Java Environment for a JUMPtec WebtoNet Embedded System

Valter F. Silva[1], José A. Fonseca[2], José L. Oliveira[2]

[1] Escola Superior de Tecnologia de Castelo Branco, Avª do empresário
6000 – 767 Castelo Branco, Portugal
vfs@est.ipcb.pt
[2] Departamento de Electrónica e Telecomunicações, Universidade de Aveiro
3810-193 Aveiro, Portugal
{jaf, jlo}@det.ua.pt

Abstract. Current automation applications are often based in embedded systems with limited resources. However, even with these constraints, they often require Internet connectivity and fast programming processes. This paper presents a demonstrator based on a Jumptec system built to show the possibility of using Java as the programming language for small embedded systems. The steps to integrate the Java virtual machine in the system are described and the simplicity of developing programs for the target system is shown. Also, a small application example with real world I/O is briefly presented.

1 Introduction

In our days, developers want to obtain results rapid and efficiently. In addition, programmers appreciate to use a familiar programming language. For desktop and server based applications, many people now choose to program in Java [2] because of the features that it offers. Some of these features are: automatic garbage collection, platform independence and similarity to C++.

In embedded systems, the languages normally used are C and assembly. Assembly is not used anymore except for small procedures that must execute quickly or access to specific hardware resources. So, the usual approach is to use C. In any case, programming embedded systems in C can be a hard job if an adequate encapsulation of details is not carefully done and if a proper memory management is not included.

When considering the use of Java in embedded systems, the idea that it is "fat" and slow still persists. However, nowadays it is already possible to write Java programs to run in devices with limited resources when compared with a PC, like mobile phones [10] and other hand held devices. The variety of this kind of devices (e.g. in mobile phones) is great, so Java, by offering platform independence, can be very useful for fast developing.

It is interesting to refer that the devices with constrained resources, typical of the electronic consumer market, were the target platforms for the early form of Java [2].

However, the substantial growth of the required resources to support the Java platform delayed the use of Java in those systems till lighter versions were offered.

So, there is again a strong interest in porting Java to the embedded world, in order to make available to programmers the features that make it become popular in the desktop world.

In this paper the problem of porting Java to small embedded systems is addressed. In this particular case, the target platforms are systems used in control and automation applications which are often much more limited than the ones used in consumer electronics. The target platform chosen is a Jumptec system that can be considered a platform with reasonable resources for that sector. It's main features and characteristics are described in section 2.

In section 3 of the paper, the steps done to achieve the integration of the lighter version of the Java virtual machine in the Jumptec system are thoroughly discussed. Two simple application examples to assess the possibilities open with that integration are explained in section 4. Also, some experimental performance are presented and briefly discussed. The paper ends in section 5 where the conclusion that points to the interest in using Java as the development language for some embedded applications is extracted.

2. The JUMPtec Embedded System

In this work we use a JUMPtec WebToNet embedded system. It is based on an Intel 386Sx processor and has 2Mb of RAM memory, 2Mb of flash memory, serial port, parallel port and Ethernet Card [1]. This platform was chosen because of its good price-performance ratio. Figure 1 present two views of the system. Its physical dimension is 11.2x11.6 cm.

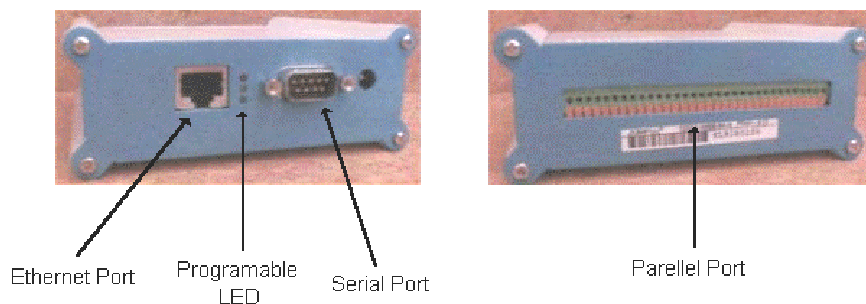

Fig. 1. JUMPtec WebtoNet system

This JUMPtec system integrates a Web server software that allows to control, monitor and manage the embedded device from any web browser. In particular, it is very useful to manage the flash disk (Figure 2). It has assigned an IP address and a host name like any other system in a network. This web server runs in background so, it is possible to keep communication with the embedded system while running

another application. This feature is useful for the purposes of this work, as it will be seen later.

