Communicating Process Architectures 2006
Peter Welch, Jon Kerridge, and Fred Barnes (Eds.)
IOS Press, 2006

Performance Evaluation of JCSP Micro Edition: JCSPme

Kevin CHALMERS, Jon KERRIDGE and Imed ROMDHANI
School of Computing, Napier University, Edinburgh, EH10 5DT
`{k.chalmers, j.kerridge, i.romdhani}@napier.ac.uk`

Abstract. Java has become a development platform that has migrated from its initial focus for small form devices, to large full scale desktop and server applications and finally back to the small in the form of Java enabled mobile phones. Here we discuss the necessary requirements to convert the existing JCSP framework so that it can be used in these resource constrained systems. We also provide some performance comparisons on various platforms to evaluate this implementation.

Keywords. JCSP, Java 2 Micro Edition, Mobile Devices, Real-time systems.

Introduction

Java, as both a platform and a development language, has on occasion been examined as a possible tool for the development of real-time systems. Evaluations of such implementations range from the almost favorable [1], to the total dismissal [2] of Java as a potential real-time solution, but this has not deterred implementations of Java in embedded systems. Mobile phones especially, in the form of Java MIDlet capabilities [3], have shown that while not the best solution, the portability and popularity of the language has pushed it into the real-time domain. Here we do not consider Java's properties as a real-time platform as this falls outside the main focus of the work presented. Such features as a more accurate system timer and reduction of the impact of garbage collection are not addressed, the main reason being that many mobile devices have a fixed Java implementation that cannot be replaced with a more real-time friendly one.

The main aim and motivation here is to provide a first initial attempt at modifying the JCSP framework to work on a mobile or embedded system device, as well as providing some ideas on how to scale back up the resulting package to provide something closer to the full scale implementation. JCSP has recently been released as open-source (www.jcsp.org), and when coupled with the expanding embedded Java market it becomes possible and desirable to implement a version of JCSP [4, 5] that can operate on systems that contain such small resource footprints as those experienced in common mobile phones. JCSP itself can find its roots in Communicating Threads for Java (CTJ) [6] which has been examined in a real-time environment [7], so initially it may appear that implementing such a system would be fairly simple. This is not the case however, as there are many limitations in a Java 2 Micro Edition (J2ME) environment when it is compared to the full scale Java 2 Standard Edition (J2SE) available on desktop machines. The necessary steps to convert JCSP to operate in this much reduced platform are covered in Section 2, and are really more a case of removing features rather than modifying code. First of all, in Section 1 we will examine current mobile device technology and how the various versions of J2ME sit within them. Section 4 presents the results of the *CommsTime* test on various different virtual machine implementations to provide a first evaluation as to how well JCSP will operate in resource constrained devices. Two different phone platforms are assessed, one of which a common PDA / mobile phone cross over and they other a common mobile phone device, and this helps us to determine the efficiency of the underlying thread model of each implementation. They are important as they can be used to determine the context switch time within a mobile device, and how it compares to that of a full scale desktop machine. Also, in Section 3 we show how small a memory footprint we can actually achieve by removing various unnecessary classes.

1. Mobile Devices

When the term mobile device is used, most people think primarily of mobile phones as opposed to anything else. This should not always be the case, and more powerful form factors such as Personal Digital Assistants (PDA) also fall under this category. Digital cameras and MP3 players can also be considered a form of mobile device; wireless network capabilities now being available for digital cameras [8]. The boundary between these is becoming vaguer however. Cameras are being integrated into mobile phones as well as other capabilities, leading to the evolution of the smart-phone [9] – PDA and phone combined. The smart-phone is not only used to make telephone calls, but to play games, send and receive email, browse the Internet – as well as having the integrated digital camera.

Considering the capabilities of most modern smart phones, it is not uncommon for a device to have a 400 MHz processor and at least 64 Mbytes of RAM. Adding a small sized memory card of 2 Gbytes costs virtually nothing, and provides a large amount of storage space in a very portable device. This specification is easily up to the standard of the common desktop around 6-7 years ago.

