Java's Alternatives and the Limitations of Java when Writing Cross-Platform Applications for Mobile Devices in the Medical Domain

Marcus D. Bloice, Franz Wotawa and Andreas Holzinger
Institute for Medical Informatics, Statistics & Documentation,
Research Unit HCI4MED, Medical University of Graz
& Institute of Software Technology, Graz University of Technology
marcus.bloice@meduni-graz.at, wotawa@ist.tugraz.at, andreas.holzinger@meduni-graz.at

Abstract. *In this paper we discuss alternatives to Java ME when writing medical applications for mobile devices across multiple platforms. The Java Virtual Machine, which runs Java programs, is not available for the majority of handheld devices, such as Palm PDAs, Windows Mobile based devices, or the Apple iPhone. As well as this, we conclude that full GUI interaction, such as the interaction provided by Java programs, is not an absolute requirement to make a program useful, and we developed an HTML-based medical information application to illustrate this. This program displays various sample patient parameters to the user in graph form, and was tested on multiple platforms and operating systems to demonstrate its platform/OS independence and usefulness.*

Keywords. Mobile Medical Informatics, IT in Healthcare, Java and Medical Informatics

1. Introduction

Healthcare, like any other institution, has seen a massive increase in employee computer use over the past decade. Medical information systems allow doctors and other medical staff to access patient data in a matter of seconds. Unfortunately, in many hospitals, these terminals are point-of-access computers fixed in certain locations around the hospital. Therefore, the use of such information systems is limited to these positions around the hospital, and can inhibit and slow the speed at which a doctor can access patient data. This is especially true when doctors are doing their rounds, as terminals are often not readily available in patient wards. A solution to this problem would be to use handheld devices to provide access to patient medical data while on the move. However, hardware issues often inhibit this option due to the cost of developing the software for the devices as well as the cost of installing the appropriate infrastructure. Unlike the personal computer world, there is no single operating system or platform which dominates the mobile device industry; there are any number of manufacturers, platforms, and operating systems available. Therefore, the software required cannot simply be purchased off the shelf, and must be developed "in house" for the device chosen by the health institution. This incurs huge costs, is not very future-proof, and makes the entire prospect unfeasible for many institutions. Hardware changes at a fast pace, and as newer devices become available older hardware becomes increasingly difficult to replace. Moreover, newer hardware may not necessarily support the existing in-house software.

What is required is a device independent platform that allows any device, with internet access, to be able to retrieve patient data while on the move. Almost all modern handheld devices, such as mobile phones or PDAs, offer internet connectivity of some description, be it Wi-Fi, GPRS or HSDPA. However, there are a multitude of manufacturers that produce these devices, and many have differing operating systems, development platforms, screen resolutions and capabilities. This makes it impossible to develop a medical application that could be universally run on any device. The application would need to be developed for every device and OS combination there is, which is absolutely not feasible.

For exactly these reasons the Java ME (micro edition) platform was developed by Sun. Applications written in Java can be run on any device that is Java compatible – in other words, any device that has a Java Virtual Machine (JVM). For the purposes of this paper we researched Java's feasibility for creating mobile and cross-platform medical applications. We also discuss any alternatives to Java, and if there are

other, perhaps better, alternatives. Our research focused on application development that would work on all devices, regardless of the underlying technology of the device itself.

We also elaborate as to whether mobile devices can be effectively used to access patient medical data or not. This work will provide a foundation for other medical institutions that may be interesting in implementing a mobile access platform for their staff.

A sample, web-based, medical application was developed for the purposes of this paper. This was done in order to test the feasibility of HTML-based applications for use in the mobile medical application domain. This application was tested on multiple platforms and devices, and some benchmarking was also performed.

