BUILDING DISTRIBUTED HETEROGENEOUS SMART PHONE JAVA APPLICATIONS AN EVALUATION FROM A DEVELOPMENT PERSPECTIVE

Ali Kattan, Rosni Abdullah, Rosalina Abdul Salam and Sureswaran Ramedas

School of Computer Science
Universiti Sains Malaysia

kattan@cs.usm.my,
rosni@cs.usm.my,
rosalina@cs.usm.my
sures@nav6.org

ABSTRACT

The advances in mobile phone technology have enabled such devices to be programmed to run general-purpose applications using a special edition of the Java programming language. Java is designed to be a heterogeneous programming language targeting different platforms. Such ability when coupled with the provision of high-speed mobile Internet access would open the door for a new breed of distributed mobile applications. This paper explores the capabilities and limitations of this technology and addresses the considerations that must be taken when designing and developing such distributed applications. Our findings are verified by building a test client-server system where the clients in this system are mobile phones behaving as active processing elements not just mere service requesters.

Keywords: Smart phone, Java ME, MIDlet, Distributed applications, Client server.

INTRODUCTION

The impact of the wide spread of mobile phones with relatively decreasing costs has been the focus of many studies (Bagchi, Kirs, & Lopez, 2008). This diffusion is far more ubiquitous when compared to the spread of PCs. 2007 was a crossover year where smart phone sales exceeded laptop computers sales.

Mobile phone customers represent a staggering 3.3 billion in the subscriber market in total (Want, 2009).

Mobile phone processing power, as well as their storage capacity, have increased dramatically during the past few years (Knyziak & Winiecki, 2003). Architecturally, a mobile phone can be divided into two basic processing components: the communication processor and the application processor. In smart mobile phones, the latter becomes a computationally powerful computer in its own right, capable of running general-purpose applications (Want, 2009). Buying a PDA no longer makes sense since smart phones have absorbed the functionality of such devices causing a noticeable decline in the PDA market (Kozel & Slaby, 2008). From now on, the term "*mobile phone*" will be used to refer to the smart mobile phone.

A special edition of the well-known Java programming language, known as Java ME (Java Micro Edition) became the common ground for developing applications for such phones (Xu, 2006). It provides relatively easy opportunity to make mobile phone applications including data communication access utilising either some common protocols such as HTTP/HTTPS or socket-based communication (Kozel & Slaby, 2008).

Fast Internet access via UMTS (3G), EDGE, or WiFi technologies would become a standard low cost service provided to any mobile phone network subscriber. The relatively slow response time for the mobile phone applications that used to utilise the former CSD and GPRS technologies is something of the past (Knyziak & Winiecki, 2003). This would open the door for a new breed of phone-based distributed applications that are to be integrated into larger existing computing infrastructures (Mock & Couturier, 2005).

RELATED WORK

In general, mobile phone technology is relatively young and has yet to expand. There have been numerous analysis and evaluation studies targeting different aspects of this technology. One of the main goals of such studies is to help mobile phone designers, manufacturers, as well as mobile software developers to identify limitations and constraints in order to produce better solutions.

Heo, Hamb, Park, Song, and Yoon (2009) conducted a study on mobile phone usability evaluation methods and heuristics. By deriving some desirable mobile phone features that served as a reference point, they proposed a framework for evaluating the usability of a mobile phone based on a multi-

level, hierarchical model of usability factors. Other studies were more specific in studying the usability of mobile phones in more specific areas like mobile commerce. Chang and Chen (2005) presented a theoretical foundation to qualify the use of the mobile phone as a client platform for mobile commerce. Desirable mobile phone system features were listed assuming an ideal client platform and then these discussed in terms of the shortcomings of mobile phones available at the time.

The development of distributed mobile phone applications was made possible owing to the increased processing power, programmability and high-speed communication capability of such devices. Knyziak and Winiecki (2005) studied the suitability of Java 2 Micro Edition or J2ME (lately known as Java ME) in distributed measurement systems. The client mobile phone was programmed to behave as a control and data presentation device in a distributed client-sever measurement system. Their study addressed diverse client-server issues like communication delays and transmission time. They indicated that Java-enabled mobile phone's cross-platform interpretability, i.e. device heterogeneity, and the client-side computation ability are amongst the strongest features that make such devices the most powerful and promising for use in this field. Mobile phone applications are no longer restricted of being mere web-server clients (Hasegawa, Nakamura, Higushima, Kawasaki, Nakashima & Sato., 2008). Mobile phone distributed applications are gaining potential in diverse areas with Java ME as the most important programming language that can facilitate the development of such applications (Yan & Liang, 2009).

The literature reviewed here forms the reasons behind our study. With the application developer in mind, the building of such distributed mobile phone Java applications is studied in order to indicate the capabilities and limitations that are involved in such task.