The real motivator behind mobile technology is wireless communication technologies, be it Wi-Fi, Bluetooth, or GPRS mobile phone networks. The original mobile phone transmission technology of GSM has been upgraded to 3G communication technologies, allowing fast connection speeds on the move as well as in a wireless hotspot. Work is also being carried out to take us further beyond 3G.

1.1 Flavours of J2ME

Of course, mobile devices are much more resource constrained than a full desktop system, where GHz of processor and Gbytes of memory are more the norm than the MHz and Mbytes of small factor devices. Utilizing existing development platforms as is becomes difficult, and therefore Sun have provided a specification for mobile and embedded devices, Java 2 Micro Edition. J2ME comes in a variety of different platforms, each aimed at various applications and device configurations [10]. Figure 1 illustrates the various Java platforms and how they relate to one another [11].

The two platforms on the left are those aimed at the more traditional desktop and server systems – Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE) respectively. The two platforms on the right are the various different incarnations of J2ME. The Connected Device Configuration (CDC) is aimed at higher end devices, such as smart phones and set top boxes. Its various profiles – Foundation, Personal Basis and Personal – add further packages and functionality upon its predecessor. The Personal Profile actually gives a platform comparable to J2SE 1.3.1, with the various deprecated methods removed to condense the final size of the architecture.

The lowest level Java illustrated here (there is also Java Card which is even smaller than this) is the Connected Limited Device Configuration (CLDC), which is aimed at mobile phones and embedded systems, ones with less than 512 Kbytes of memory [10]. This is a very different platform than the one found in J2SE. One of the major differences is the virtual machine that runs Java on small devices. KVM stands for Kilobyte Virtual Machine, and is a significantly cut down version of a standard Java VM. A KVM is often installed on a chip within a device [3], as opposed to the software based JVM on the desktop PC.

