Truly Platform-independent Data Entry Devices

John R. Surdu and Thomas D. Wagner
Department of Electrical Engineering and Computer Science
The United States Military Academy
West Point, NY 10996
Paul Manz and Greg Ilaria
Project Manager Field Artillery Tactical Data System
Ft. Monmouth, NJ

ABSTRACT

As the Army ventures into the future and becomes increasingly digitized, a need for more technically advanced equipment is imperative. Many such systems use proprietary hardware and software solutions that are often heavy and costly, tying the Army to a specific vendor long after new technology has surpassed the capabilities of these current systems. Rather than relying on a certain vendor or manufacturer, this project has chosen to pursue the implementation of platform-independent software running on common handheld units. The research and development portion of this project included searching for the technology that will best suit the needs of implementing the platform-independent software. This paper discusses the technologies considered and the decisions made regarding that technology. It also describes the use of emulators to facilitate this research. Finally this paper discusses the pros and cons of using emulators in this software development effort.

Biographical sketch:

Major John R. Surdu is an assistant professor and senior research scientist in the Information Technology and Operations Center, Department of Electrical Engineering and Computer Science, US Military Academy, West Point, New York. He received a B.S. from the US Military Academy, an M.B.A. from Columbus State University, an M.S. from Florida State University, and a Ph.D. from Texas A&M University, and is a graduate of the US Army Command and General Staff College. He has served in various command and staff positions in the Continental United States and Korea, including team chief and computer scientist, US Army Research Laboratory, Aberdeen Proving Ground, Maryland; commander, Headquarters and Headquarters Company, 1st Battalion (Airborne), 507th Infantry, Fort Benning, Georgia; and operations officer, 1st Brigade, 2d Infantry Division, Korea.

Thomas D. Wagner is an associate professor in the Department of Electrical Engineering and Computer Science, US Military Academy, West Point, New York. He received a B.S. from Tulane University and a Ph.D. from Vanderbilt University.

Paul C. Manz serves as Deputy Project Manager for Field Artillery Tactical Data Systems, has a MPA and BSEE, is a Senior Member of IEEE, and Multiple Level III Certified Army Acquisition Corps Member.

Truly Platform-independent Data Entry Devices

John R. Surdu and Thomas D. Wagner
Department of Electrical Engineering and Computer Science
The United States Military Academy
West Point, NY 10996
Paul Manz and Greg Ilaria
Project Manager Field Artillery Tactical Data System
Ft. Monmouth, NJ

INTRODUCTION

The U.S. Army is becoming increasingly reliant on sophisticated information systems. Most Army systems use proprietary combinations of custom-written software on specific, often custom-built, hardware. This often ties the Army to a specific vendor long after new technology has surpassed the capabilities of current systems. Rather than relying on a certain vendor or manufacturer, this project has chosen to pursue the implementation of platform-independent software running on commercial off the shelf (COTS) handheld units. The purpose of this research effort was to determine the feasibility of a completely platform-independent software architecture capable of running on both Palm OS and Windows CE personal digital assistants (PDAs). A number of software and hardware technologies were explored. To facilitate this research, several different PDA emulators were used. This had the confounding effect of both facilitating and constraining software development. This paper discusses the project specification, technologies examined and chosen, and the use of PDA emulators throughout development. Finally, this paper discusses the results of the research.

WHY PLATFORM-INDEPENDENT SOFTWARE IS GOOD FOR THE ARMY

While the notion of platform independent software has intuitive appeal, this strategy has its opponents. Admittedly, a completely platform-independent software application is often less efficient in terms of execution time and memory usage than software that has been optimized for a particular hardware platform. Thus the government must make a thorough preliminary examination and comparison of user performance requirements against available hardware technology capabilities. In addition, the initial rigorous development of platform-independent software is more difficult, and therefore more costly. Finally, it is not in the best (short term) financial interest of software developers to design platform-independent software; clearly a software developer would rather get paid four or five times to port/implement the same software over and over on evolving different hardware architectures. For these reasons, many government contractors oppose the notion of platform-independent software unless forced to do so by project managers.