For creating cross-platform mobile applications, Java ME would seem a good place to start. One reason for considering Java is its availability on mobile and handheld devices and according to Sun, Java is available on billions of devices [12]. Unfortunately, as we will further outline in this paper, writing Java applications for handheld devices is not as straightforward a task as Sun may like you to believe. Their "Write Once, Run Anywhere" slogan is not particularly apt in the context of handheld device development, as there are no official Java implementations for any of the major operating systems used in mobile devices. Even though Java has been successful in the mobile *phone* domain, it is often not possible to run an application built for a Nokia phone on, say, a Sony-Ericsson phone. As well as this, health institutions are unlikely to invest in thousands of identical devices at once, and newer devices run the risk of not being able to run the Java software built for the older devices.

Therefore, other approaches must be considered. This approach must be more future-proof, less platform dependent, and incur lower costs. We will therefore discuss as many alternatives to Java as possible, and also discuss the real feasibility of using these other platforms in the area of medical informatics.

The paper is organized as follows: In the next section we discuss use of Java in the medical informatics domain, and in general, as a viable solution for multi-platform mobile device application development. Section 3 provides an overview of the alternatives that were researched, including the previously mentioned browser-based approach. In section 4 we present the empirical results that were obtained, providing screenshots and benchmarks of the various platforms and devices. Last, we conclude the paper and discuss any further research.

2. Drawbacks of Java

Sun state that Java is the "most ubiquitous application platform for mobile devices", and that Java is available on billions of mobile devices [12]. These devices range from Blu-Ray devices to mobile phones (where Java is most prevalant). However, PDA-like devices often do not include Java support, as there is no official Java runtime for either Windows Mobile or Palm OS; PDA-like devices would be of most interest to developers writing programs for the mobile medical application domain, due to their size and ergonomics. In contrast, mobile phone devices are mostly being ignored as it is difficult to develop real-world applications for them; they often have no keyboards or touch screens, have varying screen resolutions and sizes [13], and cross platform support is limited. Devices which do have pre-defined and non-varying screen sizes and resolutions, such as Apple's iPhone, often do not have Java runtime environments. In the case of the iPhone, this is not likely to change in the near future; a clause in the iPhone End User License Agreement forbids the installation of any product that allows any executable code to be run. Should Sun develop an Java runtime, it would not actually be able to execute any Java code.

However, even on Java enabled devices, portability can be an issue. Java's portability relies on a device manfucaturer's 100% compliance to the Java Specification when building their implementations of the Java Virtual Machine. This guideline, if fully complied with, enables a manufacturer to build a Java Virtual Machine for their device that will run any Java built application for a particular version of Java [6]. However, in order to allow Java applications to access a device's hardware specific features, it is rare that manufacturers are able to follow the Java Specification exactly. Firms, such as Nokia, do this to allow Java programs to control the host device's low-level features, such as the phones's camera or Bluetooth chipset. This, of course, is no fault of Sun's, who cannot do much more except make the standard available to anyone wishing to build a JVM for their platform. However, disregarding the reasons why, the fact remains that a Java

application created for one Java “compatible" mobile device may not work on another Java compatible device. Therefore, even on Java enabled devices, cross-device compatibility is not a reality.

On top of this, designing applications that display correctly on differently sized screens, with different resolutions is difficult. Although this is not a Java specific problem, at least one technology we researched does not suffer from this problem (see section 3.4).

The Java ME specification also lacks several key aspects which make development more difficult when creating real-world applications. Object serialization, to take one short example, is a feature of the full Java language that would be benefical to mobile agents but is not a standard feature of the J2ME specification [14] although workarounds for this are available. Other problems, besides portability, can be found in [7] but will not be dicussed further in this paper.

2.1 Improving Java’s Portability

Sun is more than aware of Java’s portability issues, and has released the Light-Weight UI Toolkit in an attempt to better at least some aspects of Java mobile application development. This LWUIT is an Open Source (with classpath exception) framework that will allow developers to create User Interfaces that look identical on all devices running them, rather than writing platform/Virtual Machine dependent code. This may well improve Java’s adoption as a platform for mobile medical application development, as it would provide consistency among devices running the application in question. As well as this, there are Java “profiles" which further aid manufacturers in creating compatible runtimes for Java devices, such as the Mobile Information Device Profile (MIDP). Any device certified as MIDP 2.0, for example, should be able to run any application designed as an MIDP 2.0 application. Again, however, this relies on manufacturers following Sun’s guidelines exactly.

