Wireless Health and the Smart Phone Conundrum

Jonathan Woodbridge*, Ani Nahapetian*, Hyduke Noshadi*‡,
Majid Sarrafzadeh*, William Kaiser†

*Computer Science Department, University of California, Los Angeles
{jwoodbri,ani,hyduke,majid}@cs.ucla.edu

†Electrical Engineering Department, University of California, Los Angeles
{kaiser}@ee.ucla.edu

‡Google, Inc. 1600 Amphitheater Parkway, Mountain View, CA

ABSTRACT

This paper presents a study of the five best selling Smart Phones in terms of their applicability to Wireless Health. Smart Phones are generally used as central controlling units in Wireless Health applications. We carried out our investigation by implementing a wireless health application that performs sensor communication, data processing, and data visualization. Our overarching goal is to develop a plug-and-play Wireless Health software platform. Our task begins with an in depth study of Smart Phones: the central controller of Wireless health applications.

KEYWORDS

Wireless Health, Smart Phones, Bluetooth

1. INTRODUCTION

Over the past few years, researchers developed several wireless health platforms such as [3][14][16][17][22][23]. These platforms often include a mobile device (such as a PDA or cell phone) as the central controlling, processing, and visualization unit. Figure 1 depicts one such architecture. Previous work is predominantly focused on overall architectures and lacks focus on the central processing unit. Our goal is to analyze several commercial Smart Phones to determine the best targets for wireless health.

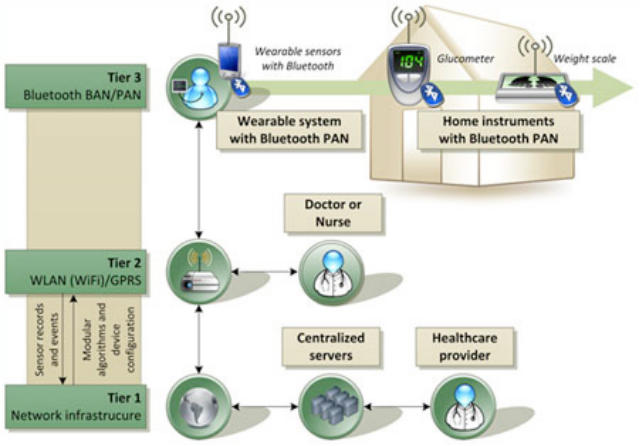

Figure 1. A standard architecture for Wireless Health Applications. The wearable system acts as a central processing unit for the patient.

Our comparison is based on a set of libraries developed for wireless health. We feel that extending current wireless health platforms by offering an additional software component (set of libraries) for mobile devices would add tremendous value. Reliability, code reuse, and decreased development times are just a few of the many benefits offered by such software.

Implementing a set of libraries requires intimate knowledge of the target devices. This paper presents an examination of popular Smart Phone platforms, based off of the design of a simplified application that uses a few basic components of a wireless health library. We design our libraries based on a few active research projects here in our labs at UCLA. We describe these projects in section 2. Next, we discuss the feasibility of implementing such an application (including libraries) on several commercial platforms. Finally, we present the development of our simplified application and libraries to prove the feasibility of such a software platform.

This paper does not present a complete wireless health software library. Our library is only representative of a complete implementation. We use this representative as a basis of comparison between Smart Phone platforms and prove the feasibility of a complete wireless health software library.

All platforms compared in this paper support cellular connectivity (such as CDMA and UMTS). Such devices offer far greater network coverage than devices that only support technologies such as Bluetooth and WiFi. In a large number of wireless health applications, this constant connectivity is required, especially in the case of applications that require high mobility.

During our comparison of platforms, we analyze feature availability as well as emulation and debugging environments. We hope this comparison serves as a guide for those wishing to develop wireless health applications for deployment on smart phone platforms.

The key contributes of this paper are three fold. First, we provide an assessment of the five best selling Smart Phones platforms and their applicability towards wireless health. Second, we determine the best software runtime environment in applicable to our five Smart Phone platforms. Finally, we developed a wireless health application to prove the correctness of our assessments.