PROGRAMMING MOBILE PHONE DEVICES

It is essential to have an idea about the Java mobile phone framework environment in order to understand the nature of mobile phone application development. Writing applications for mobile phone devices is totally different from writing applications for PCs (Mazlan, 2006).

Sun Micro Systems Java programming language is one of the most popular languages used to program mobile phone devices. Java promotes a unified development and execution platform regardless of the underlying hardware.

The different Java frameworks are shown in Figure 1. The mobile phone Java edition, namely J2ME is basically a cut-down version of the Java 2 Platform Standard Edition (J2SE) that is tailored to suit mobile phone devices. In spite of being a cut-down version, the language seems to be adequate enough to enable the building of some interesting applications in vital areas like biomedicine (Takeuchi, NoritakaMamorita, FumihikoSakai, & Ikeda, 2009).

Sun Micro Systems divides mobile phone devices into two categories: high-end representing PDAs and low-end representing mobile phones and entry-level PDAs. However, we believe that such terminology is no longer valid since many of the latest smart phones fall under the second low-end category. The processing power of the former is usually 32-bit while the latter is limited to 16-bit which is the interest of this paper since they are more ubiquitous. The framework is composed of a set of basic classes that are built into the mobile phone's firmware in addition to a set of optional packages that can be loaded dynamically into the phone memory based on the application's needs.

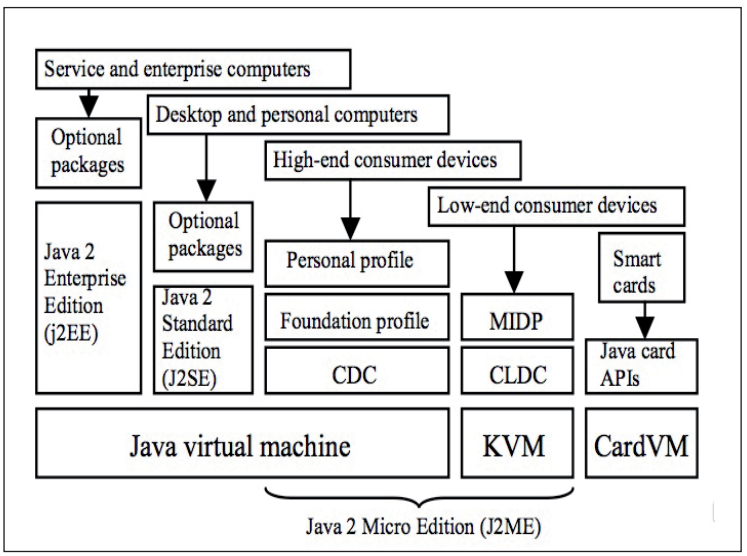

Fig. 1. The Different Java Frameworks

Java Community Process (JCP) represents an alliance of participating members (JCP, 2009b) with most of the major mobile phone manufacturers and mobile phone service providers being involved (JCP, 2009a). JCP is responsible for laying out the specifications for mobile Java. These are introduced in the form of JSRs (Java Specification Requests) to provide common implementation guidelines for mobile phone device manufacturers and service vendors to undertake. Such specifications are flexible to allow extension and promote

compatibility. Despite this, some manufactures have followed custom trends to add more functionality to their line of mobile phone devices. Unfortunately this would sometimes violate the promoted compatibility between different phone brands and might result in some unanticipated Java application bugs (Klingsheim, Moen, & Hole, 2007).

Connected Limited Device Configuration

The configuration that defines small, mobile phone devices is known as the Connected, Limited Device Configuration (CLDC) (Sun Microsystems, 2009a). These devices will have a system memory between 160 and 512 Kbytes and use the Kilobyte Virtual Machine (KVM) (Helal, 2002a) though such a memory based distinction is no longer valid. CLDC 1.1 (JSR 139) is the current version and provides two basic packages for network support:

- java.io package, provides classes for input and output through data streams, which includes reading primitive data type streams and byte array streams, and
- javax.microedition.io, provides classes for the Generic Connection framework, which includes creating connections (TCP based) and datagrams (UDP based).

However, connections established using the above classes rely on blocking IO methods to achieve its functionality. It lacks the new IO non-blocking methods of its desktop counterpart, namely J2SE. In addition, Object Serialisation is not supported. Thus, with the exception of strings, only primitive data types can be interchanged through a network connection. Another noticeable limitation is the lack of Java RMI (Remote Method Invocation) support under CLDC (Mock & Couturier, 2005). Both Object Serialisation and RMI could play a fundamental part in facilitating the development of distributed Java applications (Al-Jaroodi, Mohamed, Jiang, & Swanson, 2003).