What then are the advantages of this approach for the customer? First, truly platform-independent software decreases the customer's reliance on a particular hardware vendor. If the software can run on a variety of platforms, the customer is free to search for better hardware without having to pay the vendor to port the software. Many Army systems are running on obsolete hardware, because the software and hardware are irreversibly linked. This means that the entire system, both hardware and software, must be upgraded simultaneously. In terms of this research, if platform-independent software for PDAs was developed, it could run today on Palm OS or Windows CE devices. As Pocket Linux matures, this same software could operate on hardware running that operating system. Perhaps next year a 32 Terabyte, Peta-FLOP (that is 10^{15} floating-point operations per second) PDA that fits on a thumbnail. This same software could run on that system as well, given certain constraints.

At this point, platform independence must be defined for purposes of this research. Clearly all software, at some point, must run on specific hardware. In general, platform independence of software is accomplished in two ways. The first method is to create compilers for every platform that take the same source code and turn it into executable code for each platform. This is the approach that C, C++, and Ada have taken (with varying degrees of success, depending on the language). Similarly, graphics libraries such as OpenGL appear the same at the source-code level on all platforms, but the libraries that are called are specific to the machine on which the software will run. The second approach is to have a virtual machine running on each platform where the exact same "executable" code can run on the various hardware-specific virtual machines. This is the approach taken by Java. In this case, the Java source and byte codes (the "executable" code) are the same for all platforms to the application

developer who writes in standard-compliant Java. This approach, however, relies on the existence of a compliant virtual machine on all platforms on which the software is to be run. Platform independence, therefore, is a matter of degree, with the same code running on virtual machines on one end of the spectrum and source code that looks the same but must be compiled for each hardware platform independently on the other end. For the purpose of this research, platform independence was to be achieved by the virtual machine approach. Both methods are fairly easily achieved on desktop computers, but they are very difficult to achieve on PDAs.

PURPOSE OF THE RESEARCH EFFORT

The initial thrust for this research was to develop a Pocket-sized Forward Entry Device (PFED) to replace the current Handheld Terminal Unit (HTU) used by dismounted Forward Observers (FO) to call for artillery fire, naval gunfire, and close air support. The current HTU system (Figure 1) was expensive, proprietary, and not readily accepted by the user population due to size and weight considerations. The focus of this research was to determine the feasibility of designing a PFED that was platform independent versus designing a system capable of immediate fielding. For purposes of this project, platform independence meant that the software could run under both Palm OS and Windows CE. The overall goal then was to develop a platform-independent proof-of-concept PFED that offered much of the core Forward Observer functionality found on the HTU.

The final design of the proof-of-concept PFED, shown in Figure 2, involves the use of two PDAs with a wireless link between them. The purpose of this configuration was to allow the FO to be able to move away from his backpack or vehicle a short distance and minimize the dismounted use of body-mounted wiring to connect essential system components. The specifics of the design and the choices made in creating this design are discussed in greater detail later in this paper.

TECHNOLOGIES CONSIDERED

As stated earlier, the targets for the PFED were Windows CE and Palm OS. It is clear that as Pocket Linux becomes more viable that this system would have to be included as well. The choice of target systems constrained the search space for platform-independent software technologies. Technologies considered were Java 2 Micro Edition (J2ME), Personal Java (pJava), Waba, IBM Visual Age J9 (J9), and Kada. All of these technologies either claim or promise platform independence. As it turns out, none of these technologies offered true platform independence.

J2ME and pJava:

When Sun Microsystems presented Java to the world in 1995, there was the World Wide Web was already well established. Its principle attribute was platform independence [1]. This meant that developers could write code in Java and compile it in such a manner that it would run on any operating system. The user did not need to worry about what operating system (OS) the platform was using. Typically, for platform-specific programming languages, source code must be