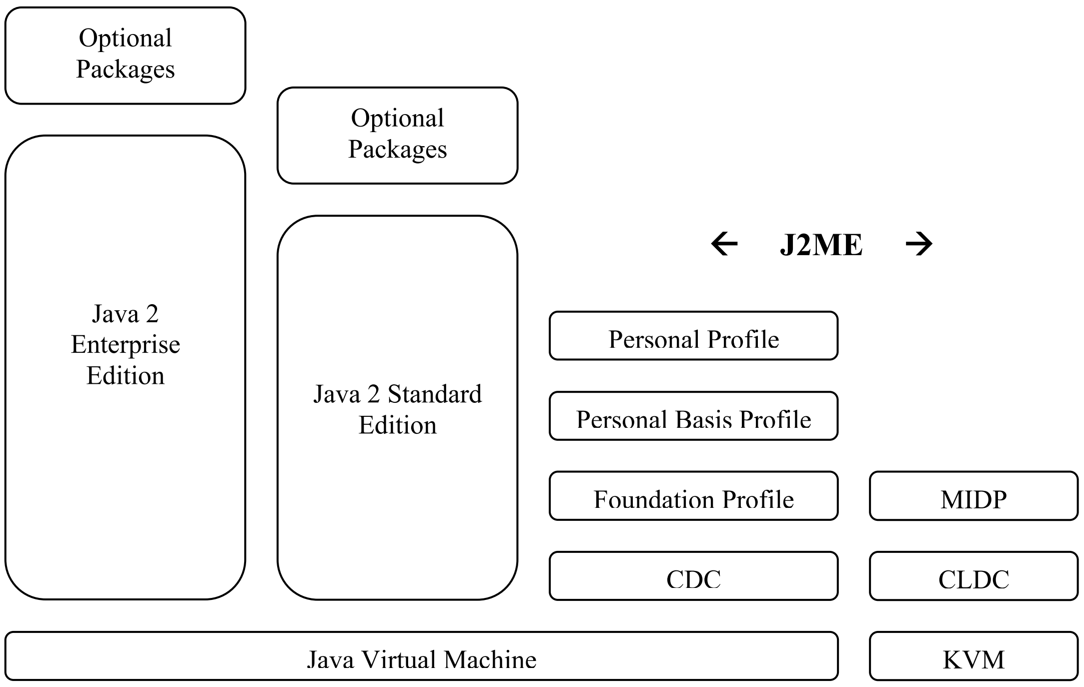

Figure 1. Java Platforms

The Mobile Information Device Profile (MIDP) lifts the basic, console based, CLDC platform allowing graphical user interfaces and other richer media content. Aimed specifically at the mobile phone market, it has now become uncommon to find a phone that does not implement this profile in some manner. As different device manufacturers implement the profile in different ways, Sun Microsystems has only provided guidance as to what this specification should include. As such, it is not uncommon to find different phones displaying slightly different interfaces for the same MIDlet; or even not running a MIDlet that another device can.

MIDlets are also packaged up into a Java Archive (JAR) file for distribution. This means that all developed class files must be included within the application itself. There is no such property as a class path within a MIDP environment, meaning that the same class may be replicated across several different application JAR files, with no method of sharing between them.

CLDC and MIDP do not utilize the Java Native Interface (JNI) so actually updating the standard thread model in Java for a more efficient approach is not possible. This is itself justified as the aim is to have a package that can be used on a multitude of mobile devices and embedded systems without having to recompile device specific code for each, an approach more suitable for a C++ CSP approach. Combined with the possibility of embedded Java chips within a mobile phone, this upgrading possibility is not the best option. Some mobile phones also have very limited available SDK's for developing specific libraries, and many do not even have this possibility, being hardware based only.

Because of this deep relationship between Java on mobile phones (MIDP) and Java on other embedded systems (CLDC), it becomes only necessary to initially aim a micro edition of JCSP at the CLDC configuration.

2. Converting JCSP

Before converting JCSP to operate on J2ME, a first analysis as to the general package differences between CLDC and J2SE needs to be undertaken. An investigation of the specification of CLDC shows a number of changes that will need to be made to JCSP to allow it to first compile, and only these are discussed here. Many other differences exist between CLDC and J2SE but these are not relevant to this discussion.

2.1 Removal of Certain Packages

The first package that can be removed from JCSP for the micro edition is the active components in the `jcsp.awt` package. As previously mentioned CLDC is console based only, and although MIDP does provide user interface components these are in no way related to the basic AWT components of J2SE. Thus, it would be necessary to create a new range of active components for MIDP, which at the moment is left for future work.

The second major package removal relates to the network functionality provided with JCSP Network Edition. CLDC uses a completely different technique for creating underlying connections that relies on a governing factory class called Connection. Specifically, there is no such usable object as `Socket` and `ServerSocket`, meaning that the capabilities of the `jcsp.net` package can not be compiled under CLDC in its current form. There are other reasons for removing this package also, which we will explore in Section 2.3. For now however, the different communication model is enough justification for its removal.

Two other packages, `jcsp.win32` and `jcsp.demos`, are also removed. These are not required parts of the overall framework, `jcsp.win32` being platform specific and `jcsp.demos` being demonstration programs some of which require the already removed `jcsp.net` and `jcsp.awt` packages.

2.2 Differences in the Thread Model

Another major difference when trying to implement the JCSP framework is the thread models implemented in J2ME when compared to J2SE. Obviously CLDC will have some form of cut down thread model, but of particular importance is the removal of background (or in Java terms Daemon) threads. Within JCSP the underlying threads of the processes in a parallel construct are background threads, so for the CLDC implementation of JCSP this feature needs to be changed. Therefore a call to exit the system (`System.exit(0)`) must be made whenever the main process has ended in case any residual threads are still in operation.

As mentioned earlier, deprecated methods are removed from J2ME implementations, so such (dangerous) capabilities as pause and resume are not available. These methods were never used in the underlying functionality of JCSP, but some other functionality has been removed in CLDC. The method to set the name of a thread in particular is no longer available, and was used in the underlying threads in JCSP. However, this value can be set in the constructor of the Thread object, and this has been done instead.