2. WIRELESS HEALTH APPLICATIONS

The following section presents a few wireless health projects under development at UCLA. We use these projects as inspiration for a set of wireless health libraries. These projects by no means represent all wireless health applications. However, they do establish a baseline for comparison.

2.1 SmartCane

Falls are the leading cause of death in the elderly. To mitigate this phenomenon, The Wireless Health Institute at UCLA has developed the SmartCane System [27]. This system performs a series of signal processing algorithms to assess the users current state. These algorithms assess various attributes such as improper cane usage, high-risk behaviors, and potential injuries (such as falling). Once these attributes are detected, the SmartCane can propagate these attributes to patients, care givers, clinicians, as well as emergency services. To accomplish signal processing and network

connectivity, the SmartCane connects to a PDA, Cell Phone, or tablet PC via Bluetooth. This central controlling unit can predict hazards, store behavioral data, notify health care professionals, as well as display visual feedback to the user [27].

3. Smart Shoe

Smart Shoe is an orthotic shoe developed in our labs at UCLA [8][21]. Through the use of gyroscopes, accelerometers, and a few well-placed pressure sensors, Smart Shoe is able to monitor feet motion and pressure distribution to evaluate the state of a patient. The Smart Shoe can currently detect the formation of foot ulcers in patients with diabetes. Similar work has been done in other labs. For instance, [19] have developed a shoe-integrated sensor system for gait analysis. Like the SmartCane, Smart Shoe wirelessly connects (via Bluetooth) to a cell phone or PDA for data processing, visualization, and network connectivity.

4. Developing Wireless Health Applications

Projects presented in section 2 share many of the same requirements. The following lists a series of features required by the aforementioned projects:

- Short Range Connectivity (such as Bluetooth or Zigbee)
- Internet Connectivity (through Wifi, CDMA, etc…)
- Visualization (such as OpenGL ES)
- Data Storage (such as SQL)
- GPS Services

Usability is an important addition to our technical requirements. End users range from health care professionals to patients (who are often elderly). Therefore, we can expect a large percentage of non-technical users who are not computer savvy. A successful wireless health application must provide a usable experience that integrates seamlessly into a patient's life. Otherwise, we risk low adoption rates.

4.1 Sample Application

To provide a basis of comparison, we created a wireless health application that interfaces with UCLA's Smart Shoe. This application connects to wireless sensors via Bluetooth and displays their output graphically. This application requires four main modules:

- Connectivity module for Bluetooth (Zigbee was not supported by any of our Smart Phones)
- Graphical module for displaying data
- Data storage module for archiving sensor data
- GPS module for associating locations with data

Our comparison is limited to Smart Phones to account for Internet connectivity. Other mobile platforms such as PDAs typically connect to the Internet through WiFi. Wireless Health Applications require a constant level of connectivity regardless of locality. This requirement cannot be satisfied with WiFi alone. Therefore, we require a more ubiquitous network such as CDMA or GSM. With such networks, patients are constantly connected regardless of their locations. However, even cell coverage has its limitations in remote locations and may experience dead spots in urban areas. However, we feel that there is no technology that offers a higher level of connectivity. Also, dead spots could often be mitigated through WiFi (supported by many Smart Phone platforms).