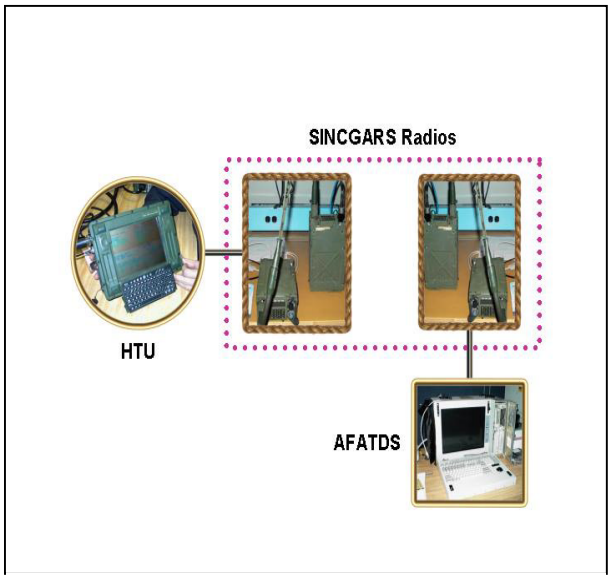

Figure 1: Current HTU

generated for each platform – with Java, this is not the case. When a user requests a Java program from a Web server for example, the client receives an *applet*. An applet is the result of compiling Java source code and is in a form known as Java byte code. This Java byte code is the equivalent for Java of an executable that results from compiling other languages. The user needs an interpreter, what has now become more commonly known as a virtual machine or VM, on the machine to execute the applets. Today Java Virtual Machines are available for most commonly used platforms, and many applications that are not Web based are written in Java as well.

Over the past five years, Sun has developed subclasses and different implementations of the Java virtual machines to coincide with the current technologies. Java has also taken into consideration the various echelons of programming requirements. For

common programming use, Sun provides J2SE or Java 2 Standard Edition. For large corporate software development, Sun offers J2EE or Java 2 Enterprise Edition. And recently, the edition of interest relevant to this paper, Sun developed a more compact version of Java, called J2ME, or Java 2 Micro Edition, for the development of software to run on personal digital assistants (PDAs).

Sun provides various virtual machines for the different versions of Java. Java Virtual Machine (JVM) supports all the classes included in the J2SE version. JVM runs on common desktop operating systems such as Windows and Unix based systems. Kilobyte Virtual

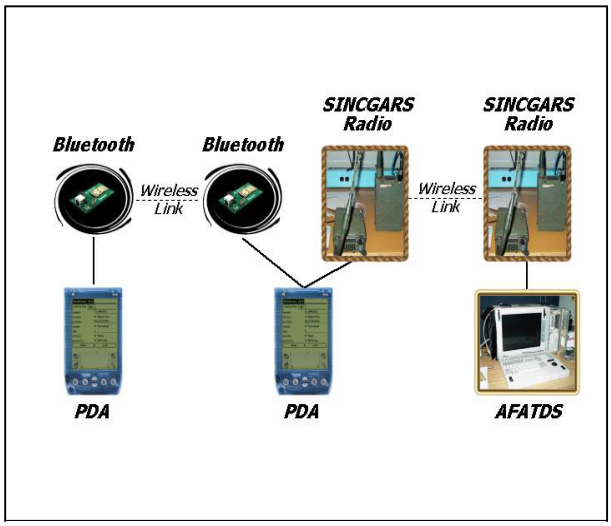

Figure 2: Target System

Machine (KVM) is designed to support the more limited version of Java, J2ME CLDC and the corresponding kAWT classes. Currently, the only PDA operating system on which KVM runs is PalmOS. As for the Windows CE devices, Sun offers Personal Java (PJava). PJava and KVM are both subsets of Java 2; however, they do not support all of the same classes. In order to achieve true platform independence, the application programmer can use only the subset of classes supported by both PJava and J2ME VMs. All of the Java resources mentioned in this paper are free and can be downloaded from Sun's Java web site [3].

Waba:

Waba is a programming platform intended specifically for small devices. Waba defines a language, a virtual machine, a class file format and a set of foundation classes. The design of Waba allows developers to use Java development tools to develop Waba programs; however, Waba is not a derivative of Java and has no connection to Sun Microsystems, the owner of the Java brand and related trademarks [3]. In fact, the software to build and run Waba programs is available free on Wabasoft's web site [4]. The software is licensed under the GNU license, which is the same licensing chosen by the developers of Linux.