Another omission is the `ThreadGroup` object, which is used in JCSP to determine the priority of processes within a `PriParallel` structure. Instead, we use the `Thread` object.

2.3 Unavailable Interfaces

A number of interfaces are no longer available in CLDC. Of these, two are used extensively within the remaining JCSP packages and classes. The `Serializable` interface allows objects to be converted into bytes for either storage or transfer across a communications medium. This interface is a central feature required for the `jcsp.net` package, another reason for its removal. As well as this, a high proportion of the remaining classes implement `Serializable` for this very reason of network transference. These classes have been altered to remove the use of this interface.

The second interface of note is `Cloneable`. This interface allows deep copies (i.e. full copies, not references) of objects to be made by calling a clone method; although clone itself may simply return a reference to the actual object if so desired, thereby only providing a shallow copy. CLDC does not support object cloning – there being no `Object.clone()` method that all objects can use. Cloning is used by the various buffers in the `jcsp.util` package. The decision to take here is whether to remove the Buffered Channels from JCSP completely (along with the `jcsp.util` package) or to create a `Cloneable` interface of our own – this latter method only requiring a clone method to be declared within a new interface. The second choice has been taken in this circumstance as it is fairly simple, and allows the useful functionality of Buffered Channels to remain.

2.4 Reduced Collections Framework

The Collections Framework in Java defines a number of data structures that can be used to store objects for later retrieval. J2SE has a large range of varying collection types, from Array Lists (essentially a dynamic array) to Hash Maps. CLDC obviously cannot support many stored objects due to its smaller size and generally smaller amount of available resources. In particular, the availability of the various hashed data structures has been removed. The Hash Table was used within JCSP to store various objects (threads in fact) in the underlying parallel structures, so this data structure has been replaced by a Vector (another simple dynamic array type). This does mean that it is harder to obtain a specific thread in the parallel object, but as less threads should (in theory) be running, this will hopefully not be a major impact on performance.

2.5 No Number Class

The final necessary changes involve the plug and play components. Some of these utilized `Number` objects (the parent object of the `Integer`, `Float`, `Double`, etc, primitive data wrapper objects). CLDC does not support this class, so `Integer` is used instead. This is justified in that the plug and play components provided with JCSP converted the `Number` object into an `Integer` object before using them anyway.

3. Reducing the Memory Footprint

Once all the previously described alterations have been made, the remaining JCSP packages compile quite easily, resulting in a set of classes that can be used on a CLDC based platform, such as a Java enabled mobile phone. The final package size is still quite large however, around 163 Kbytes on a device that is meant to have less than 512 Kbytes. On these more memory constrained devices this is obviously going to become a problem, but within a MIDP environment this can be much worse. As previously mentioned, a MIDlet requires all the necessary classes to be packaged together into a single JAR file – there being no such property as a class path. If two JCSP enabled MIDlets are on the same mobile device, then the JCSP package is effectively replicated twice – once for each MIDlet – thereby taking up over 300 Kbytes of memory. Therefore it can be surmised that removing classes that are not always used becomes favorable. These classes can then be added to any implementation that requires them.

3.1 Removing Classes

Specifically, there are four groups of classes that can be removed from the framework and still provide the main functionality expected in JCSP. These four groups are:

- *Plug and Play Components* – A whole package can be removed in the form of Plug and Play. The classes in this package, although useful, are not required for every application, and can be added as required. This could be viewed as quite a major loss, but if included as an optional addition then this is not entirely the case.
- *Connections* – Connections hide a two channel structure (input and output) from the developer. The basic premise is to provide a method that appears very similar to a client-server connection over a network.
- *Rejectable Channels* – These allow input ends to reject messages sent over a channel. These are usually used extensively in the `jcsp.net` package already removed.
- *Call Channels* – These channels provide *synchronising* method calls between processes (over which data may flow in both directions). Both processes must agree to the call – in the same way as both processes must agree to communicate on a channel. The accepting process may wait for it as part of an `Alternative`. They simplify many kinds of transaction that would otherwise need a sequence of normal channel communications and they have the potential for increased efficiency. However, distributed versions are not yet available and, for now, they are removed. [Note: *Call Channels* should not be confused with `synchronized` method calls in standard Java.]

