USING JAVA LANGUAGE FOR CONTROL ALGORITMS

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

1 – Escola Superior de Tecnologia e Gestão de Leiria, Morro do Leno, Alto Vieiro, 2401 Leiria
vfs@est.ipcb.pt

2 – Departamento de Electrónica e Telecomunicações, Universidade de Aveiro, 3810-193 Aveiro,
{jaf,jlo,alex}@det.ua.pt

Abstract: constrained resources devices used often in embedded systems are normally programmed using languages considered of low level. This has important implications in program development and maintenance. In fact, the code becomes less legible and the memory management in not an easy task. This problem can be overcome if the use of a high level language with garbage collector mechanism is considered. However, it is often a belief that from these high level languages results slowly programs, and so, they are unsuitable for embedded applications. In this paper this problem is addressed to show the applicability of Java to develop programs for constrained resources devices. An example of a control system for temperature control is presented to illustrate the advantages of the approach.

Keywords: Control systems, Java, virtual machine, garbage collector.

1. INTRODUCTION

The idea that Java is a "fat" and slow language still persists in our days. Many works, like the use of a hot spot virtual machine (Sun, 2001), show that this is not correct. Also, the current possibility to use the Java virtual machine in systems with limited resources contributes to change that belief. However, it has not been shown yet that Java is suitable to be used in the control of real time systems such as temperature control, or, more exigent ones, like mobile robots. In this paper this issue is addressed by presenting some experiences in which Java is used as the programming language for some control algorithms running in an embedded system with limited resources. Also, some performance comparison with the C language is done.

The results show that, in spite of being slower, Java is perfectly adequate for that purpose.

2. CONTROLLING EMBEDDED SYSTEMS

2.1 Typical Architecture of Systems for embedded Control

Typical systems for embedded control are based on microprocessors or micro-controllers. Normally these are connected to some acquisition device such as an A/D Converter or to a more specific device, like a temperature measurement solid-state chip. After the pre-processing of the data acquired, the CPU does some calculations and determines a resultant output that will actuate in the system through a D/A unit and interface electronics.

In embedded systems the controller CPU is in principle a low-processing power device (e.g. an 8 bit micro-controller). This is reflected in the program development. For example, normally, for these devices, the programmer can only use integer arithmetic, which is a major limitation for control systems. To overcome this problem the programmer can build his/her own library, or can use the compiler floating point library if one is available. However, the use of floating point arithmetic at this level can waste many resources, namely memory and execution time. Another issue in programming these systems is the use of the C language and, in some specific situations, Assembly. This originates a need to perform a very careful programming to achieve the correct encapsulation and memory management and can be a hard work to do.

The next step in devices for real-time control shows in industrial applications. They consist in another kind of devices with some resources that offer more possibilities than micro-controllers such as a simplification in the use of floating point arithmetic and some similarity with desktop PCs. However, these systems are more complex and expensive than micro-controllers. Many companies offer in their product line systems with these characteristics. One example is JUMPtec, a German company that offers many systems with various levels of resources, from very simple ones to more complex systems based in Pentium processors. In this paper, one of the intermediate systems in the JUMPtec line was used for the experiments carried on.

2.2 JUMPtec System

The JUMPtec system used in this work is called JUMPTec WebToNet. It includes a 386SX processor, 2 MBytes of RAM and 2 MBytes of flash memory. This system offers also many input/output interfaces namely a serial port, a parallel port and an Ethernet card. A view of the system can be obtained in figure 1.

The system integrates a Web Server, and so, it can be accessed from any web browser. The management, control and monitoring of the system is done using this web server and an application supplied by JUMPtec. For example the disk management is done via the web browser with the help of the Web server. This web server runs in background, so it is possible to keep communicating with the embedded system while running another application. The operating system used is the Dr-DOS, which is an Ms-DOS compatible OS oriented to constrained resources systems. The JUMPtec system has assigned an IP address and a host name like any other system in a network. So, for the web browser, its web server is just a "normal" one, without any specific constrains. The JUMPtec system is compatible with a desktop PC, so the usual programs for desktop can be used.