The syntax of the Waba programming language is a strict subset of the syntax of the Java language. This allows developers who are familiar with Java to quickly start programming in Waba. The Waba class file and byte code format are strict subsets of the class file and byte code format supported by Java. This allows developers to use Java development tools to write programs for the Waba platform as long as they only use the subset of functionality supported by Waba, much like using J2ME.

The Waba language, virtual machine, and class file format were designed to be optimal for small devices. Features that would use substantial amounts of memory or that were deemed unnecessary for small devices were omitted from the design of the Waba language and platform. Waba has a set of foundation classes designed to be as small as possible while still containing the functionality needed to write fully featured programs for small devices.

Waba comes with a set of "bridge" classes that allow Waba programs to run anywhere Java is available. Waba programs can run as Java applets and applications. Using the bridge classes, a Waba program can run under Windows and UNIX and could appear on a web page as a Java applet. With a native Waba virtual machine, the same program could run on a small device, such as the PalmPilot.

Waba provides several advantages including mobility, functionality, reliability, and portability. Waba was designed for small, usually mobile, devices. Waba virtual machines are available that are under 64K in size (including foundation classes) and that run programs in less than 10K of memory. Waba allows developers to quickly develop versatile programs on almost any platform with development tools that are cheap (free in many cases), familiar, and easy to use. The Waba language is object-oriented and includes language features such as bounds checking and garbage collection that speed development time and allow for more robust applications. The Waba foundation classes were designed specifically to encapsulate the functionality required to build applications for small devices. Small devices normally only contain limited memory and no external storage device, so if a program corrupts memory, the whole machine may need to be reset. Waba safeguards memory access, like PJava, to prevent these types of failures. Since Waba uses garbage collection, memory leaks are extremely rare compared with programs written in other languages. With Waba, developers can write one program that will

run on a PalmOS device, Windows CE (version 2.2.1. and later) device, and any machine that supports Java (either the JDK 1.02, 1.1, 1.2 or 2.0) [4].

The advantages of Wabasoft's Waba programming language are similar to those of Java's. First, it is interoperable. Much like Java, the code can be ported from desktops running NT to PDAs running PalmOS or WinCE. Wabasoft's intention in creating Waba was to provide a capability to develop software that could run on handheld devices. Additionally they wanted to make it simple to write those programs. Its syntactical similarity to Java and its serial capabilities make it very attractive for these applications.

However, the main concern with Waba is that it is proprietary. The intent of this project was to move away from proprietary vendors in order to produce expandable software. Because of Waba's fairly recent debut, it is not established or as widely accepted as Java. Finally, its inability to communicate with the Bluetooth API stack disqualifies this programming language as a valid candidate.

J9:

IBM promises to have the most widely available, robust, and full-featured solution to platform-independent software on PDAs: Visual Age Micro Edition (VAME) Java and their J9 compiler [9]. The J9 virtual machine is available for Palm OS, HardHat Linux, and Windows CE (on several different hardware platforms). The J9 virtual machine is purported to be the most efficient PDA hosted virtual machine for interpreting Java byte codes. In addition, it can perform just-in-time (JIT) compilation as well as ahead-of-time (AOT) compilation of Java. In addition, J9 supports Java Native Interface (JNI) calls, which often seem necessary to provide access to machine-specific functions. Finally, J9 comes with its own interactive development environment (IDE) and GUI builder, MicroView. While Visual Age Micro Edition appears to be a leading candidate for the long-term solution to platform independent PDA software, currently it has an important drawback: it is impossible to write platform-independent GUI code. The only way to create VAME GUIs for Windows CE is to use the MicroView GUI builder. (This GUI builder, by the way, is surprisingly complex and counter-intuitive.) One cannot, however, use MicroView to build a GUI for PalmOS. On PalmOS, you must use the kAWT classes for J2ME. From IBM's VAME Web pages, there seems to be no ongoing effort to abstract away the platform-dependent GUI issues from the application developer.