With these classes removed, the size of JCSPme is around 90.3 Kbytes, which is a significant reduction in size from the 163 Kbytes of the first compilation, and a dramatic decrease in size from the 1.06 Mbytes of the full JCSP package. This could possibly be reduced further if other classes were removed, such as Buffered Channels and primitive integer channels. For now however, we will leave the remaining classes in place.

4. Testing

Most recently, comparisons between various CSP implementations have been performed using the *CommsTime* test [12]. This involves four basic processes – prefix, delta, successor and a custom consumer – connected together to create the Natural numbers in sequence. The time taken to produce a number is measured to provide an idea of the context switch time on each platform. In 2003, using a 667 MHz Celeron processor with 256 Mbytes of RAM, JCSP running under Java 1.4 recorded a time of 230 microseconds to produce a number (occam scored a far more impressive 1.3 microseconds). To test the Micro Edition of JCSP, two different platforms were used. Firstly a PDA/Smart Phone running Windows Compact Edition 4.2 with a 416 MHz Intel PXA272 processor and 64 Mbytes of memory; and secondly a Sony Ericsson S700i mobile phone. The smart phone was capable of utilizing a number of different virtual machines, as well as a .NET implementation of CSP currently under development. The results are presented in Figure 2. A standard desktop machine running normal JCSP is given as comparison, as well as the smart phone running a CDC Personal Profile with full JCSP support.

As these results show, the *CommsTime* on such a reduced platform is still quite reasonable when compared to the Celeron 667 MHz results. Something to point out here is that although the Smart Phone has a 416 MHz processor, in general it tries never to operate at such a speed to preserve battery lifetime. The two IBM JVMs have differing results, but this is probably due to the MIDP smaller footprint and better efficiency thereof.

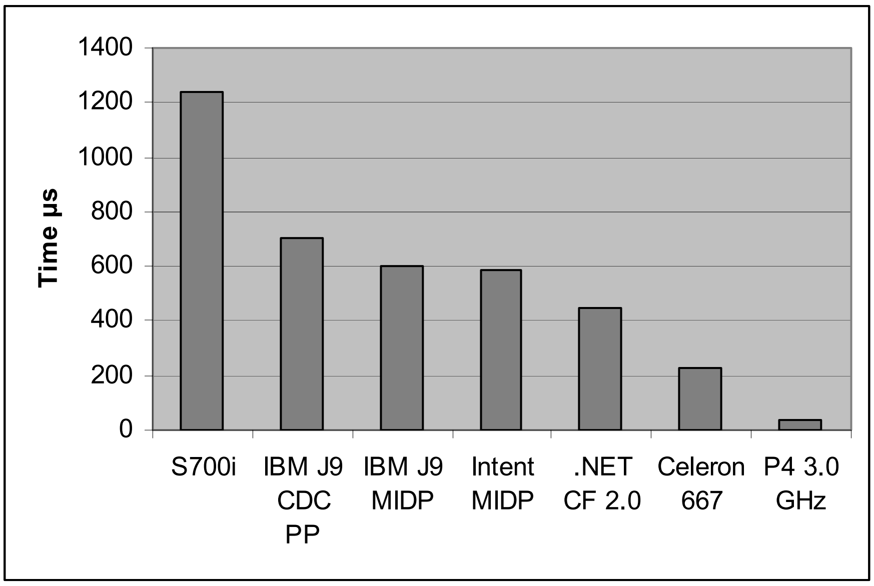

Figure 2. *CommsTime* Results

The Intent VM is common on many Windows CE based smart phones on the market, and is claimed to be the fastest such VM available for smart phones (`http://tao-group.com`), apparently a true statement looking at these results. The .NET result is, however, surprisingly better than any of the Java platforms. The most likely explanation for this is Microsoft's knowledge of the underlying architecture of the device and its operating system, something unlikely shared by IBM and Intent. This is interesting in that it shows that there is possible room for improvement within the Java implementations if more information can be gathered as to how the threading system is handled on Windows CE devices.