2.2 Security of Java ME

Patient data is especially sensitive and the transfer of such data should be made using a secure protocol at all times. Since MIDP 2.0, SSL v3.0 has been supported. Any Java midlets created prior to the MIDP 2.0 standard can not support SSL, making this data extremely vulnerable to interception. However, a flaw exists in the implementation of MIDP 2.0 where the seed used to generate the random key that enables the first handshake with the HTTPS server can be computed. This is because the seed used to create the random key is the time in milliseconds (using `System.currentTimeMillis`) which can be guessed by a determined hacker. The security aspects of Java ME are discussed in detail in [4], and are beyond the scope of this paper; however, it will form the basis for further research. HTTP security is discussed in section 3.4.

3. Alternative approaches for mobile access of medical data

Through our work in the domain of medical information systems we were able to determine a number of alternatives to using Java for creating medical applications. These alternatives are discussed in the following sections, starting with a relative newcomer, Google Android. As well as discussing any alternatives, we performed a number of experiments on multiple devices, running various operating systems and browsers. To do this, an HTML-based application was developed which displays sample patient information to the user in the form of a graph displayed as a static PNG file. Each time the HTML page is accessed, it uses data from a MySQL database to generate the PNG image on the fly from the patient data contained there. All logic is performed on the server side using PHP, meaning that the device using the application must only be able to render HTML and be capable of displaying PNG files. Each time the page is called, the graph is generated from the current data in the database, meaning it is up to date and valid at the time of execution. In order to illustrate the ability of many devices being able to use the application, screen-shots were taken for each device that was tested, and are displayed in their relevant subsections.

3.1 Google Android

Google’s Android is a new platform for mobile devices. It consists of an Operating System (OS), middleware, and some of Google’s own applications, such as Gmail. Applications can be developed for Android using the Android

SDK, which allows programs to be developed using Java. However, Android utilizes a custom Java runtime environment called Dalvik meaning that it is not a standard Java runtime. This means that applications built to run on Android, will not run on any other Java Virtual Machine. Android can be seen rendering the test application in Figure 1.

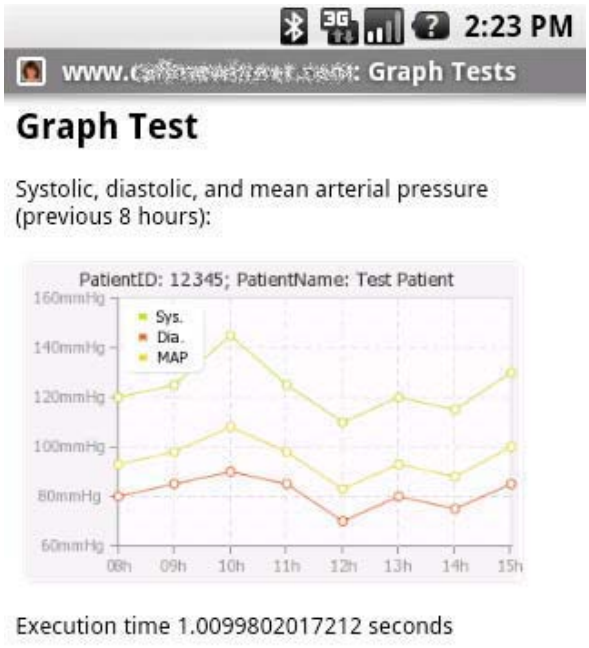

Figure 1. Google Android

However, despite the fact that applications built for Android will not run on other virtual machines, Android is interesting because the entire OS is open source, and could be made to run on any device - devices with or without keyboards, touch screens, or even devices without screens at all. In other words, Android makes no assumptions whatsoever as to the minimum requirements of the device it is being built to run on. This has the advantage that the OS could be made to run on custom hardware, and could therefore be considered to be very future proof and device independent.