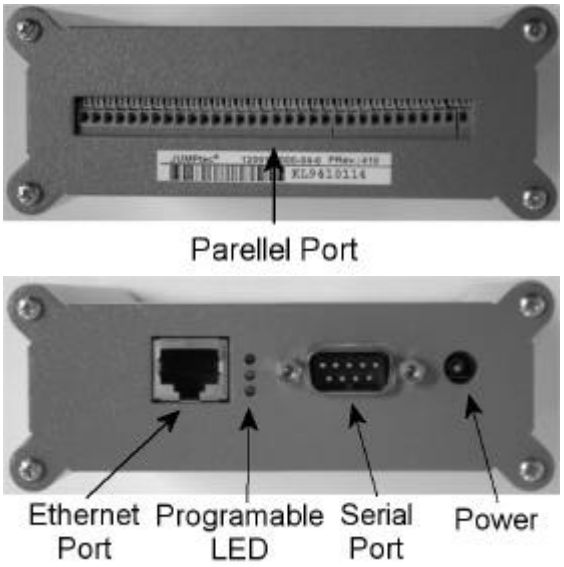

Fig. 1. Some views of the JUMPtec system.

2.3 Programming control systems

As it was said before, the languages typically used to program embedded control systems are C and Assembly. These languages lead to programs with very good performance. However, the development time is relatively long and the portability is not the best. If the target system changes, it is almost certain that the program must be adapted even if the compiler is the same. The opposite is also true, i.e., if there is a decision to substitute the compiler even without changing the system, it is probable that the program must change too.

In some, fortunately rare, situations the use of Assembly is yet required, either to increase speed or to spare memory. However, the development of a not very complex Assembly piece of code takes a large amount of time and pushes the programmer to be extremely cautious with the program flow. Even when using C, with which the development is much

quicker, the programmer must care about memory management.

The memory management problem can be solved if a language with a Garbage Collector is used. This mechanism cleans the objects in memory that are not used any more. A language that supports this mechanism is Java. Also, when using Java, the development time can be reduced, because this is a very structured language and it is object oriented. This last issue is very important not only from the code structure point of view but also because most programmers are nowadays trained in object-oriented languages. This will certainly increase the popularity of Java even for embedded systems programming.

3. PROGRAMMING CONTROL ALGORITHMS IN JAVA

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 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 development. This is the case, for example, of the small devices/terminals for the UMTS mobile network for which the Java programming language is being used (KVM applets). This will be an important test to verify the assumption about the interest of Java for limited devices.

When talking about Java, 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 the language. However, the substantial growth of the required resources to support the Java platform delayed the use of Java in those systems until lighter versions were offered. So, there is again a strong interest in porting Java to the embedded world, in order to make available to the programmers the features that made it become popular in the desktop world. But to use Java in a specific processor, the virtual machine must be ported to the new target, as well all the necessary classes must be present in the system (Silva, *et al*., 2001).

Java 2 Micro Edition (J2ME) is a recent edition of Java very flexible and easy to parameterise as it is targeted to a large number of devices. To help in achieving flexibility, the edition provides configurations and profiles. Configurations define the minimum functionalities for a device family. Profiles are more oriented to a particular device, defining more classes or, in addition, more specific functions (called native functions). The architecture overview is presented in figure 2.

Currently, Java 2 Micro Edition defines two configurations: the CLDC, *Connected Limited Device Configuration* and the CDC, *Connected Device Configuratio*n. The CDC uses the usual Java virtual machine and the CLDC uses the new *Kilo Virtual Machine* (KVM), available in open source in Sun Microsystems site (Sun, 2002). Like the name shows, KVM is oriented to devices with some Kbytes of memory and reduced processing capacity. The code is relatively easy to port to any platform. This KVM is prepared to run in systems were some sockets stack and others features characteristic of operating system not based on DOS are available.