A question that could be asked is how much does a thread cost within CLDC and MIDP? This is difficult to answer, and the likely results will not be comparable as each VM implementation is specific. Memory allocated to the virtual machine can vary from device to device, and how to compare a MIDP implementation built upon a chip with one implemented in software is questionable. The results initially gathered here give us a good idea as to how likely a multithreaded application will perform on a small scale device.

5. Future Work and Conclusions

Much further work can be undertaken to extend JCSPme to provide functionality closer to the full package. Of particular interest is an Active MIDlet UI and providing networked capabilities. By doing this, a definite move into the mobile device market area can be made using JCSP, providing high level abstractions to develop complex systems.

5.1 Active UI for MIDlets

As JCSP has an AWT package, and work has been done on a Swing equivalent, it seems feasible that a collection of active components for MIDlets be achieved. MIDP provides a Liquid Crystal Display User Interface (LCDUI) API as an alternative to AWT, and it is a very simple set of very basic components. Developing these might actually overcome problems in developing MIDlet applications in general, which have a very different development model to standard applications, having more in common with the Java applet model.

5.2 Persisting Objects

The lack of the `Serializable` interface is a major hindrance to the inclusion of network functionality to JCSPme, although it may be overcome by utilizing object persistence methods instead. These generally require the developer to determine how an object should be converted into an array of bytes. This is different from the methods generally used in Java; there are no such objects as object streams and object inputs and outputs. As the basic types (`String`, `int`, etc) can easily be converted into byte arrays, the belief is that any other object should also be, as long as all its component parts are basic types or can be persisted themselves. This would require any object requiring persistence to implement their own persist and restore methods, unlike object serialization which is hidden. This will not always be the case as even in standard Java some objects contain elements that are not of a primitive type (such as a pointer to a database record).

5.3 Networked Channels

Networked channels can quite easily be developed as is, if primitive data types are used instead of objects. By using the `Connection` class and creating TCP/IP connections, architecture not unlike standard JCSP can be developed. If a method of persistence is also incorporated then it will be possible to send full scale Java objects across these networked channels. This would take the JCSPme platform a long way to becoming a framework for developing distributed systems on mobile phones.

The same can not be said for a mobile process based system at present, as CLDC does not provide any method to load classes dynamically, due to resource constraints. However, as the platform matures it is likely that some form of customizable class loading system will emerge.

5.4 Gaming

A possible use for JCSPme is the development of games using a different approach to standard development. MIDP does provide a game API unlike other Java incarnations, `javax.microedition.lcdui.game`. Game engines usually follow a basic game loop [13] of get user input, process logic, update elements and update display. Taking a process and channel based approach to this may involve processes acting like game elements and thereby processing their own logic internally and sending messages to a centralized game engine process which sends back any necessary status updates. Rendering to a display has already been implemented in the full scale JCSP package in the form of Active Components, particularly `ActiveCanvas`. This is a definite area of interest as a possible simplification of game design.

5.5 Conclusions

The results gathered for *CommsTime*, coupled with the ever expanding mobile phone market, show that mobile devices are becoming a worthy area of research. When considering the possibilities of Ubiquitous Computing [14] and the nature of the mobility of such environments, JCSP can be seen as a possible solution to some of the problems that must be overcome for the software infrastructure of Ubiquitous Computing. Indeed, the π-calculus [15] has been mentioned in the literature [16], and JCSP provides us with the basic functionality of mobile processes and mobile channels in a multi-platform environment, although some work still needs to be carried out.

These performance results may be improved on even further by trying to refine the behavior of various components in the framework to make them more suitable for the constrained resources, and testing on other mobile phone platforms needs to be carried out to determine the overall efficiency of any changes made. Other tests need to be carried out (Stressed Alt for example) to stress test the overall capabilities of the J2ME on these various platforms, and really determine how well JCSPme can operate. Testing must also be carried out on various real-time hardware boards, and as Java is used more and more to teach the basics of real-time software engineering, JCSPme could be used to demonstrate certain concepts of concurrency.