Mobile Information Device Profile

On top of the CLDC lies another set of classes that are referred to as profile. These classes are also part of the phone's firmware, known as MIDP (Mobile Information Device Profile). MIDP extends CLDC's functionality further (Klingsheim, Moen & Hole., 2007). MIDP 2.0 (JSR 118), which is an enhancement over the former MIDP 1.0. It is currently the most commonly used profile in mobile phones.

Most of these enhancements address security and privacy issues due to the added networking capabilities and the increased functionalities of the device

(Klingsheim, et al., 2007). The profile does not allow, for security reasons, dynamic class loading from sources different than its own JAR file (Mock & Couturier, 2005). MIDP lacks a JDBC (Java Database Connectivity) component so direct access to databases from a mobile phone is not possible. The recent development in mobile phone hardware made the development of such JDBC driver possible as part of the MIDP (Hopfner, Schad, Wendland, & Mansour, 2009).

Optional Packages

Many optional packages can be included with the application based on need during compilation and they are loaded dynamically during run-time. These are also specified under JCP as JSRs (Klingsheim, et al., 2007). These optional packages are usually tied to the provision of certain hardware features within the mobile phone device itself. For instance, the Location API 2.0 package contains classes that enable the use of the device's GPS (Global Positioning System) circuitry for GPS-enabled mobile phones (Abramov & Rogov, 2009). Figure 2 shows some of such optional packages.

The mobile phone manufacturer should state clearly which of these are supported to facilitate application development and testing (Mazlan, 2006). In addition to the optional packages, we found that vendors sometimes would achieve extra functionality or boost performance by providing their own customised versions of these standard packages as will be discussed later.

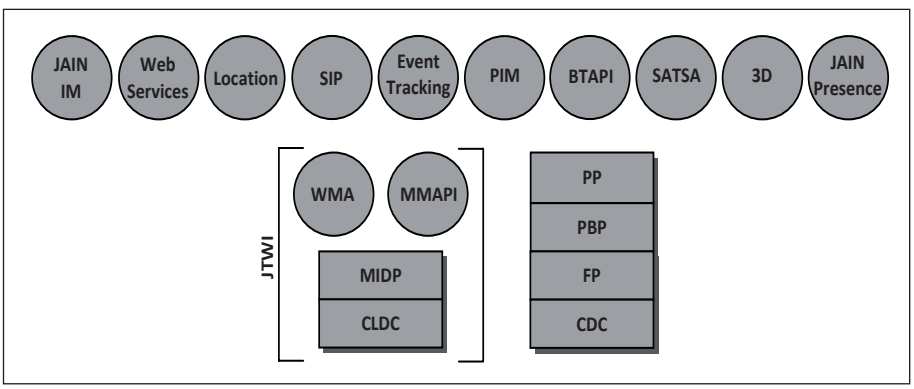

Fig. 2. J2ME Optional Packages

MIDLETS

A MIDlet is a Java ME mobile application. MIDlets are analogous to Java Applets known under the J2SE framework. The mobile phone has its own

dedicated OS, namely the AMS (Application Management System). AMS is responsible for the loading, starting, pausing, and destroying of MIDlets (Marejka, 2005). Most of the recent mobile phones have a more complete and multi-threading capable OS like Symbian™ (Jode, 2004).

MIDlet Lifecycle

In order to develop distributed Java-based mobile phone applications, it is essential to understand that MIDlets have different execution states (Marejka, 2005). Once the MIDlet files are installed in to the phone's memory, the user can usually run the MIDlet by selecting it from a menu. The AMS would create an instance of this MIDlet and prepare it for execution. The MIDlet has three different states: Paused, Active, and Destroyed. All of these states are reflected by special methods within the MIDlet's code (Mock & Couturier, 2005; Helal, 2002) and as shown in Figure 3. Developers must override these methods in order to include their intended code.