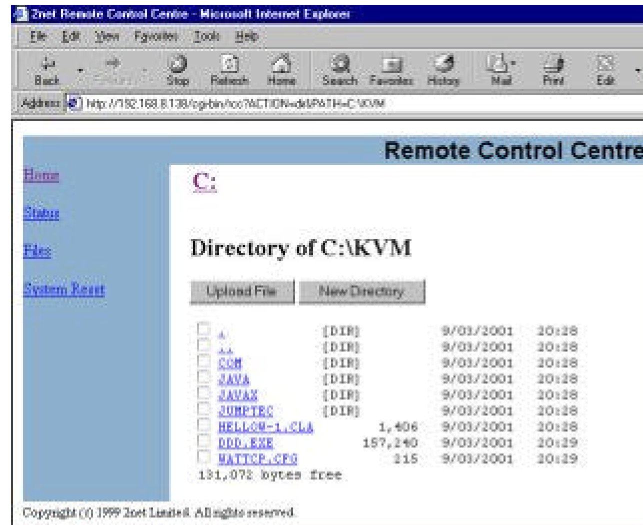

Fig. 2. The flash disk managing interface

The JUMPtec operating system uses Dr-DOS [4] that is compatible with Ms-DOS, but it is more oriented to small devices. Thus, all programs developed to run in this JUMPtec system must be compatible with DOS based operating systems.

3 Embedding the JVM in a Limited System

Nowadays, there are three editions of Java, each one oriented for a target market:

- Java 2 Enterprise Edition (J2EE), for the enterprise world, described as a solid and scalable internet solution.
- Java 2 Standard Edition (J2SE), for desktop computers and their common applications.
- Java 2 Micro Edition (J2ME). This edition is dedicated to small devices, with a limited processing capacity, and imposes a limited memory footprint.

Java 2 Micro Edition is a very flexible and portable edition of Java. To provide this flexibility, J2ME is organized in configurations and profiles. One configuration defines the basic Java Runtime Environment (JRE). It includes the virtual machine and the essential classes for a device family with similar capabilities. A profile adds specific features to a particular configuration. Profiles are more oriented to a particular system. This architecture is depicted in figure 3.

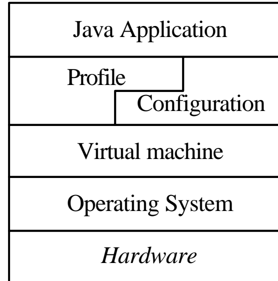

Fig. 3. J2ME architecture

Presently there are 2 configurations supported by J2ME: CLDC, *Connected Limited Device Configuration* and CDC *Connected Device Configuration*. The CDC uses the classic virtual machine, while CLDC uses the new Kilo Virtual Machine (known as KVM). The CLDC configuration is available in Sun Web page [3] for free download. This archive includes the virtual machine, the API, and some tools, like the preverifier, all in open source written in C. The available release is intended to run in Windows and Solaris. In a previous work we have evaluated the suitability of the KVM to run on different operating systems, namely in Windows, MS-DOS and Linux [12].

The operating system available in JUMPtec is DOS, so, first, the virtual machine must be ported to this operating system. According to the KVM porting guide [5], the code must be compiled with an ANSI C compiler with 32 bits pointers. So, to fulfill these requirements, the DJGPP Gnu C from Delorie [6] was chosen to compile the KVM code. This compiler supports 64 bits integer arithmetic, but for compatibility with others that doesn't support, some functions to perform basic operations must be provided. This is necessary because in Java a `long` takes 64 bits. These functions are identified in table 1.

Table 1. Speciffically develloped functions

Function	Java equivalent
`long64 ll_mul(long64 a,long64 b);`	`a*b`
`long64 ll_div(long64 a,long64 b);`	`a/b`
`long64 ll_rem(long64 a,long64 b);`	`a%b`
`long64 ll_shl(long64 a,long64 b);`	`a<<b`
`long64 ll_shr(long64 a,long64 b);`	`a>>b`
`long64 ll_ushr(long64 a,long64 b);`	`a>>>b`

The `long64` data type is a struct containing 2 integer numbers.

Some of most populars applications for embedded systems are control systems, which normally include floating point arithmetic. So, in our port of KVM we include the floating point support in KVM, and we develop four more functions shown in table 2.