3.2. Adobe Flash

With the release of the open source Flex Software Development Kit (SDK) Adobe have made available a language that can be compiled into cross-platform flash files. These files can be run on Windows, Mac OS X, and GNU/Linux/Unix, and an increasingly large number of traditional mobiles devices. The recent AIR platform will mean that Flash programs can be saved from the web and run from the desktop, just like any other application [1]. Spurred on by the success of YouTube, mobile phone makes have been quick to adopt Flash players into their products. An iPhone implementation is currently in the development phase, while unofficial Flash players are available for Palm devices. However, the Flash runtime is too large and complex for many handheld devices, and most manufacturers only supply a subset of Flash's functionality in their mobile implementations. Flash Lite (see section 3.3) may well improve portability and be more suitable as a framework for mobile medical application development [9].

3.3. Flash Lite

It is difficult to judge any impact that Flash Lite may have as it is a very recent technology. Its impact will depend on its acceptance by the various major mobile device manufacturers, such as Apple's iPhone [3]. Its usefulness will also depend on the uniformity of the runtimes written by the various device vendors. It has also been suggested that Adobe's Flash Lite may prove to be significantly faster than current technologies [15], although in this early phase it is difficult to confirm this.

3.4. HTML

HTML is the most platform independent language there is. It is a well established, well documented, and open specification [11], supported by hundreds of different devices, browsers and operating systems. The most compelling feature of pure HTML is that the browser does not need to support any device specific technologies or environments. It must simply be able the render the HTML; any logic required is performed on the server-side by any number of frameworks and environments. In most cases it is nearly impossible to tell which type of server a user is actually communicating

with when browsing on the web. Internet servers running on GNU/Linux, UNIX, and Windows return only text which is then interpreted by a browser and displayed as a web page to an end user. This means that any device that can render HTML can get information from any type of web server.

Of course, an HTML page is not a program, and cannot do much except display formatted text and tables to a user; there is little interaction and a HTML page can perform no calculations. Conversely, HTML's primary disadvantage is also its main advantage: all calculations and logic are performed entirely on the computer serving the file: this is especially useful for battery powered, mobile clients with weak CPUs. By placing the computational emphasis on the server, you are also effectively remove the programming language/platform-dependence barrier altogether - because all servers return HTML, it does not matter what technologies actually produced the HTML. Additionally, the majority of server-side technologies that are currently in use are multi-platform. PHP, Perl, MySQL, and Apache are available on all major platforms.

And, by storing the logic on a server, any bug fixes or changes in functionality are instantly made available to all devices using the application. This is not certainly not so in the case of installable programs, as devices will invariably not be properly updated or patched.

As mentioned in section 2, screen resolutions can often make programming mobile application particularly problematic. Traditional programs must be designed with at least a range of resolutions in mind, while HTML-based applications are far more dynamic in this regard, meaning the developer can focus more on creating the functionality of the program, rather than worrying about device resolutions or screen sizes. Each device displaying an HTML page makes its own decision on how to scale the page and display it to the user. Devices often give you the choice of pan and zoom, or full screen mode.

As long as a mobile device has a standards compliant HTML browser, it is inherently capable of communicating with multiple platforms and systems, which remain invisible to it. The presentation is simply made in HTML, and the logic could be performed using any number of different languages and technologies.

Disadvantages of HTML

HTML has several inherent limitations, however. User interaction is extremely limited as HTML applications are very linear - they do not offer much in the form of feedback, and do not dynamically update themselves like traditional applications. A HTML application would not be an ideal way of viewing constantly variable information in real time. Having said this, it could be used to view the current status of certain data; by creating graphs on the fly on a server and displaying these graphs to the user as a static image, this can be adequately achieved, for example (see section 4).