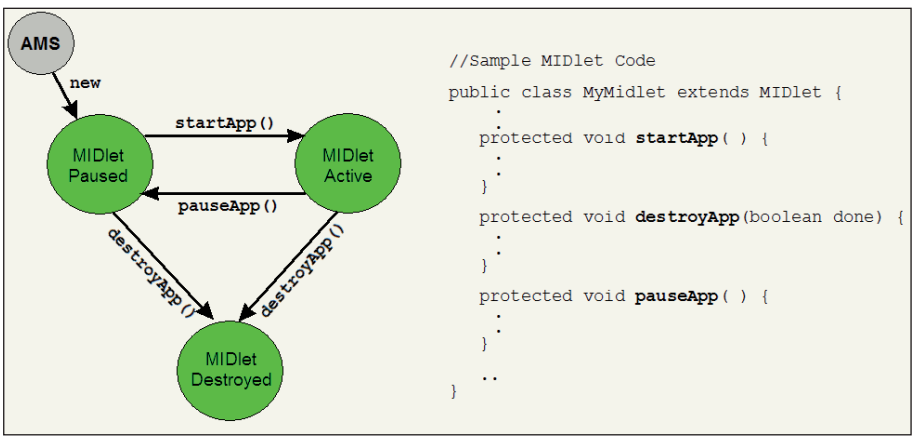

Fig. 3. MIDlet Lifecycle

The Active state is where the MIDlet is doing its intended functionality. The paused state is the state where the MIDlet would be in the event of an incoming call or other high priority event that requires the MIDlet to pause. The MIDlet in such case would release its resources and wait untill the high priority event is completed, where by then it can ask the AMS to resume its functionality. Finally the destroyed state is the state where the final housekeeping is done to release any used resources and save any data prior to MIDlet termination. MIDlets can save persistent data on the phone memory using a system known as RMS (Record Management System) (Jode, 2004). Once the MIDlet instance is terminated, it ceases to exist from the working memory of the device. However it may keep the RMS saved data for use in the next run.

MIDlet Development

Sun Micro Systems have provided a special SDK (Software Development Kit) that makes use of the existing J2SE compiler to develop MIDlets. Java Wireless Toolkit for CLDC can be used to develop, test and debug mobile phone applications. It has a special set of emulators that will mimic a mobile phone environment on a PC. The developer would use his/her preferred text-editor or integrate the toolkit with an IDE (Helal, 2002b) to edit the program code since it is not provided along with the toolkit.

Sun's toolkit represents a generic platform to develop mobile phone applications without targeting a specific mobile phone brand. Java promotes the concept of "write once run any were". Unfortunately, this is not totally true when it comes to mobile Java applications. To be able to access the device's specific features and avoid compatibility issues that might exist between different mobile phone brands, special tailored versions of this toolkit are being offered by the device vendors (Klingsheim, et al., 2007; Helal, 2002). These customised toolkits would include special packages that augment the original set. In addition, the emulators are extended to emulate actual commercial phone sets and not just generic virtual emulators like those provided with Sun's toolkit. The developed MIDlets must be re-compiled and tested using those customised toolkits to avoid possible bugs (Klingsheim, et al., 2007).

MIDlet Signing and Installation

MIDP 2.0 has introduced a new security model. In order to have a trusted MIDlet suite, the origin and integrity of the MIDlet must somehow be authenticated. This is accomplished by having the MIDlet suite signed using a public key infrastructure (PKI). It uses the X.509 PKI, an ITU-T standard (Klingsheim, et al., 2007).

Trusted MIDlet suites will be associated with a root certificate, which in turn is associated with a protection domain. The device vendor installs many of such root certificates on the device itself. The MIDlet suite should explicitly declare what permissions are needed. Such permissions must be a subset of the permissions given to the associated protection domain, otherwise the MIDlet suite installation will fail. The signing process is subjected to a fee by the root certificate party. Steps for requesting and using a code-signing certificate are depicted in Figure 4.

