thickness of the arrow corresponds to the amount of traffic

- Refreshes involve redrawing complete pages. While wireline solutions typically have room for the high volume traffic between the server and the client, such wasted bandwidth is not viable on wireless networks.
- Finally, the communication model involves client pull. The server can't push information to its clients, preventing real time display of data. Instead, clients must periodically *poll* the server for updates, a non-scalable approach that greatly increases network traffic and doesn't eliminate the delay.

Note that all current technologies like Java Server Pages (JSP), Active Server Pages (ASP), etc. have these limitations. While people have learned to live with these problems (a sign of this acceptance is the large number of Web users), in the context of personal wireless devices these limitations become evident very quickly. Note that thick clients can avoid these problems. A successful solution for personal wireless devices must provide a radical departure from thin client application server architecture.

4.2 Presentation Server

The presentation server architecture tilts the balance towards the client side. The server-based application logic is still at the focal point of the architecture. However, instead of sending to the client complete HTML pages, the presentation server is event-driven.

Figure 2 sketches the presentation server architecture. On the server side the application employs a widget-based GUI, a widely-used programming model for graphical environments like X, Windows, and MacOS. This programming model lets developers build wireless applications with as little regard as possible for the client's idiosyncrasies and limitations. Note, however, that although both the presentation server and the X Window System use events, the former operates at a higher level than the latter. X was built with the mechanism policy split as one of its fundamental design principles [10]. Consequently, (remote) X applications must provide implementations for all widgets they use. In contrast, the presentation server uses native widgets, which are already implemented on the device. This distinction translates into lower bandwidth requirements for the presentation server.

The server-side proxy widgets have device-specific corresponding widgets on the client side. Thus, GUI events replace the HTML pages typical for application servers. The client uses these events to update the appropriate widgets. In response to user actions on the GUI, the client sends the corresponding events back to the server. Therefore, the widget-based approach eliminates the need for full page redraws specific of HTML-based presentations. This dramatically reduces the traffic between the server and the client, an important issue for bandwidth-limited wireless devices. Additionally, the use of native device widgets makes thin client applications have the device-specific look and feel, which represents an important usability factor.

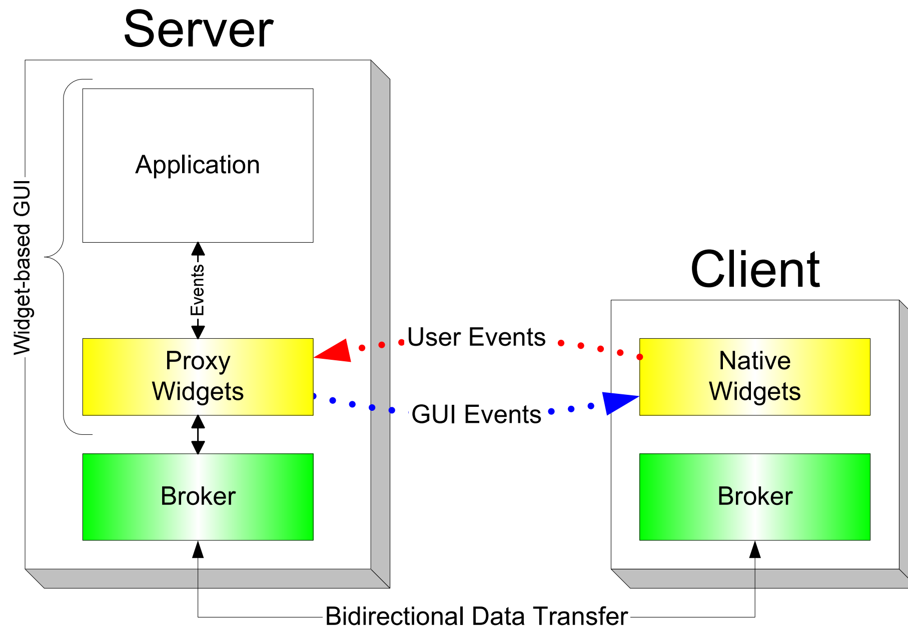

Figure 2: Presentation server, an architecture for smart thin clients

Due to its unidirectional nature, HTTP can't accommodate the flow of user events from the client to the server and of GUI events from the server back to the client. Consequently, the presentation server architecture must use a bidirectional data transfer protocol optimized for low-bandwidth connections. The bidirectional protocol also solves the problem of client pull: the server can push data to the client, making instantaneous notification possible.

The presentation server architecture uses brokers to manage the client-server communication. These provide network transparency to the layers above them. The server side broker sends the events from the proxy widgets to the client, and updates them in response to changes on the client. Presentation server applications on the same device share the client side broker. This broker dispatches the incoming events to the appropriate native widgets, and transmits the outgoing events to the server. The resource sharing on the client enhances modifiability and portability and facilitates building new applications [2].

Naturally, the above characteristics have a cost. The presentation server architecture trades simplicity for functionality. Overall, the client in the presentation server architecture (Figure 2) does more processing than the client in the application server architecture (Figure 1). Therefore, the viability of this approach depends on dealing with the challenges associated with the increase in complexity. I discuss these challenges in Section 6. First I examine some of the most important technology and business issues associated with the presentation server architecture.