Table 2. Functions to support floating point

Function	Java equivalent
`long64 float2ll (float f);`	`(long)f`
`long64 double2ll (double d);`	`(long)d`
`float ll2float (long64 a);`	`(float)a`
`dlouble ll2double (long64 a);`	`(dlouble)a`

Our target processor is a 386sx, so it has no Floating Point Unit. In that way, we emulate the floating point in software linking the library for emulation supplied with the compiler.

Other issue in porting the KVM to DOS is the length of the file names which is 8 characters plus 3 at the extension. In Java, the compiled files have a `.class` extension and the name of a file can contain more than 8 characters (Joliet format). One of the possibilities would be, of course, to limit the length of the class names. However, this would not be pleasant for a Java programmer and would violate Java specifications. So, it was decided to modify the *class loader* to overcome the limitation. The "shrinking" problem was solved recurring to the possibility of identifying the class name within the class file itself (class file format [7]). So, before loading any class, the system checks if the name is longer than 8 characters. If not, a file must be available with the same name. If the name is longer than 8, the system generates a sequence of possible names by truncating the name to 6 characters and adding a ~#. This # is a sequence from 1 to 9. The system starts opening each of the files till it locates the one that contains the full name of the desired class. One example is shown in table 3.

Table 3. Example of two classes and the respective file names in DOS

Joliet name	DOS name
`my_program.class`	`my_pro~1.cla`
`my_program1.class`	`my_pro~2.cla`

The new algorithmused in the modification of the *class loader* was obtained from the original and is presented in figure 4.

All the files that run in KVM must be pre-verified [8]. Because of the file length problem, the pre-verifier utility was also modified in order to change the file extension from `.class` to `.cla`.

In Java, it is also possible to define native functions, i.e., functions specific for the target platform. As shown in figure 5, this allows the program developer to create more specific applications. In the specific case of the KVM, all the native functions must be compiled with it, which is not the usual procedure in the classic virtual machines.

For the JUMPtec platform, two native functions were developed in order to have access to the hardware ports. These are important functions not only for debugging purposes but also to the usual applications in which this type of platform is used. The

source code of the virtual machine was also modified in order to redirect all warnings to the serial port.

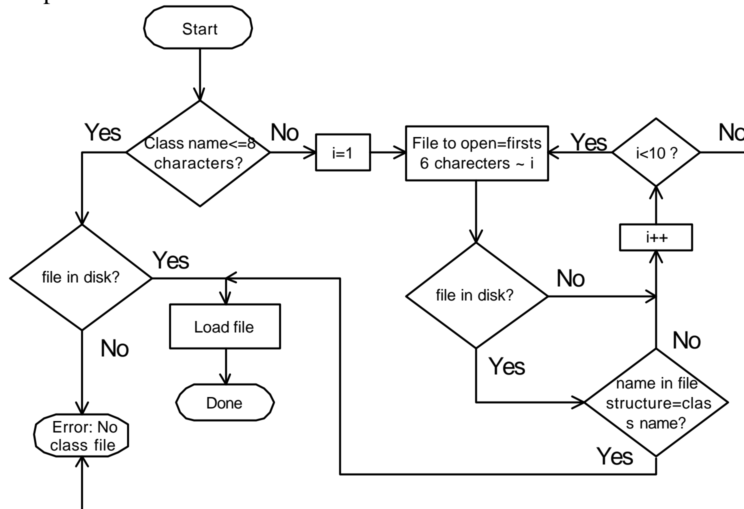

Fig. 4. Class Loader algorithm

To have access to the Ethernet card, the JUMPtec has a specific DOS packet driver. In this case, to make available in Java the access to the network card, we decided to use the connections framework available with CLDC (partially implemented in native functions) joined together with a library obtained from part of a so-called Wattcp system. We also use some function in native form, which are necessaries to implement a simple ping program. The Wattcp system provides access to a network via a packet driver and is an open source available for download [9].

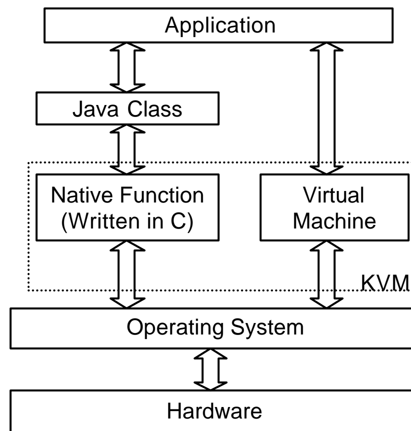

Fig. 5. Native functions

4 Program Development

In figure 6 the steps used in the development of a Java application program for the JUMPtec system are identified. The first step is obviously the writing of the program in any common text editor available in a desktop computer. All the semantics of Java are in this case the same as for any other system. The program is after compiled using `javac` from Java Development Kit (JDK).