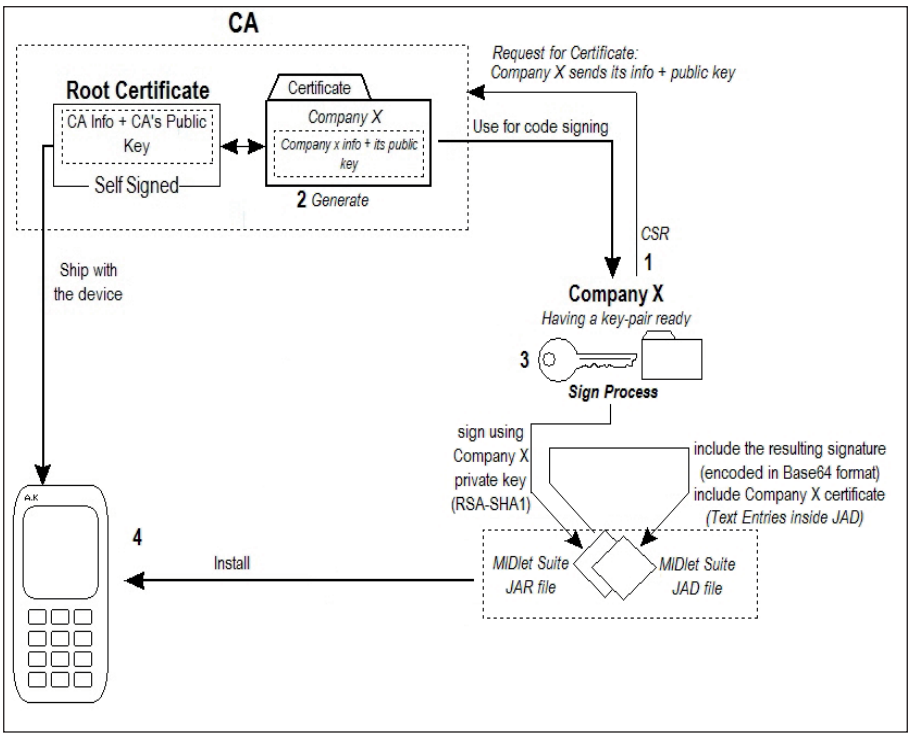

Fig. 4. MIDlet Signing Process

The MIDP 2.0 security model also provides the concept of protected API where access to those APIs is controlled by permissions. A protection domain is used to define a set of interaction modes and permissions, which grants access to an associated set of protected APIs. An installed MIDlet suite is bound to one protection domain. MIDP 2.0 supports at least one protection domain; the untrusted domain. A set of protection domains supported by an implementation defines the security policy.

Signed MIDlets could acquire special privileges. Such privileges are not granted to untrusted MIDlets and user intervention may be needed to grant them access explicitly. This could become an inconvenient process and the MIDlet's functionality could be crippled if it is not granted the right permissions since user intervention is not always possible.

The MIDlet suite is composed into two basic files: a JAR file and a JAD file. The JAR file is a Java standard JAR file including all the MIDlet classes, optional packages, and data files if any. The JAD file is a text-based file used to store MIDlet properties. The latter would also include the MIDlet signature. This is to be read by the mobile phone system upon MIDlet installation and running to determine which protection domain to use.

DISTRIBUTED MOBILE PHONE APPLICATION MODEL

A client-server model is basically a distributed system where processes in the distributed system are divided into two (possibly overlapping) groups. The request-reply behaviour is when the client requests a service from a server by sending it a request and subsequently waiting for the server's reply (Tanenbaum & Steen, 2002).

This model was adopted to develop a basic system that would promote testing the distribution and network functionality in the mobile phone devices and as seen in Figure 5. Though the model we had adopted looks like a traditional service-oriented architecture, the goal here is to let the clients do the actual processing and not the server. This is in contrast with some models where the clients request service from a web-based server and tasks in this case are completely performed on the server-side (Kozel & Slaby, 2008).

The client application, the mobile phone device in this case, would ask the server to provide a random (double) number. Upon receiving this number the client is to calculate the square of it and then send it back to the server for acknowledgment. Then the whole process is repeated again until terminated by the user.

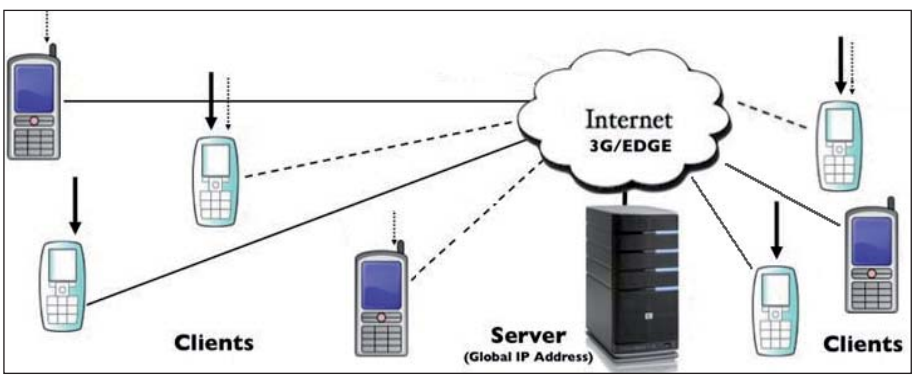

Fig. 5. Test Application

The server would receive incoming connections and dedicate a thread to each successful connection. Then a double number ranging between 0.0 and 100.0 is sent to the requesting client for processing. The server would wait for the client to send back the squared number for checking and acknowledgment. The connection is then terminated upon success or after a certain time-out period pending for client's response. The server will also monitor response times as well as connection outcomes.

BUILDING THE MODEL AND CONDUCTING TESTS

Seven different models of 3G/EDGE Java-enabled mobile phones from Nokia and SonyEricsson with line subscription from three different service providers were available for testing. All the clients were Java ME enabled mobile phones based on CLDC 1.1 and MIDP 2.0. The server code was written using J2SE 6 and the server application was hosted in a Windows 2003 server connected to the Internet system via global IP address.