5 Technology and Business Issues

What makes a thin client approach attractive? The key factor is that it works with current technology (i.e., personal devices and wireless networks). Additional factors include integration with existing infrastructure, live notifications and location-based services, simpler programming model, and slow-changing API. Here's how these factors line up.

Compatibility with current wireless technology Due to its server-centric model, thin-clients typically involve more client-server traffic than thick clients. This traffic is subject to the bandwidth limitations specific to wireless devices. Current wireless networks (i.e., 2G) have bandwidth in the range of 9.6 through 19.2 Kbps. To complete the picture, add high latency to the mix. The presentation server addresses these constraints in several ways. First, it uses widgets driven by events. This eliminates full-page refreshes, thus reducing the data traffic between the server and its clients. Second, it uses a protocol designed for the ground up for wireless connectivity. Optimizations at the protocol level reduce the bandwidth requirements even further. Finally, message batching helps it deal with the high latency. This design makes the architecture viable with current wireless technology. In contrast, Starner's article doesn't provide an immediate solution; his suggestion to move to thick clients requires waiting for the technology to become available.

Real time, wireless servers The combination of thin clients and the presentation server architecture doesn't aim at replacing wireline access. Instead, it augments it when end users need wireless, real time connectivity. The integration with existing infrastructure lets users quickly transform existing J2EE servers into real time wireless servers. Additionally, general-purpose computers acting as thin clients and hooked up to wireline networks could also use the presentation server architecture. This makes possible live, responsive Internet applications on a wide range of thin clients, regardless of how they are connected to the network.

Application wake-up and server push People with personal wireless devices need them because they can benefit from being mobile and connected. Sometimes they need to know promptly whenever time-sensitive information changes, and act upon it. The bidirectional data transfer protocol make this possible. Additionally, carrier-supported messaging can notify devices in real time, even when they are disconnected from the Internet. The wireless device wakes up and requests a connection to the server in response to the notification. Once the connection is established, the server can push data to the device. In effect, this mechanism gives the illusion of always-on applications and allows for location-based services. Additionally, since messaging costs less than regular airtime, using the messaging service significantly lowers the operational costs.

Simpler programming model The presentation server architecture reduces client side programming to composing widgets. Application developers don't write any code on the device. Instead, they build software exclusively on the server, with the tools and within the development environments they already know. In contrast, the thick client involves client-server programming. This poses more challenges to developers than writing all the code in one place. For example, client-server involves distribution, 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 [6]. The simpler programming model of the presentation server translates in fewer defects compared to thick clients. Additionally, developers reuse existing resources, knowledge, tools, and development environments.

Narrow, slow-changing API Presentation server involves programming to a widget API. This is a relatively fixed API; it changes only when the device adds support for new widgets. Once debugged, developers can build applications without worrying about bugs outside the application code. In contrast, the thick client approach requires designing custom protocols and APIs for each application. When one side sends a message to the other, there are a wide variety of responses that can happen. This is more error prone. Additionally, optimizing each debugged application is also a hard and time-consuming problem.

6 Cost and Challenges

The presentation server architecture trades simplicity for functionality. In Section 4.2 I've already mentioned the brokers and the bidirectional data transfer protocol. Here are the factors that contribute to the increased complexity.

Brokers On the client side, the presentation server replaces the microbrowser (spoon fed WML pages by the server) with a broker. The broker must implement the bidirectional data transfer protocol, and to provide services like serialization, application data management, and messaging. Therefore, adopting this architecture involves designing, building, and debugging the broker for the server, as well as for the clients. Additionally, given the limited processing capabilities of current portable devices, the client-side broker typically requires optimizations for speed, space, and resource consumption. This is a challenging endeavor; however, developers have to tackle it only once. Once available the brokers act as shared resources that application developers use it solely through its API.

Communication protocol Adopting the presentation server architecture also involves designing, implementing, optimizing, and testing the bidirectional data transfer protocol. Generally network protocol design is a hard problem. In the context of personal wireless devices, the communication spans over wired as well as wireless networks. This makes protocol design even a harder problem. The designers have to address a wide range of issues, including reliability, scalability, recovery, latency, asynchronous messaging, limited bandwidth, and dynamically changing connectivity. Additionally, they're faced with important design decisions: should the protocol guarantee message delivery? Should the messages return values? Should the protocol include support for asynchronous messages? These are all hard questions, and finding the right answer requires a great deal of expertise. However, similar to the challenges of implementing the brokers, providing the communication protocol is a one time deal. Once available, the brokers use it transparently for the application programmers.