All virtual machines must be able to identify and reject invalid class files. However, since the standard class file verification approach defined in J2SE needs substantial memory resources, it is unsuitable for small devices. CLDC defines then an alternative mechanism for class file verification in which an attribute is added to each method during an operation that is done at the development workstation, called preverification. This is specific of the KVM development process. However, the modified class file is still runable in other virtual machines [8]. This modification leads to an increase of about 5% in the size of the class file.

After the preverification, the file is ready for download to the JUMPtec system. This operation can be done using the web application shown in figure 2.

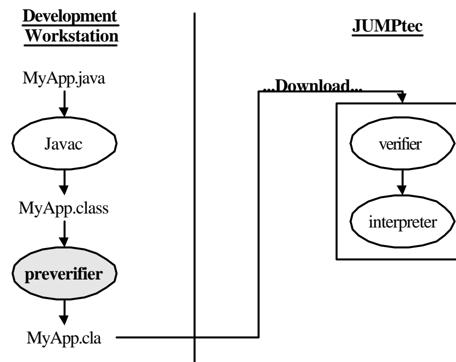

Fig. 6. Program development

In order to leave the maximum memory space available for the KVM, a small boot loader was developed. This boot loader, written in C, reads one bit of the JUMPtec parallel port and runs the programmer's application or the web server depending on the logical state of the bit. So, if the bit is '0', the boot loader runs the file "my_bat.bat". This overcomes the absence of a keyboard.

Before running the virtual machine, the packet driver must be installed in order to have access to the Ethernet port from the Java program. For the same reason, a file called `wattcp.cfg` [9] containing the Ethernet configuration, IP address, host name, subnet mask, DNS server, gateway and network domain must be present in the KVM directory.

Note that the boot loader is only executed in the JUMPtec system at startup so, whenever the user wants to run the application program, he or she must reset the system. This operation can also be done using the web server.

In order to facilitate the development of applications, two packages were previously developed.

4.1 Package `jumptec`

This package has three important classes: the class *Port*, the class *Net*, and the class *Led*.

The *Net* class has some static methods derived from wattcp [9]. Its main function is to provide some basic Ethernet functionalities.

The *Port* class has two static methods similar in parameters and functionality to C functions: `outportb()` and `inportb()`.

The last class, *Led*, serves to turn on and off the programmable led in the front of the JUMPtec case (see figure 1). It is useful for diagnosis and debugging.

4.1 Package `jumptec.comm`

This package has two important classes, one to handle communication through rs232, and another to deal with the parallel port.

Some common rs232 functions are supplied in static methods, like send_string(), send_byte(), receive(), etc. For the parallel port, the most commonly used features like set_port(), set_bit(), read_port(), are also supplied.

5 Applications Examples

In order to be able to run a user application in the Jumptec system it is necessary first to download the compiled version of the KVM and, at least, the adequate APIs. The KVM occupies 325 Kb. If all the APIs are also included, it is necessary to use about 525 Kb, so the overall system takes 850 Kb of disk. Although this system uses flash disk, there still is a division into 4Kb blocks. So, each file occupies an integer multiple of 4Kb. This leads to a significant waste of memory resources. Once the download of these tools is done, the user must prepare and dowload the application.

Two basic examples were prepared in order to demonstrate the functionalities of the system. The first example illustrates the use of the system in a simple embedded application. The second example is a typical network application.

All applications for the JUMPtec system are developed in a desktop PC and can be tested using a DOS operating system or even in a DOS window under Windows. This is very useful for test purposes because JUMPtec has limited facilities. This is possible because JUMPtec is indeed a PC IBM compatible system.

The first application example scenario is show in figure 7.

In this example, depending on a character sent by the host PC, a corresponding LED turns on or off. Also the state of JUMPtec switches can be read. This is a very simple example, but demonstrates the functionality of the system to work easily with I/O as it is the case of typical embedded applications in the fields of control, robotics, etc.

With this system the time to develop an application is substantially reduced when compared with the one needed if other typical language was used for this kind of device. Besides this advantage, it should be noted that the final size of the application

is rather modest, being 1Kb in this case. In this experience the startup time was also determined and was found to be approximately 5 seconds, including system reset, DOS start and application launch.