The MIDlet development process is shown in Figure 6. Such process is analogous to that of building a Java client Applet using J2SE. Thus, and in comparison to building a Java client Applet, our first goal was to see how convenient it was to build the mobile phone client MIDlet for the system discussed section 4 in terms of the software development tools used, the debugging process, and the overall time/effort needed to complete the task. Since the environment lacks RMI, we had to rely on building basic client-server sockets to achieve the mentioned functionality. The connection code within the MIDlet was written as a thread to avoid application lock-ups since Java ME relies on blocking I/O methods. Other than that the Java ME framework supports a wealth of classes and methods that are comparable to what is being provided on the desktop version J2SE. A short summary of (J2SE) features versus Java ME features is shown in Table. 1.

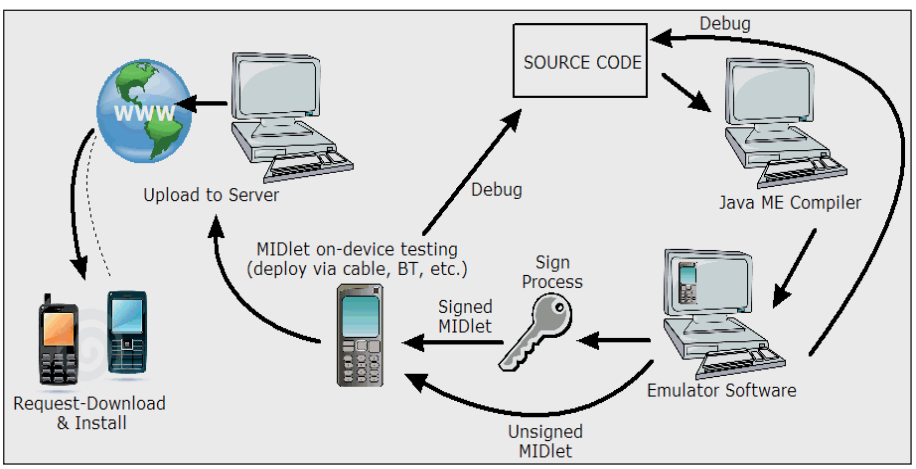

Fig. 6. MIDlet Development Process

Our MIDlet's code was written using standard CLDC 1.1 and MIDP 2.0, while avoiding any custom vendor packages. Compiling it using standard Sun SDK, Nokia's SDK, and SonyEricsson's SDK produced three versions of the same MIDlet. The development process on the three aforementioned SDKs was straightforward and shown in Figure 6.

We tested the client application using one of the included emulators with each SDK as well as an actual on-device testing. The on-device testing is an extra necessary debugging process in which the MIDlet is installed on an actual mobile device using a link cable, Bluetooth connection, or Infrared connection depending on the device type. Although there are some considerable differences in terms of GUI appearance between the three emulators from each SDK and the actual device, the basic functionality is still the same.

Table 1. Desktop Java 6 (J2SE) versus Mobile Java (Java ME)

Feature	J2SE	Java ME	Notes (Java ME)
Support for RMI	Yes	No	
Support for streamed connections	Yes	Yes	
Support for Object Serialization	Yes	No	Only strings are allowed
Support for full TCP/UDP	Yes	Yes	
Support for unblocked connections (NIO)	Yes	No	Threading can solve this problem
Support for multithreading	Yes	Yes	Total number of threads is limited by mobile device processing power

Apart from the on-device testing, the whole development process used a familiar set of software development tools in comparison to those used to build Java Applets using J2SE. However, testing has revealed that care must be taken to handle the mobile phone application Paused state, such that the device would be able to store and retrieve its current connection status. This was done in order to compare how the development of this application would go when targeting a specific vendor mobile phone. The Pause state was completely ignored by Nokia mobile phones and we had to modify the code substatially a bit in order to handle interruptions differently. Some vendors follow this trend since the application is actually still running in the background and there would be no need to pause it. This might have a negative effect on how the code is handled across a range of different devices.

Since our MIDlet uses many networking classes that fall under protected APIs, our MIDlet must use a protection domain that allows the use of those APIs. Eventually this would mean that the MIDlet must be signed to obtain the proper permissions. Obtaining a proper certificate to sign our MIDlet was costly since such certificates are usually subject to annual fees. Lacking a MIDlet-signing certificate we decided to upload the three unsigned MIDlet versions to a web-server and make them available for download via the mobile phone Internet browser.

At first we decided to install Sun's compiled version of our MIDlet on all of the seven test mobile phones we had. We faced issues in the application's installation process on all of them since an explicit user interaction was

required to grant the right for installing our unsigned MIDlet. Upon running the installed MIDlet, another user interaction was required once the MIDlet tries to establish an actual internet connection for the first time.