Arcane platforms The designers of personal devices trade form-factor and convenience for power. Consequently, current devices are arcane. They impose various sorts of limitations that many developers haven't had to deal with from the 1980s. Several factors make writing software for these devices a challenging endeavor. First, software development involves stripped-down versions of general-purpose programming languages. Often times this is a subset of C and an increasing number of devices support J2ME, the micro edition of the Java Platform [13]. Second, the current personal devices are fragile systems. In spite of the availability of high level languages like Java, application developers have to deal directly with concerns like resource management, platform specific types of database records, etc. They have to ensure that their application doesn't corrupt other applications or the operating system. Finally, in addition to the optimizations for speed and bandwidth, the developers also have to optimize for resource usage like stack size (e.g., 2K on the Palm). Imagine implementing a multi-threaded protocol with batching and support for serialization, and fit it into a 2K stack! Dealing with these limitations involves techniques that most developers are not familiar with. For example, a small stack size may involve converting recursive algorithms (like the graph traversal required for serialization) into iterative ones to avoid stack size limitations. The presentation server architecture removes the arcane platform from the picture

in that once the client-side broker is available, application development takes place solely on the server. The architecture reduces client side work to build the widgets, a task that doesn't require any programming.

Disconnected operation Wireless network access means intermittent connectivity. Unlike in the wired world, applications running on personal wireless devices must tolerate the temporary loss of connectivity. However, the loss of connectivity must remain transparent for the user, who should still be able to use the application. The key idea is to use the thin client approach as a technique when applications need a server. As I've discussed in the previous sections, thin clients lower the number of programming errors and let application leverage optimized server code—which is particularly important in the context of wireless personal devices. However, when applications don't need remote services, they should act as stand-alone. For example, consider a mail agent. Sending and receiving email depend on it being connected to a server. For these operations, the mail agent is a thin client. However, composing messages doesn't require remote services. As such, it should act as a stand alone application and queue the messages locally until the connection is reestablished, when it starts acting as thin client. Note that this may require proxies for additional resources like mail records, etc.

Solving the above problems facilitates the work of developers building applications for personal wireless devices, providing them with a reusable tool on the client side. Application developers write software in a single place—the server—using a familiar programming model—event driven widgets. Once the problems described in this section are solved, the presentation server architecture provides a simpler solution than other approaches. For example, the application server architecture requires knowledge of and involvement with the presentation technology (JSP, ASP, servlets, etc.). The thick client approach includes even more work. Developers have to program on the client as well as on the server. They also have to design the APIs and. Finally, since current remote invocation protocols such as CORBA and RMI do not support message batching (critical for dealing with the high latency), developers have to implement their own protocol from scratch.

7 Summary and Conclusion

An increasing number of personal devices provide wireless network access. The ability to connect mobile users to the Internet has opened the doors to exciting possibilities. It has also brought under the spotlight new challenges for people building software for personal wireless devices. Due to the limitations of current technology, devices trade a great deal of power for form-factor and convenience. How should developers build applications for these devices? What choices does the trade off leaves them?

Starner would have us wait until the technological advances make possible compact personal devices with processing power comparable to today's desktop computers. This would allow developers to use architectures that rely processing capabilities on the device, like today's thick client approach. The alternative involves tailoring an architecture designed for limited processing capabilities like the thin client approach to the constraints typical of current technology (i.e., portable devices and wireless networks). While this solution requires solving several hard problems, developers don't have to wait for the technology to catch up with an architecture designed under different assumptions.

Although the ability to work with current technology is a key factor differentiating the above approaches, it is not the only one. Several other characteristics make the thin client approach a viable alternative to thick clients. In this paper I have discussed development cost, integration with existing infrastructure, instantaneous deployment, improved security, and lower cost of ownership. Software developers seeking architectures for wireless applications should not dismiss the thin client approach before considering all these factors.

I have discussed why application server, a thin client architecture widely used on the Web is not suitable for applications running on wireless personal devices. As an alternative, I've presented the presentation server architecture. Presentation server departs from the application server architecture in order to deal with the constraints

of current technology while providing the benefits of the thin client approach. I have discussed the key benefits of the presentation server architecture, as well as some of the hard problems that developers adopting it will have to solve.

Finally, let me note that the presentation server architecture as outlined in this paper is more than just speculation. In fact, it is the focal point of a proven commercial product for the wireless market; Mobile Classic Blend from Applied Reasoning, Inc. [1]. This product demonstrates the viability of the architecture, and proves that the problems discussed here can be solved with current technology. It also shows that software developers don't have to wait for technology to catch up with wishful thinking. Instead, sometimes they can adapt an existing architecture to satisfy the constraints and limitations of current technology.

Acknowledgments

Bill Burdick, Rich MacDonald, and David Mitchell from Applied Reasoning, Inc. have provided valuable input during the preparation of this paper.

References

[1] Inc. Applied Reasoning Systems Corporation. Mobile Classic Blend. Available on the Web from `http://www.appliedreasoning.com/products_what_is_Mobile_Classic_Blend.htm`.

[2] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley, 1998.

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

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

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

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

[7] 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`.

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

[9] 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.

[10] Robert W. Scheifler and Jim Gettys. The X Window System. In *ACM Transactions on Graphics*, volume 5, pages 79–109. ACM Press, New York, NY, April 1986.

[11] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2000.

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

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