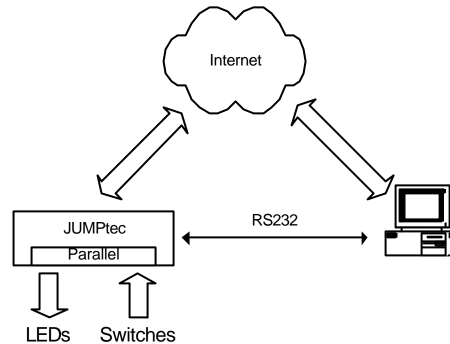

Fig. 7. Application example

In order to evaluate the performance of the Java solution when compared with other traditional development languages, the cycle time of the led on/off operation was measured. The results are shown in **Error! Reference source not found.**.

Table 4. Comparision between diferents programming languagues

Programming language	Execution time	Tool
Assembly	14 µs	Masm 6.11
C	14.8 µs	Turbo C 2.0
C++	47 µs	Djgpp 2.95
Java	640 µs	Javac

As expected, the Java program leads to a cycle time at least one order of magnitude larger. However, the result also shows that the temporal response is adequate for most common applications in the control and instrumentation fields. In fact, if processes involving movement are not considered, (like mobile robotics) many industrial processes such as, e.g., temperature or flow control, can accept cycle times much larger than the one obtained here.

If a comparison is done specifically between the application in C++ and in Java, it should be noticed that the dimension of the application code to load in the system is highly favorable in Java. Obviously this is due to the need to include class codes in the C++ code.

Besides the fast prototyping, another conclusion that can be extracted from this work is that there is an important advantage of Java if the on-line loading of applications is considered an important feature. This opens an interesting possibility for using mobile agents [11] in distributed embedded systems.

The usage of mobile agents or remote boot implies that some network connectivity must be provided. Thus, a second example consisting in a simple typical network application was developed. Giving an IP address sent from the development PC to the Jumptec system via the serial port the application must return the correspondent host

name. This small example was developed only to illustrate the use of the ethernet card. It is obvious that the packet driver must be previously installed and the file containing the system configuration must be in the same directory of the KVM.

6 Conclusions

The preliminary work presented here shows that it is possible to embed a Java virtual machine in systems with limited resources such as the ones used in some instrumentation and control applications. In this specific case, the Kilo Virtual Machine (KVM) was embedded in a 80386SX based Jumptec system with just 2Mbytes of solid state flash memory. It was shown that it was possible to limit the KVM and the necessary classes footprint to just 850 KBytes. This leaves enough memory space to many interesting applications in the referred domains.

In the paper, two very simple examples were briefly discussed. Some measures were then presented showing that the applications written in Java were slower than if other languages like C or C++ were used (what was obviously expected). However, it was clear that the response times obtained are acceptable for the needs of many industrial applications.

From this work it is then possible to conclude that Java offers promising features when considering its use for embedded systems: easy and very fast code development, small size of downloadable code, sufficient response time. Further investigation must nevertheless be carried on to fully demonstrate this assumption, namely with more demanding application examples. Future work also includes exploring the possibility of using mobile agents in distributed embedded systems.

References

1. “WEBtoNET technical Manual”, JUMPtec Industry, 1997.
2. Eric Giguère, “Java 2 Micro Edition”, John Wiley & Sons, Inc, 2000
3. Java Web page, http://java.sun.com
4. Dr- DOS web page, http://drdos.com
5. “KVM porting guide”, Sun Microsystems, Inc, 2000.
6. Delorie Web Page, http://www.delorie.com
7. Tim Lindholm, Frank Yellin, “The Java Virtual Machine Specification”, Addison-Wesley, 1996.
8. “The K Virtual Machine (KVM), a white paper”, Sun Microsystems, 1999.
9. wattcp web page, www.wattcp.com
10. Qusay H. Mahmoud, “MobiAgent: A mobile Agent Based Approach to Wireless information systems”, School of Computer Science, Carleton University, Canada.
11. V. Pham and A. Karmouch, Mobile Software Agents: An Overview, IEEE Communications, Vol. 36, No. 7 (1998) pp. 26-37.
12. V. Silva, J.L. Oliveira, J.A. Fonseca, “Ambiente de execução de aplicações Java para sistemas de recursos limitados”, CRC’2001, 29-30 November, Covilhã, Portugal.