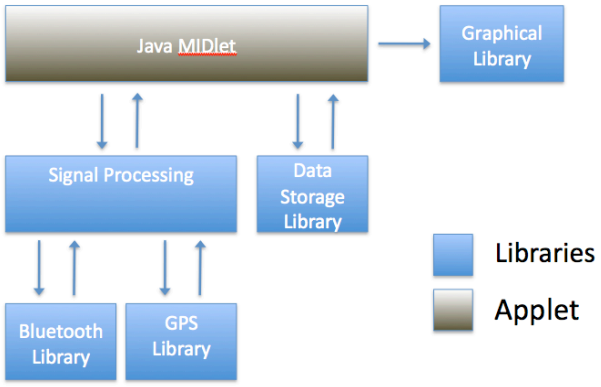

Figure 2. Our Simplified Library Architecture

5. Mobile Devices

The list of potential wireless devices is practically endless. In order to provide a useful survey, we limited our set of devices to five. Our devices include Symbian, Rim Blackberry, Windows Mobile, Android, and iPhone. Symbian, RIM Blackberry, and Windows Mobile have the three highest world market shares at 57.1%, 17.4%, and 12% respectively [1]. iPhone holds the fifth largest market share at 2.8% and offers a unique user interface paradigm unlike the four preceding platforms (Linux being number 4 at 7.3%). We chose Android to represent the Linux platform. There are several Linux platforms in the mobile market. Android was chosen due to its openness, support, and standardization offered by Google; thereby lending itself as an attractive research platform. Android also offers a similar user experience to the iPhone and serves as a suitable comparison. Android was also chosen due to Google's past record and market penetration by their other products.

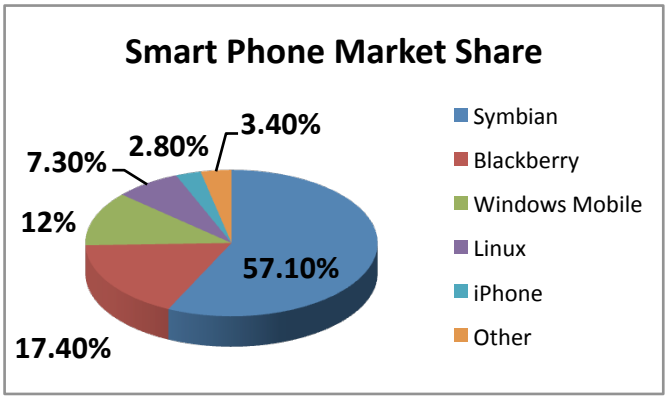

Figure 3. Smartphone market share

5.1 BlackBerry/Symbian/Windows Mobile

Blackberry, Symbian and Windows Mobile support a standard J2ME port. Applications can be compiled to a jar file and loaded to all three devices without the need for recompilation. However, while developing for such platforms, we must verify support for required JSRs for each device. Many JSRs are optional such as JSR 82 for Bluetooth support. However, we found that many of the latest devices (such as Nokia's N95) advertise support for all JSRs listed above.

5.2 Android

Android is built upon an open Linux Kernel and consists of a virtual machine optimized for mobile environments. Android uses the Java

programming language. However, their JAVA port is for the Dalvik JVM. This port consists of a mix of standard JAVA and Android specific APIs. Therefore, JAVA applets compiled against standard ports, such as J2ME, are not compatible with Android.

Android is open source lending itself nicely to research and industry. Developers can run their custom Android builds on unlocked hardware available through Google. Android also makes no distinction between applications; all applications, whether static core applications or dynamic third-party applications, are treated identically and have equal access to the device's functionality [2].

5.3 iPhone

iPhone OS offers a similar platform to MAC OS X. IPhone OS runs a variant of the same Mach kernel used by MAC OS X. This enables iPhone to support standard core services such as BSD Sockets and POSIX Threads. Also, objective C is a superset of C and C++. For these reasons, open source projects can be trivially ported to the iPhone. While working with this device we ported a simple JSON interpreter [12]. The port consisted of importing the necessary headers and source files and compiling. We found this functionality an attractive feature of iPhone. Unfortunately, it was the only platform in our selection to offer standard core OS services.

6. Platform Comparison

To host our sample application, our platforms must provide several APIs including Security, Bluetooth, GPS, storage APIs (such as SQL), standard networking APIs, and graphical APIs. Platforms must also provide a user-friendly interaction model. While not required, a large touch screen is highly attractive for such applications. Large touch screens allow us to display large text, graphics and controllers (such as buttons and lists). Much of our work is often targeted towards the elderly where vision and finger acuity is diminished. Large touch screen displays afford an experience much better suited to such users.