Kada:

Kada Systems has created what they call the "industry's smallest, fastest, most complete and easily ported Java application platform for mobile devices." [10] Kada Systems claims that the Kada VM can be easily ported to other systems and that a Windows CE VM will be available soon. An advantage to the Kada VM is that is a full-featured implementation of Java that offers full use of all java.awt, java.sql and java.net classes. Once this is available for many operating systems, the full implementation of these libraries will make application development much easier; application developers will not have to learn a new version of Java. In addition to its full-featured implementation of Java, Kada Systems claims that their VM is about half the size of other full-function virtual machines. In terms of the platform independence goal of this research, Kada Systems' claim that "all code was developed with particular attention to isolating processor/OS-specific differences in an abstraction layer" is a significant draw. Unfortunately when this research began, the Kada VM was only available for Palm OS 3.5.

TECHNOLOGIES CHOSEN AND WHY

For reasons not within the scope of this paper, the technology chosen for the wireless link between the PFED and the PDA controlling the radio was Bluetooth [7]. To a great degree, this constrained the available software choices. Wireless communications through Bluetooth and IEEE 802.11 are generally achieved using platform-specific APIs written in C or C++. Finding a Bluetooth API written in Java was difficult. It appears that the only source of such an API, short of writing it oneself, is Zucotto Wireless, Inc. [8]. When this research began, the Zucotto implementation of the Bluetooth protocol stack ran under J2ME. The recently released Beta version of WHITEboard SDK 2.0 from Zucotto promises to implement the Bluetooth protocol stack in any Java environment. As stated earlier, J2ME does not run on Windows CE, and pJava does not run on PalmOS. As a result, the software architecture chosen for this research was the intersection of J2ME and pJava.

This intersection of J2ME and pJava, apart from allowing the use of Zucotto's Bluetooth stack, enabled the construction of a platform-independent user interface as well as platform-independent main code. It was not possible to write a platform independent user interface in J9. After several days of research, a representative from IBM confirmed this. Waba and Kada both promise platform independence. Waba can be run, using bridge classes, in any Java virtual machine, but Java classes could not be used in the Waba virtual machine. Kada was released late in the development process, so its use was infeasible.

USE OF EMULATORS DURING DEVELOPMENT

For a number of reasons, several emulators were used during software development. The Palm Operating System Emulator (POSE) was used to test code to see if it would run under PalmOS. To test pJava code, the pJava emulator was used. Using both POSE and the pJava emulator enabled Java code to be written on a PC and run under J2SE, J2ME, and pJava quickly. The use of these two emulators eliminated the constant, time-consuming need to load new versions of the software onto the actual devices to run each test. These emulators permitted the early development of the base functionality of the PFED as well as development of the interface.

Incorporating Bluetooth into the prototype required a third emulator, the one that comes with Zucotto's WHITEboard SDK. The reason for this was that the 1.0 version of WHITEboard did not provide a Bluetooth API that could be used independently of their emulator. The Bluetooth API relied on some machine-specific functions present in the emulator. The 2.0 beta version of WHITEboard SDK promises to be a purely Java implementation of the Bluetooth protocol stack. Using the Zucotto emulator and version 1.0 of WHITEboard SDK, the Bluetooth communication code was written, tested, and debugged.

Figure 3: Sample User Interface Running on POSE

PROS AND CONS OF USING EMULATORS

The use of emulators is a powerful tool. The use of POSE and the pJava emulators facilitated rapid development of code for three different platforms (J2ME, pJava, and J2SE) concurrently. The emulators accelerated this development by removing the need to copy the application to each platform to test each change. In some cases, the use of emulators not only sped development but also *enabled* development. Since the currently available version of WHITEboard SDK required an emulator, the development of communications code that used Bluetooth could not have been accomplished without the Zucotto emulator. Software development would have been on an halted indefinitely waiting for the 2.0 version of WHITEboard, or for local development of a Bluetooth API.

The dark side of emulators, however, was that they also constrained development to some extent. First, none of the emulators seems perfect. In more than one instance, code that ran under the emulator would not run on a real device and vice versa.