In (Silva *et al*., 2001) it is presented the integration of the virtual machine (KVM) and all the necessary classes of J2ME for *Windows*, *Linux* and DOS and consequently for the JUMPtec system described in section 2.2. Some modifications made to the virtual machine are described and also the necessary steps to add some new features to it, like the floating point support, not included in the original version.

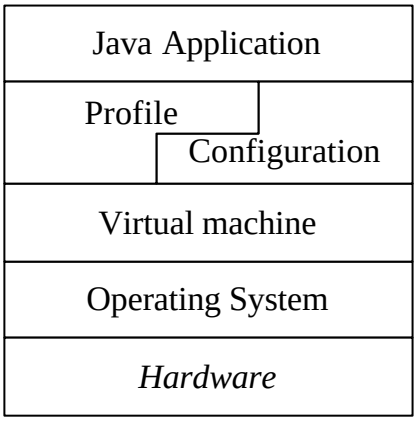

Fig. 2. System architecture

The idea behind the referred work was to show that it is possible to use Java with systems limited in resources. Thus a first step towards the use of the language in programming control systems can be achieved.

Besides embedded systems resources, another problem concerning control systems is the need to impose a specific sampling interval. This requires often the knowledge of hardware details of the target system such as the timers' programming specifications. Java offers a simple solution for this, which is the time window generation.

In Java it is possible to develop a class derived from the `Thread` class and sleep the task the desired time. This is very easy to do and leads to very simple code in the main application. In fact, it just requires the creation of an object of this class and to start its execution. This feature is important because it is totally independent of the hardware and of the operating system. Using it, it is possible to generate some Java soft real-time applications within just a few minutes.

On the other hand, the use of KVM with hard-real time applications in any platform is not yet possible. There are two proposals for a real-time Java standard published (Bollella, *et al*., 2000; Consortium, 2000).

They have in common the belief that there is no such thing as real-time garbage collection, and to avoid the non-determinism of normal GCs they propose extended memory models where some areas are manually managed. This approach does not solve the GC problem, but leaves the memory management in the hands of the program developer, whereas we believe that one of the main benefits of Java is the automatic memory management.

However, some research is already made and in the present there are at least two Java systems with hard real-time capabilities: Jamaica (Siebert, 1999; Siebert, 2000; Siebert and Walter, 2001) and PERC (NewMonics, 2002). These systems are hybrid, so, the Java code is converted to the platform native code and linked with some specific Java features, like the garbage collector.

In the University of Karlsrhue, Germany, a Java virtual machine totally implemented in hardware with hard-real time capabilities is in development (Fuhrmann *et al*., 2001). This project also has some others important features for embedded systems, like interrupt service routines. On the other hand, because is implemented in hardware, it is not portable to any other system.

4. EXPERIMENTS WITH JAVA USING THE JUMPTEC SYSTEM

In order to demonstrate the possibility of using Java for control applications, some experiences were made having the described JUMPtec system as the target platform. The first experience is just a simple program which, depending on the contents of a frame received in the serial port, activates some bits of the parallel port and reads some of the parallel port inputs (Silva, *et al*., 2001). In this program, also the Ethernet connection is tested recurring to a specific frame. This first test uses all the subsystems available on the JUMPtec.

The next experience made was just a simple test consisting in a cycle turning on and off a led available on the back of the JUMPtec system (see figure 1). The voltage signal at the led can be used to measure the cycle time. The JUMPtec has also a parallel port but the accessible outputs of this port are lines switched by relays. The switching times are too slow to be useful to make the measurements. The test was done using Java, C++, C and assembly (Silva et al., 2002). The results showed that Java is clearly slower than the others languages. Even comparing with the other object orient language, C++, Java is one order of magnitude slower. However, there are some advantages in using Java, namely the size of the application to be downloaded to the JUMPtec. This characteristic opens an important possibility to use mobile agents in this kind of devices (Mahmoud, 2001).