Both iPhone and Android had no programmatic support for any low power wireless protocols (such as Bluetooth and Zigbee). Wireless sensors are typically connected via Bluetooth or Zigbee. Support for such APIs is a strong requirement for wireless health. Without such APIs, wireless platforms are severely limited. Unfortunately, Zigbee was supported by none of our Smart Phone platforms. This is quite a drawback since Zigbee provides an extremely energy efficient wireless alternative to Bluetooth [6]. However, it is important to note that Google has announced Bluetooth support in future SDKs.

Our initial goal was to choose a single mobile device to serve our research interests. However, as we researched several devices, we found that many of these devices are extremely similar and creating a library that extends several platforms was possible.

For these reasons, we targeted Symbian, BlackBerry, and Windows Mobile. Each of these mobile platforms supports J2ME. This allows us to create libraries that we can share across all 3 platforms without the need for recompilation. Also, J2ME offers JSRs that support security (JSR 219), Open GL ES (JSR 177/239), GPS (JSR 179) and Bluetooth (JSR-82) [15]. By choosing J2ME as our target, we include a much larger set of mobile devices than those we present in this paper. These include mobile devices mot considered Smart Phones (such as low end cell phones and other embedded devices).

Figure 2 lists our APIs of interest and their respective support by our five Smart Phones. (Symbian, Windows Mobile, and BlackBerry were merged to J2ME).

Table 1: List of supported APIs

	J2ME	iPhone	Android
Bluetooth	√	-	-
WiFi	-	√	√
ZigBee	-	-	-
GPS	√	√	√
Open GL ES	√	√	√
Security Suite	√	√	√
SQL	√	√	√
Touch Screen	√	√	√

6.1 Ease of Deployment

Ease of deployment refers to a developers‘ ability to deploy applications to a handset. For our purposes, applications are often cable loaded. We found loading applications to J2ME and Android trivial. Neither platform required any licensing. Tools were free and easy to access. iPhone, on the other hand, requires developers to join their Developer Program. This process required an application as well as a nominal fee. For us, the entire process took several weeks. While we understand the business justifications for such a process, we feel that it quite a deterrent to the academic community.

6.2 Emulation and Debugging

We found Androids debug environment impressive. First, Android emulates several features such as GPS coordinates, network speeds (such as UMTS and GSM), SMS, and voice calls. Android also contains a debugger for stepping through code while monitoring various attributes such as thread states, heap usage (with manual garbage collection), and file system state. All of this is accomplished through the Eclipse IDE with no external tools with a minimal startup time. We were able to load an application and start using all these features in about 45 minutes with a little help from Androids documentation.

However, we found two areas of improvement for Android. First, we would like to see a graphical CPU monitoring that works on the device as well as the emulator. CPU monitoring only worked on the emulator. Also, the output was a small line drawn on the upper part of the screen. A graphical display that allows us to save info and correlate CPU usage to its respective code segments would be quite useful. Second, GPS simulations only worked for the emulator (as for all devices presented by this paper). We would like to see GPS simulation supported by the device as well.

For J2ME, on device debugging is quite device specific. Several manufacturers do release their own device specific debugging tools that integrate with common IDEs such as EclipseME [9] and Netbeans [20]. However, we found this fragmented experience to be quite a limitation of J2ME. While the other two platforms provided a uniform and robust debugging framework, J2ME's on device debugging support relied solely on the device manufacturer (granted, iPhone is developed for only Apple hardware).

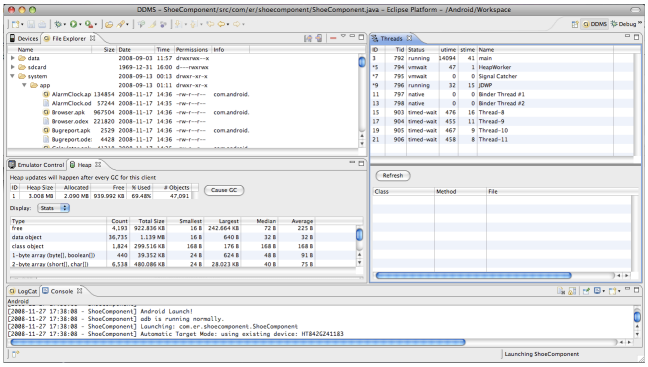

Figure 4. Android's Debugging Environment

We used the Java Debugger (jdb) from sun while debugging on the Simulator. We ran this tool from the command line as well as NetBeans and found the debugger to be quite rudimentary. We were able to set break point, check variables, and step through code as with most debuggers. However, we felt that jdb lacked the advanced features offered by Android and iPhone. The emulator supported standard simulated features, such as GPS and telephony features