Another disadvantage of HTML, that is made much more apparent in the medical domain, is patient data security [5]. HTML that is transmitted over the unsecure Hypertext Transfer Protocol (HTTP) is not safe [8], and could potentially be read by third parties. In order to overcome this, the secure HTTPS (Hypertext Transfer Protocol Secure) protocol would have to be used to transmit all sensitive patient data [8]. This measure, however, would severely limit the number of devices that could access the application as many smaller devices and mobile browsers do not support the HTTPS protocol. Authentication is also a security issue. Not all medical personnel have access the same patient data. Therefore, access must be restricted at a device level, or by requiring the user to login at the beginning of the session.

4. Experimental Work

The test application was created using PHP, a multi-platform, dynamic programming language, with the aid of a freely available PHP library called pChart [10]. The pChart library can be used to create PNG image format graphs on the fly from data stored in text files or SQL databases. Therefore, if a user were to access the test application, they would be presented with a graphical representation of the data in the database at the time of access. Because all of the logic of the application is performed by the server, the device accessing the page must only need to be able to render some simple HTML and be able to display a PNG file (Portable Network Graphics image file).

The application's simplicity is highlighted by the code shown in Program 1 (The PHP code is not visible, as it is interpreted on the server side, and only HTML is returned to the device accessing the page). The graph itself displays the previous eight hours of systolic, diastolic and mean arterial pressure data for an example patient.

```
<html>
  <head>
    <title>Graph Tests</title>
  </head>

  <body>
    <h2>Graph Test</h2>
    <p>
      Systolic, diastolic, and mean
      arterial pressure (previous 8 hours):
    </p>
    <img src="example1.png">
  </body>
</html>
```

Program 1. Test Application Code

Because the logic is performed on the server side, the device accessing the application must only be able to interpret the code displayed in Program 1. Creating the graph is the task of the server, and it does not matter to the browser which technology is used to carry this out.

4.1. Results

In order to perform the experiments, we accessed the test application using mobile devices that varied in both their OS and design. Therefore, more traditional devices, such as the Nokia 5310, running the Opera Mini browser (see Figure 2), were tested, as well as new devices such as the iPhone (see Figure 3). Windows Mobile 6, accessing the device using its built in browser, was also tested as it is a popular OS used by PDAs. The Nokia device tested accessed the web server using a GPRS/EDGE data connection, while the other devices were tested using emulators on a computer with a DSL connection. All screenshots were taken from emulators.