After, the impact of using floating point or integer arithmetic was evaluated. To do this, a set of calculations, similar to the execution of a control algorithm, was included in each cycle of the previous experiment. The final program was tried in a desktop PC (turning On and Off a bit in the parallel port) and in the JUMPtec system. C and Java were the languages used in this case. It should be noticed that the JUMPtec system doesn't include a floating point unit. The obtained cycles time are presented in table 1.

Table 1 Test results

Language	Arithmetic	JUMPtec	PC
Java	Floating point	14 ms	33.5 µs
	Integer	1.28 ms	25 µs
C	Floating Point	12.5 ms	8.6 µs
	Integer	65 µs	0.6µs

With theses tests, we can also verify the importance of using the floating point unit.

In the table it can be seen that the absence of a floating point unit in the JUMPtec has an important effect in the performance of an application with floating point arithmetic. For example, considering the same program in C with integer arithmetic and in C with floating point arithmetic, the last is about 200 times slower than the first. In contrast, in the desktop PC, with floating point unit, the program using only integer arithmetic is only 14 times faster than the one using floating point arithmetic. If we look only for the row correspondent to the integer arithmetic and to the JUMPtec column, it can be realised that the same program in Java is twenty times slower than in C.

An important result here is the fact that, considering just floating point arithmetic in a system without specific FP unit like the JUMPtec, Java performs at the same level as C. This means that, for control applications in constrained resources devices, there will be an equivalence of performance and so, the other properties of Java clearly point to this language as a good substitute of C.

After these experiments, a real application close to systems used in industry was tried. It consists in the temperature control of a small kiln similar to the ones used in ceramics. Three algorithms were tested in the control of the kiln temperature: an on-off algorithm, a ramp algorithm and a PI algorithm.

The system architecture is presented in figure 3. The system incorporates a HP model 34970A data acquisition and switch unit. This device acquires the temperature from the kiln, and acts in the kiln with the help of a power controller. The interaction between the JUMPtec and the system is made via a

serial interface. To access this interface in the JUMPtec a class developed in a previous work (Silva *et al*., 2002) was used.

The communication between the JUMPtec and the HP is based in a specific command sequence. To have a correct encapsulation of the details implementations of the commands sequences a class with some relevant methods was also specifically developed.

The temperature acquired during operation of the system is stored in the hard disk (implemented by a flash memory) of the JUMPtec. After the desired operation time this data is downloaded to the PC via Ethernet. Because the JUMPtec includes a web server, this data can also be saved in a "html" file and viewed in a Web Browser.

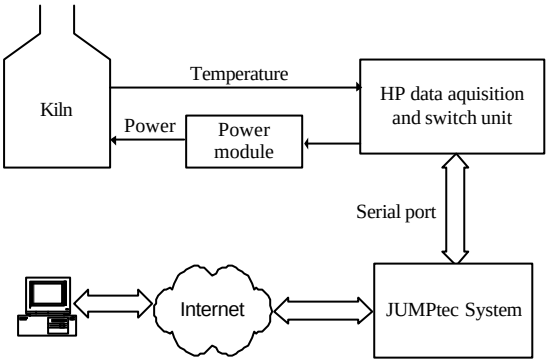

Fig. 3. System architecture

The class code of the controller is very simple and can run in any platform where a virtual machine is available. The developed code was tested in a PC and in the JUMPtec system without any change or recompilation.

For the On Off temperature controller, the code is:

```
class OnOff extends Thread{
OnOff (String file, float setPoint){
//Initialise the file
//the HP and the Set Point
  }
public void run(){
 while (true){
//controller code
sleep (2000); //sleeps for 2 seconds
  }
}
```

The class code presented has a constructor responsible for the initialisation of the file to store the read temperature and for initialisation of the set point. As it can be seen in the code, there is no specific reference to timers programming and thus this code can be run in any hardware or OS where a virtual machine is present.