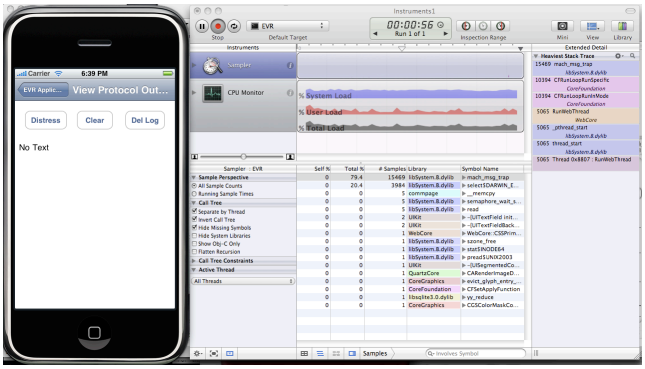

Figure 5. Apple's Instruments Tool

iPhone had the most impressive statistical tools out of the mobile devices we compared. Apple offers their Instruments tool that allows developers to graphically monitor several features such as memory usage, CPU usage, frame rate, etc. Instruments also allows developers to save traces to later analyze or send to fellow developers. Instruments not only worked for the simulator, but for the device as well. For intensive applications that require a high level of optimizations, we rank iPhone as the clear winner for support tools.

However, we were unable to emulate GPS coordinates from the simulator. While we were able to create an instance of their Location API, the API consistently returned the same coordinates at Cupertino, Ca.

7. ANALYSIS

While comparing mobile platforms, we quickly realized that development of Wireless Health libraries could extend across multiple mobile platforms using Java. With Java's compile once run anywhere environment, we gain a level of standardization that not only improves our code portability, but also provides a set of standard APIs shared across mobile devices. Of the five platforms compared, three supported a standard Java implementation (specifically J2ME). We found these three platforms tended to provide necessary APIs we need for Wireless Health such as security, Open GL ES, GPS and Bluetooth. We also found that functional requirements such as threads and background applications were supported. Through providing a standard environment, Java has not only provided a portable platform, but an environment inclusive of our requirements for Wireless Health.

Interesting to note, we also ran our initial libraries on standard Windows XP and Mac OSx laptops. Porting to such platforms involved a simple integration with the BlueCove library provided by [5] and some changes to our visual APIs. This exercise provided us with a much higher level of optimism for our Wireless Health libraries. As Java moves across many new embedded platforms, our libraries can be leveraged not only in mobile devices (such as mobile phones and PDAs) and laptops, but also by any embedded device supporting a standard implementation of Java.

With these observations, it was abundantly clear that J2ME offers the best runtime environment for wireless health applications. This does leave out both Android and iPhone in the interim. Since iPhone runs a similar kernel (Mach) as standard OSX, it is possible for apple to include a standard Java virtual machine (J2ME or J2SE). As for Android, the Dalvik JVM is not a far reach from Sun's J2ME standard. In fact, much of our code on both J2ME implementations and Android could be shared. We feel that our Wireless Health libraries could extend to Android with increased API support from Android as well as some ingenuity from us.