Table 2. Experimental Results for Seven Java ME Enabled Mobile Phones[a]

#	Internet Line	Location[3]	Establish Connection (ms)	Average Response (ms)	Establish Connection (ms)	Average Response (ms)
			Sun's SDK		**Vendor's SDK**	
1	3G	City A	414	21	388	16
2	EDGE	City A	523	43	497	39
3	3G	City A	494	34	479	32
4	EDGE	City B	447	28	421	25
5	3G	City B	376	24	459	23
6	3G	City C	389	21	364	19
7	EDGE	City C	431	38	387	24

[a] Permission was not granted to publish the exact mobile phone brand/model number
[b] Our server is located in City C.

During the running of the mobile phone application, we would simulate interruption by calling or texting the client mobile phones. The performance of each of the devices tested, in comparison to each other, was relatively similar in terms of connection time, response time, and the ability of recovering from network disconnections. Connection time is the time needed by the device to establish or re-establish, an internet connection. The response time is the overall time taken for a client to request a number from our server, calculate the square of it, and send it back to the server. A summary of our results is listed in Table 2.

With the development process in focus, this table provides merely a sample picture of the performance of our simple distributed application. Further analysis of this data is beyond the scope of this work.

Our tests have also revealed how the behaviour of the mobile phone Java application would differ from one device vendor to another due to the different ways of handling MIDlet's states and security measures by different vendors. A summary of such behaviours is listed in Table 3. For instance, since our MIDlet was unsigned, user permission was explicitly needed upon each Internet reconnection. This would occur once internet disconnection was encountered due to low signal coverage or once the phone's screensaver starts on some models causing an Internet disconnection. Disabling the phone's screensaver or setting it to start after a longer period of time solved the latter problem.

Table 3. Summary of Problems and Solutions

System Behavior	Cause	Solution	Total Affected Phones
Explicit user permission required to install	Unsigned MIDlet	Sign the MIDlet using CA certificate supported by the device	7
MIDlet installation failure	Using specific vendor's compiler on non-vendor machine.	Use either vendor's compatible SDK or Sun's Java ME SDK	7
Explicit user permission for internet connection	Unsigned MIDlet	Sign the MIDlet using CA certificate supported by the device	7
MIDlet Pause State handled incorrectly	Nokia Multi-tasking OS ignores the Pause State.	Modify the code to handle the Pause State differently	4
Explicit user permission to reconnect after connection failure (grant again)	Phone OS design/ Provider restriction.	Sign the MIDlet using CA certificate supported by the device	3
MIDlet crashed upon connection failure	Phone OS issue (certain model)	No solution found (most probably a system bug)	1
Internet disconnection once screensaver starts	Phone OS design	Disable screensaver or set to a longer period	4

Finally, we experimented with the other two versions of our MIDlet. We were not able to install Nokia MIDlet version on a SonyEricsson mobile phone or vice versa. Each vendor SDK compiled MIDlet ran only on the respective vendor devices. In general, the MIDlet ran somewhat faster than the standard Sun's version and as shown in Table 2. In addition, the behaviour was also a bit different upon interruptions. It was apparent that the vendor's SDK was optimised to run faster on the vendor's mobile phone hardware. In addition, these SDKs include extra packages that address platform specific features to gain extra performance or provide extra functionalities via additional proprietary methods.

CONCLUSIONS

Java enabled mobile phones definitely have the potential for running diverse distributed applications. There are many programming limitations in the mobile Java version when compared to its desktop counterpart. These

limitations include the lack of RMI support, lack of Object Serialisation, and the support of only blocking I/O connections. Yet, and due to the increasing processing power and memory capacity of such mobile phone devices, those limitations can be compensated to a certain extent, making mobile phones eligible candidates in any distributed computation that is part of larger computer infrastructures. However, it should be mentioned that this would require some extra effort from the developer.

The software development tools that are used to develop such mobile Java applications are analogous to those used to develop an Applet application on the desktop Java counterpart. With exception of a special on-device testing process, the same development cycle with similar tasks was required to build the mobile Java application.

The structure of a Java mobile phone application, namely a MIDlet, is affected by three main transition states during run-time that would consequently affect how the Java application is designed. However, not all mobile device vendors adhere to the same standards of run-time behavior and care must be taken to address this problem properly. Ensuring compatibility or at least portability across a wide range of the "said to be" compatible Java-enabled mobile phone devices would require some extra effort due to different manufacturer implementations of Java ME. This would definitely decrease the sought compatibility since such MIDlets compiled with vendor SDKs ran only on their respective vendor devices. In this case only actual on-device testing can confirm such compatibility.

MIDlets are to be signed with a special key obtained from a certification authority for a fee that is usually annual. This would grant the MIDlet special access permissions that otherwise must be granted explicitly by the user. MIDlet installation or MIDlet internet/network connection establishment is subject to such permissions. This is a bit of stringent security requirement that cripples the MIDlet functionality. These issues must be well considered when designing and developing mobile phone based distributed applications.