As mobile phones progress to becoming smart phones, providing more resources to the developer, the future of MIDP becomes unclear. Much investment has been made using MIDP as a gaming platform for mobile phones, with various companies selling games over the Internet or by other means such as text messaging. To say MIDP will disappear is probably not justified completely, but as phones become more powerful the likelihood is that more extensive Java platforms, such as CDC, will become more prominent and MIDP will disappear into the background; or evolve and merge into the higher specification implementations. This would allow the full use of the JCSP package as is, and IBM already provides JVMs for a number of mobile device platforms. This may make JCSPme unnecessary for the mobile phone market, but as Java is pushed more and more as a possible real-time platform, JCSPme may find a home in the embedded systems market.

References

[1] A. Corsaro and D. C. Schmidt, "Evaluating Real-time Java Features and Performance for Real-time Embedded Systems," presented at the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 2002.

[2] K. Nilsen, "Adding Real-Time Capabilities to Java," *Communications of the ACM,* 41(6), pp. 49-56, 1998.

[3] G. Lawton, "Moving Java into Mobile Phones," *IEEE Computer*, 35(6), pp. 17-20, 2002.

[4] P. H. Welch and J. M. R. Martin, "A CSP Model for Java Multithreading," in P. Nixon and I. Ritchie (Eds.), *Software Engineering for Parallel and Distributed Systems*, pp. 114-122. IEEE Computer Society Press, June 2000.

[5] P. H. Welch, J. R. Aldous, and J. Foster, "CSP Networking for Java (*JCSP.net*)," in P. M. A. Sloot, C. J. Kenneth Tan, J. J. Dongarra, and A. G. Hoekstra (Eds.), *Proceedings of Computational Science – ICCS 2002, Lecture Notes in Computer Science 2330*, pp. 695-708. Springer Berlin / Heidelberg, 2002.

[6] G. H. Hilderink, J. F. Broenink, W. Vervoort, and A. W. P. Bakkers, "Communicating Java Threads," in A. W. P. Bakkers (Ed.), *Proceedings of WoTUG-20: Parallel Programming and Java*, IOS Press, Amsterdam, The Netherlands, 1997.

[7] G. H. Hilderink, J. F. Broenink, and A. W. P. Bakkers, "A Distributed Real Time Java System Based on CSP," in B. M. Cook (Ed.), *Proceedings of WoTUG-22: Architectures, Languages and Techniques for Concurrent Systems,* IOS Press, Amsterdam, The Netherlands, 1999.

[8] G. D. Hunt and K. I. Farkas, "New Products," *IEEE Pervasive Computing*, 4(2), pp. 10-13, 2005.

[9] S. J. Vaughan-Nichols, “OSs Battle Smart in the Smart Phone Market,” *IEEE Computer*, 36(6), pp. 10-12, 2003.

[10] J. White, “An Introduction to Java 2 Micro Edition (J2ME); Java in Small Things,” in *International Conference on Software Engineering, Proceedings of the 23rd International Conference on Software Engineering,* pp. 724-725. IEEE Computer Society.

[11] “CDC: An Application Framework for Personal Mobile Devices,” Sun Microsystems Inc. June 2003. Available at http://java.sun.com/j2me.

[12] N. C. Brown and P. H. Welch, “An Introduction to the Kent C++CSP Library,” in J.F. Broenink and G. H. Hilderink (Eds.), *Communicating Process Architectures 2003 (WoTUG-26)*. IOS Press, Amsterdam, The Netherlands, 2003.

[13] D. Clingman, S. Kendall, S. Mesdaghi, *Practical Java Game Programming,* p. 25. Charles River Media, Inc., Hinghan, MA. 2004.

[14] M. Weiser, “The Computer for the 21st Century,” in *Scientific American*, pp. 94-104. September 1991.

[15] R. Milner, *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[16] T. Kindberg and A. Fox, “Systems Software for Ubiquitous Computing,” *IEEE Pervasive Computing*, 1(1), pp. 70-81, 2002.