Android and iPhone currently have limitations that severely hinder wireless health applications. The most noticeable limitation was the lack of a low power networking APIs for technologies such as Zigbee or Bluetooth. While these devices do support WiFi, we feel this technology is too power hungry for wireless health applications. However, both of these devices do offer hardware support for Bluetooth 2.0 and could very well provide API support in the future. iPhone also lacks the ability to run background applications (without using discouraged means such as jail breaking). Wireless Health applications are often required to constantly monitor their environment. However, when using a mobile device such as a mobile phone, we must remember that these devices are intended for multiple purposes. While our medical monitoring is extremely important we cannot completely control the device rendering other functionalities inaccessible. As stated earlier, one of our requirements is the ability to seamlessly (as possible) add our applications into patients' lives.

7.1 User Experience

As noted earlier, user experience is critical to wireless heath applications. End users may or may not be technically savvy. Therefore, applications should be intuitive. In addition, many wireless health applications are intended for the elderly, such as SmartCane and Smart Shoe (described in section 2). In general, eyesight and finger dexterity decrease with age. Enhanced features such as larger attributes (such as fonts, images, and inputs) and better color contrasts are necessary to address the needs of the elderly [10][11]. Large touch screens cater to these attributes quite well. Fortunately, all five platforms presented in this paper support such displays. Hardware support was the largest limiting factor when considering user experience. We found that both Android and iPhone excelled in the area of user experience. While Blackberry, Windows Mobile, and Symbian support similar displays, there are few commercial products that utilize such a display.

8. SECURITY

This paper has purposely left out an in depth comparison of security

related features. We feel that security is extremely important and includes such a broad area, that this topic deserves its own dedicated analysis. For the purpose of this paper, we only compared whether each application supports a suite of security APIs. It is important to note that security in wireless health as well as mobile/embedded devices is an area of ongoing research. Authors in [4][13] have noted several areas of security concerns in protecting health information (wireless health info for [13]) such as authentication, confidentiality, secure links for data exchange, data integrity, and access protection for stored data. Authors in [26] have noted several potential solutions for these issues. Authors in [7][18][24] have discussed how resource constrained embedded platforms offer a new set of requirements for security measures beyond their "wired" counterparts. We feel that future work should include a survey of all these aspects and how current mobile platforms and their respective security suites help alleviate issues in security for wireless health.

9. IMPLEMENTATION

Our Analysis was completed with an implementation of a simplified library on the J2ME platform. J2ME lends its self quite nicely to developing wireless health libraries. As noted earlier, Bluetooth, GPS, SQL, OpenGL, and a security suite are all supported by J2ME.

9.1 Implementation Details

We used a similar configuration as [26] in our implementation. Our shoe consists of one MicroLeap [3] processor for both the left and right shoe. This processor connects two pressure sensors (one in the heel and toe) as well as an accelerometer and gyroscope in all three, X, Y, and Z, axes. Figure 9 shows our application running on the Nokia N95.

Figure 9. J2ME Application running on the Nokia N95

Our JAVA library consists of MicroLeap, data storage, graphic, and data processing abstractions. Each of these abstractions hides the intimate details of their respective functionalities. We also retrieve and store GPS data through the data storage APIs.

Once activated, the application retrieves sensor input from the pressure sensors, accelerometer and gyroscope. We sampled data at about 50Hz (although, much higher sampling rates are possible). This data is processed by the data processing API and stored by the data storage API. We performed basic analysis of the shoe's sensory data at runtime to determine the person's current balance. We used a basic algorithm that compared data from the left and right shoe to determine symmetry. This was accomplished by comparing the standard deviation of the X, Y, and Z accelerometer data for each foot. This rudimentary algorithm was able to determine if a person was walking abnormally (such as limping, stumbling, or shuffling).

Data stored on the device was later transferred to a PC where we could do more complex data processing. We implemented a playback mechanism for our PC using the same Java libraries. This mechanism allowed us to visually replay data retrieved by our Smart Phones

9.2 Implementation Analysis

We found some discrepancies while deploying our libraries on Windows Mobile, BlackBerry, and Nokia devices. For Windows Mobile, we used an HP iPaq. While the device has Bluetooth and GPS support, JSR 82 and JSR 179 were not a part of their JVM. For Blackberry, we used the Blackberry Pearl. This device fortunately does support JSR 82 and JSR 179. However, some minor features were not supported. For example, when using Bluetooth serial port profile (btspp) we could not set the server as the master node. This issue was also present on the Nokia N95. Fortunately, we were able to work around this dilemma with our implementation. However, this could be an issue for other implementations.