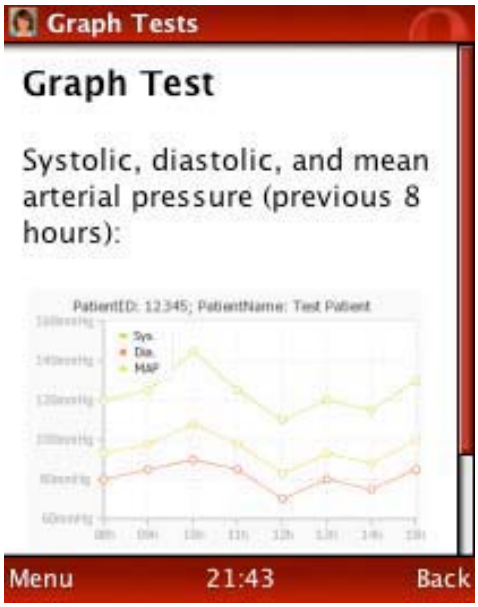

Figure 2. Opera Mini 4.1

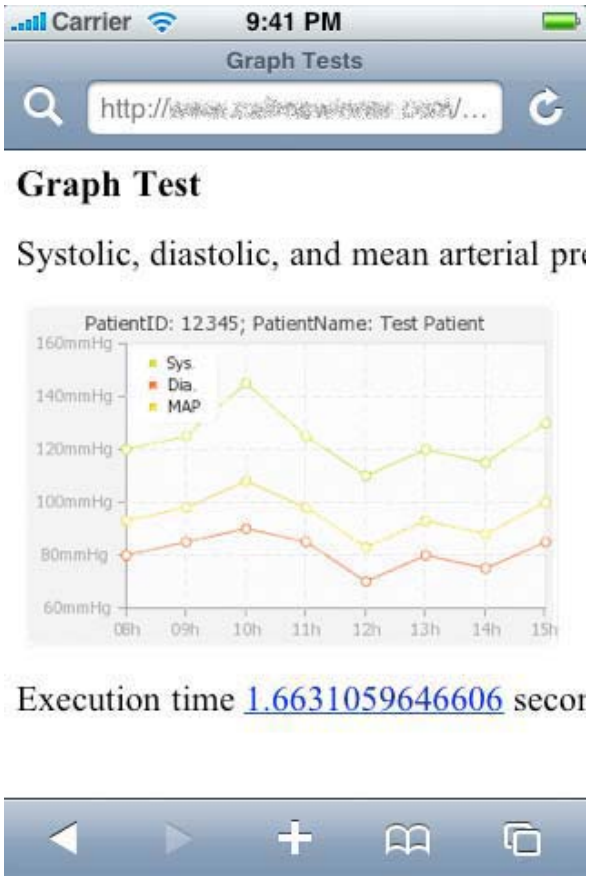

Figure 3. Apple iPhone

Stresses on the server side were also kept reasonably low; the test application's HTML file

itself is just 0.25kB (see Program 1) in size, and the images created by the test application were generally just over 20kB in size. This results in approximately 21kB of data that has to be transferred from the server to the client per request made. Again, this is main advantage of the client-server architecture; the vast majority of the processing that occurs takes places on the server, freeing the client of much of the work.

Execution time was also within reasonable thresholds. The graphs were almost always rendered in less than one second and averaged 0.7 seconds per rendering. Also worth noting was that every device that was tested was able to legibly render the graph (see Figures 1, 2, 3, and 4). None of the devices tested had any issues rendering the HTML file, despite the differing operating systems, browsers, screen sizes, and resolutions.

And, due to the server-client architecture, loading times did not differ between mobiles devices and desktop browsers, as shown in Table 1.

Table 1. Benchmarks (Best Times)

Device/Platform	Time in secs.
Opera Mini 4.1	0.5913
O_2 Xda II	0.6639
Firefox 3 / Win XP	0.6151
Apple iPhone	0.7418
Windows Mobile 6	0.6486
Safari / Mac OS X	0.7887

It is conceivable, however, that under certain circumstances, with many hundreds of concurrent connections, that close to one second loading time per request made would result in too much stress on the server, but with a little effort the efficiency of the program could be dramatically improved. It would be relatively easy to build in a check to see if the data source has changed before creating a new image for a new request. This alone would reduce the average time for each request significantly. In order to illustrate the difference this would make, one test run was performed using a cached image, rather than a created image, which resulted in a loading time of 5.10×10^{-5} seconds (this time of would only be achievable if a graph was created before the dataset changed and the cached version was sent the browser).

5. Conclusions

The use of mobile devices in the field of medicine is well documented [2], [16], [17] yet its potential has yet to be properly reached. This is a combination of factors, and we believe that a development environment which is less restrictive, more feasible, and properly device independent would spur development in this area. By developing the sample application discussed in this paper, we show that at least some patient information, which is not constantly variable, can be displayed in an efficient and usable manner by using nothing more than an HTML-based application.

In the near future we will perform usability analysis tests to ascertain whether HTML offers the level of interaction required to make it a viable choice for developing mobile medical applications. A comprehensive requirements analysis will also be performed, in order to make clear what the real requirements are for mobile applications in the medical domain. The will be performed from the medical professional's point of view, and from a technological standpoint. For example, at what point does a static HTML page not offer the level of interaction needed to make an application useful?