Second, the emulators do not fully model all features of the emulated device. Looking back at Figure 2, note that the PDA connected to the Bluetooth device and the SINCGARS radio, must have two communications ports. In the case of this software prototype, both ports needed to be serial ports. No PDAs come with two serial ports. If the PDA has a CompactFlash (CF) slot, a CF Serial card can be added. The Zucotto emulator allowed the software to open a serial link to the Bluetooth card. It would not, however, allow a second communications port to be opened from within the emulator. (As it turns out, while opening two serial ports can be done in C++ on a PDA, it may not be possible under current J2ME and pJava implementations.) The inability to open two serial ports under the Zucotto emulator meant that the code to allow a PDA to act as a bridge between the Bluetooth network and the SINCGARS radio network could not be written.

FUTURE WORK

Platform-independent software technology continues to mature. Since this project began, the Bluetooth protocol stack written in pure Java has become available (in beta form) from Zucotto. Sun has released versions of a Wireless Took Kit for J2ME. IBM J9 is available for several versions of Windows CE as well as Palm OS. Future work on this project involves the re-evaluation of platform-independent technologies, the redesign of the proof-of-concept system to take this reevaluation into account, and the construction of a new proof-of-concept PFED.

CONCLUSIONS

One of the primary goals of this work was to investigate the feasibility of creating platform independent data entry software to run on handheld devices. True platform independence is currently an elusive goal. The two pieces of the puzzle that currently make that goal hard to attain are the GUI and the external connections. With PJava and J2ME the GUI can be built in a platform independent fashion. TCP/IP connections can also be handled in a platform independent fashion; however the choice of Bluetooth technology limits the target architecture to J2ME for the present time. As that technology matures this problem will certainly be solved.

Another factor that causes problems with platform independence is what facilities the different target architectures provide for file access. While WinCE provides a file system that is similar in form and use to that provided by other Microsoft operating systems, Palm OS does not provide any such facility. Managing persistent data on the Palm OS platform requires the use of Palm databases that are accessed using methods that are different than simple file access.

The use of emulators in this project was a two-edged sword. First, without using emulators some of the technology would not even have been tested. For example the Zucotto Bluetooth emulator was the only choice to test this technology. The Bluetooth protocol stack is becoming available for implementation on the PDAs but could not have been tested without the use of the emulator. This allowed development using emerging technologies before they were widely available. The problem caused by using emulators is that code that performs flawlessly on an emulator is not always an easy port to the actual hardware. Again, as the technology matures these problems will certainly be resolved.

REFERENCES

(1) Peter Kestenbaum, "What is Java? A 10 minute guide for the uninitiated." On-line. Available from http://www.javaworld.com/javaworld/jw-03-1996/jw-03-10minute.java_p.html. Accessed 12 February 2001.

(2) "Object Oriented Basic Concepts and Advantages." On-line. Available from http://www.mmrg.ecs.soton.ac.uk/publications/archive/melly1995a/html/node3.html. Accessed 12 February 2001.

(3) "The Source for Java Technology." On-line. Available from http://www.java.sun.com. Accessed 12 February 2001.

(4) "Wabasoft." On-line. Available from http://www.wabasoft.com. Accessed 12 February 2001.

(5) Nazmul Idris, "Benefits of using XML." On-line. Available from http://65.1.136.127/developerlife/xmlbenefits/default.htm. Accessed 12 March 2001.

(6) "Java Technology and XML: Portable Code, Portable Data." On-line. Available from http://java.sun.com/xml/. Accessed 12 February 2001.

(7) Ericsson Microelectronics AB, *ROK 101 007 Bluetooth Module* (Kista-Stockholm: Ericsson Microelectronics AB, 2000).

(8) "Zucotto Wireless." On-line. Available from http://www.Zucotto.com. Accessed 12 March 2001.

(9) "IBM Offers Java Compatible Technology for Embedded Systems." On-Line. Available from http://www.embedded.oti.com/press/emcompat.html. Accessed 20 June 2001.

(10) "KADA Mobile Delivers on the Promise of Portability" On-Line. Available from http://www.kadasystems.com/. Accessed 28 May 2001