REFERENCES

Abramov, E. S., & Rogov, S. V. (2009). *New opportunities for Java ME developers with location API 2.0*. IEEE Eurocon 2009. Saint Petersburg, Russia, 438-443.3.

Al-Jaroodi, J., Mohamed, N., Jiang, H., & Swanson, D. (2003, April). *Modeling parallel applications performance on heterogeneous systems.* Paper presented at the Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France.

Bagchi, K., Kirs, P., & Lopez, F. (2008). The impact of price decreases on telephone and cell phone diffusion. *Information & Management, 45*, 183-193.

Chang, Y.-F., & Chen, C. S. (2005). Smart phone - the choice of client platform for mobile commerce. *Computer Standards & Interfaces, 27*, 329-336.

Hasegawa, M., Nakamura, K., Higashijima, A., Kawasaki, S., Nakashima, H., Sato, & K. N. (2008). High accessible experimental information on CPD experiment. *Fusion Engineering and Design, 83*, 402-405.

Helal, S. (2002a). Pervasive Java. *Pervasive computing, 1*(1), 82 - 85.

Helal, S. (2002b). Pervasive Java, Part II. *Pervasive computing, 1*(2), 85-89.

Heo, J., Hamb, D.-H., Park, S., Song, C., & Yoon, W. C. (2009). A framework for evaluating the usability of mobile phones based on multi-level, hierarchical model of usability factors. *Interacting with Computers, 21*, 263-275.

Hopfner, H., Schad, J., Wendland, S., & Mansour, E. (2009). *MyMIDP: An JDBC Driver for accessing MySQL from mobile devices*. Paper presented at the First International Conference on Advances in Databases, Knowledge, and Data Applications

JCP (2009a). *Participation - Executive Committe info*. Retrieved October 10, 2009, from http://jcp.org/en/participation/committee

JCP (2009b). *Participation - Overview: Getting involved.* Retrieved October 10, 2009, from http://jcp.org/en/participation/overview

Jode, M. d. (2004). *Programming Java 2 Micro Edition for Symbian OS: A developer's guide to MIDP 2.0*. Chichester, West Sussex: Wiley.

Klingsheim, A. N., Moen, V., & Hole, K. J. (2007). Challenges in securing networked J2ME applications. *Computer, 40*(2), 24-30.

Knyziak, T., & Winiecki, W. (2003, Sept.). *The new prospects of distributed measurement systems using Java™ 2 Micro Edition mobile phone.* Paper presented at the Proceedings of the Second IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Lviv, Ukraine.

Knyziaka, T., & Winiecki, W. (2005). The new prospects of distributed measurement systems using Java 2 Micro Edition mobile phone. *Computer Standards and Interfaces, 28*, 183-193.

Kozel, T., & Slaby, A. (2008, June). *Mobile access into information systems.* Paper presented at the 30th International Conference on Information Technology Interfaces, Cavtat/Dubrovnik, Croatia.

Marejka, R. (2005). *MIDlet life cycle*. Retrieved December 15, 2008, from http://developers.sun.com/mobility/learn/midp/lifecycle/

Mazlan, M. A. (2006). *Stress test on J2ME compatible mobile device*. Paper presented at The Innovations in Information Technology.

Mock, M., & Couturier, S. (2005). *Middleware - integration of small devices*. Paper presented at the Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation.

Sun Microsystems. (2009a). *Connected limited device configuration* (CLDC). Retrieved January 10, 2009, from http://java.sun.com/products/cldc/

Sun Microsystems. (2009b). *Mobile Information Device Profile* (MIDP). Retrieved January 10, 2009, from http://java.sun.com/products/midp/

Sun Microsystems. (2009c). *Sun Java wireless toolkit for CLDC*. Retrieved January 10, 2009, from http://java.sun.com/products/sjwtoolkit/

Takeuchi, A., NoritakaMamorita, FumihikoSakai, & Ikeda, N. (in press). Development of a Comprehensive Medical Recorder on a Cellphone. *Computer Methods and Programs in Biomedicine*.

Tanenbaum, A. S., & Steen, M. v. (2002). *Distributed systems principles and paradigms*. Singapore: Pearson Education.

Want, R. (2009). When cell phones become computers. *Pervasive computing, IEEE, 8*(2), 2-5.

Xu, C.-w. (2006). *A framework for developing wireless mobile online applications*. Paper presented at the 5th IEEE/ACIS International Conference on Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse. ICIS-COMSAR.

Yan, L., & Liang, Z. (2009). An accelerator design for speedup of Java execution in consumer mobile devices. *Computers and Electrical Engineering, 35*(6), 904-919.