We will also propose a system that could be realistically developed that would display patient information to doctors working at the University Hospital Graz. This system would include an administration front end that could be accessed using a desktop browser, which would allow the medical professional to decide what data they would like to view and for which patients. This information for the chosen patients would then be accessible using the mobile device, and would avoid the need for the doctor to use the mobile device to choose the data and patients they want.

Last, HTTPS and SSL will be researched for mobile devices and the Java ME.

8. References

[1] Apollo for Adobe Flex Developers Pocket Guide. O'Reilly, 2007. ISBN 9780596513917.

[2] Elizabeth S. Chen, Eneida A. Mendonça, Lawrence K. McKnight, Peter D. Stetson, Jianbo Lei, and James J. Cimino. PalmCIS: A Wireless Handheld Application for Satisfying Clinician Information Needs. Journal of the American Medical Informatics Association, 11(1):19–28, 2004.

[3] T. Chen. The Web is Everywhere [Note from the Editor-in-Chief]. Communications Magazine, IEEE, 45(9): 16–16, 2007.

[4] M. Debbabi, M. Saleh, C. Talhi, and S. Zhioua. Security analysis of mobile Java. Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on, pages 231–235, Aug. 2005. ISSN 1529-4188. 10.1109/DEXA.2005.172.

[5] J.W. Goldwein and I. Benjamin. Internet-Based Medical Information: Time to Take Charge. Annals of Internal Medicine, 123(2): 152–153, 1995.

[6] R. Hayes. 100% Pure Java Cookbook. Sun Microsystems, Inc.

[7] D.S. Kochnev and A.A. Terekhov. Surviving Java for mobiles. Pervasive Computing, IEEE, 2(2):90–95, April-June 2003. ISSN 1536-1268. 10.1109/MPRV.2003.1203758.

[8] D.M. Kristol. HTTP Cookies: Standards, Privacy, and Politics. ACM Transactions on Internet Technology, 1 (2):151–198, 2001.

[9] Kevin Patrick, William G. Griswold, Fred Raab, and Stephen S. Intille. Health and the Mobile Phone. American Journal of Preventive Medicine, 35 (2):177–181, 2008. ISSN 0749-3797. DOI: 10.1016/j.amepre.2008.05.001.

[10] pChart. pChart — a PHP Charting library. URL http://pchart.sourceforge.net/ [10/29/2008]

[11] D. Raggett, A. Le Hors, and I. Jacobs. HTML 4.01 Specification. W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C), Dec, 1999.

[12] Sun Microsystems. The Java ME Platform. URL http://java.sun.com/javame/index.jsp. Online; accessed Nov. 25 2008.

[13] Alf Inge Wang, M.S. Norum, and C.W. Lund. Issues related to Development of Wireless Peer-to-Peer Games in J2ME. Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on, page 115, Feb.

[14] D. Wong, N. Paciorek, and D. Moore. Java-based Mobile Agents. Communications of the ACM, 42(3):92–102, 1999.

[15] Imran A. Zualkernan, Yaser A. Ghanam, Mohammed F. Shoshaa, and Amir S. Kalbasi. An Architecture for Dynamic Generation of QTI 2.1 Assessments for Mobile Devices Using Flash Lite. Advanced Learning Technologies, IEEE International Conference on, pages 194–195, 2007. http://doi.ieeecomputersociety.org/10.1109/ICALT.2007.55.

[16] A. Holzinger, A, M. Hoeller, M. Bloice, B. Urlesberger. Typical Problems with developing mobile applications for health care: Some lessons learned from developing user-centered mobile applications in a hospital environment. In: Joaquim Filipe, David A. Marca, Boris Shishkov & Marten van Sinderen (2008), Proceedings of the International Conference on E-Business (ICE-B 2008), Porto (PT), 235-240, 2008

[17] A. Holzinger, M. Errath, M. Mobile Computer Web-Application Design in Medicine: Research Based Guidelines. Springer Universal Access in Information Society International Journal. 6, 1, 31-41, 2007