But, this is not the only class necessary to the system. The complete program has also a main class responsible for the starting of this thread. The main class is independent of the algorithm in use. It just creates an object of the class presented and executes the `run` method of the object.

This program example has just a thread. However Java permits the use of multiple threads. Multi-threads systems can be used for more complex tasks, for example in a multi-sensor system, or for better encapsulation. In the case of this controller a thread for temperature acquisition and another for power calculation can be used, so some data between these threads must be shared.

When considering using more than one thread sharing data between them, mutual exclusion must be carried on. In Java, the mechanisms necessary to guarantee mutual exclusion are handled by the virtual machine. The programmer only has to indicate the classes or fields for which the virtual machine has to guarantee mutual exclusion. This is done with the qualifier `synchronized` provided by the Java language.

With a multi-thread system, the thread scheduling and management must be performed. This operation is done by the virtual machine even if the operating system does not have multi-task capabilities, as it is the case of the Dr-DOS OS used in the JUMPtec system.

5. CONCLUSIONS

In this paper the advantages of using a high level language such as Java to program embedded systems, particularly control algorithms, were presented. This change in the language brings several benefits, namely the reduction of the program development time and the simplification of the memory management. Java is an object oriented language and so, the resultant programs are very structured and easy to read. On the other hand, the platform independence and the garbage collection can be very important features for embedded systems because of the reduction in the time necessary to develop or port an application.

Some performance comparisons between Java and C in programs using either integer arithmetic or floating point are also presented and discussed. The measurements demonstrate the applicability of Java in system with some performance limitations and that Java can perform practically as well as C when using FP arithmetic in systems without FP unit.

Some experiences to control the temperature of a real kiln were also done using a JUMPtec system with constrained resources. To demonstrate the platform independence, the same program was executed in a desktop PC. With Java the programmer can be unaware of hardware details as, for example, the ones required to define the sampling time window as they can be totally managed by the virtual machine.

REFERENCES

Bollella, G., Brosgol, B., Dible, P., Furr, S., James, G., Hardin, D., Turnbull, M. (2000), *The Real Time Specification for Java*, Addison-Wesley, June.

J Consortium (2000), *Real-Time Core Extension*, P.O. Box 1565, Cupertino, CA 95015-1565, Setembro.

NewMonics (2002). *NewMonics Web site*. http://www.newmonics.com

S. Fuhrmann, M. Pfeffer, J. Kreuzinger, Th Ungerer e U. Brinkschulte (2001), *Real-time Garbage Collection for a Multithreaded Java Microcontrollor*, In 4ª IEEE Symposium on Object Oriented Real-time Distributed Computing (ISOR), Magdeburg, Germany.

Siebert, F. (1999), *Hard Real-Time garbage collection in the Jamaica Virtual Machine*, In The 6th International Conference on Real-Time Computing Systems and Applications (RTCSA '99), Hong Kong, Dezember, IEEE.

Siebert, F. (2000), *Eliminating external fragmentation in a non-moving garbage collector for java*, In Compilers, Architectures and Synthesis For Embedded Systems (CASES 2000), San José, November

Siebert, F., Walter, A. (2001), *Deterministic execution of Java's primitive bytecode operations*, In Java Virtual Machine Research & Technology Symposium '01, Monterey, CA, April.

Silva, V. F., Oliveira, J.L., Fonseca, J.A. (2001), *Ambiente de execução de aplicações Java para sistemas de recursos limitados*, Acta da 4ª Conferência de Redes de Computadores, 29-30 Novembro, Covilhã, Portugal.

Sun Microsystems (2001). *The java HotSpot Virtual Machine*. May. http://java.sun.com/products/hotspot/docs

Sun Microsystems (2002). *Sun Microsystems Web site*. http://www.sun.com

Valter Silva, José A. Fonseca, José L. Oliveira (2002), *Java Environment for a JUMPtec WebToNet Embedded System*, In Architecthure of Computer Systems – Java for Embedded Systems, Karlsrhue, Alemanha, April.