Several mobile devices and laptops require a passkey when pairing with a Bluetooth device. Typically embedded devices account for this with a hard coded passkey in their firmware. However, we found lacking support in some of our prototype hardware. While this is an issue of the embedded hardware, developers should be aware of such technicalities.

Overall, we still feel that J2ME is the best target for developing wireless health libraries. However, we found that some devices had no support for GPS and/or Bluetooth (even when hardware support was present).

We also feel that J2ME's debugging utilities are lacking. While the current jdb is sufficient for debugging issues (such as the Bluetooth discrepancies we described earlier), the experience is quite fragmented. iPhone and Android, on the other hand, offer a completely seamless debug experience with a series of tools for optimization. We hope to see similar support in J2ME in the future.

iPhone and Android are severely limited by their lack of low power connectivity APIs (such as Bluetooth and Zigbee). However, their platforms may suite intensive data processing quite well due to their optimization tools (especially iPhone). However, iPhone also lacks the ability to run background processes; a requirement necessary for wireless heath applications similar to those described in section 2.

10. CONCLUSIONS

Our initial intent was to find the best mobile platform for wireless health applications. We based our comparison on the development of a simplified wireless health library. The requirements for this library were based on several ongoing research projects in our labs at UCLA.

We developed these libraries (where possible) on five mobile platforms (Windows Mobile, Blackberry, Symbian, iPhone, and Android). Through this process, we became quite familiar with all five environments. During this process, we noticed several advantages afforded by J2ME such as a unified runtime

environment across a large number of devices. These advantages, along with the lack of necessary functionality by both iPhone and Andoird, led us to a J2ME implementation.

This exercise proved that J2ME was indeed a prime candidate for developing wireless health applications. However, we found support for GPS and Bluetooth varied across devices. Therefore, the number of devices that can actually host such applications is smaller than preferred. We hope to see a more unified support for Bluetooth and GPS on J2ME devices. We also hope for a better debugging environment, similar to that of Android and iPhone.

11. REFERENCES

[1] 2008 Press Releases, Gartner, Inc., 2008. http://www.gartner.com/it/page.jsp?id=754112

[2] Android, Open Handset Alliance, 2008. http://www.openhandsetalliance.com/android_overview.html

[3] AU, L. K., WU, W. H., BATALIN, M. A., MCINTIRE D. H., KAISER, W. J. 2007. MicroLEAP: Energy-aware Wireless Sensor Platform for Biomedical Sensing Applications. Biomedical Circuits and Systems Conference, BIOCAS 2007. Montreal, Canada, November 2007, 158–162

[4] BlackBerry Developer Zone, Research In Motion Limited. 2008. http://na.blackberry.com/eng/developers/

[5] BlueCove, BlueCove.org. 2008. http://www.bluecove.org

[6] CALLAWAY, E., GORDAY, P., HESTER, L., GUTIERREZ, J. A., NAEVE, M., HEILE, B., BAHL, V. 2002. Home Networking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks. IEEE Communications Magazine, 40(8), August 2002, 70–77.

[7] CLAYTON P. D. 1997. For the Record: Protecting Electronic Health Information. National Research Council, National Academy Press. Washington DC. 1997.

[8] DABIRI, F., VAHDATPOUR, A., NOSHADI, H., HAGOPIAN, H., SARRAFZADEH M. 2008. Ubiquitous Personal Assistive System for Neuropathy. The 2nd International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments (HealthNet), in conjunction with ACM MobiSys, Breckenridge, Colorado, July 2008.

[9] EclipseME, EclipseME.org. 2008. http://eclipseme.org/

[10] HANSON, V. L. 200. Web access for elderly citizens, in Proceedings of the Workshop on Universal on Accessibility of Ubiquitous Computing, WUAUC'01, Alccer do Sal, Portugal, May 2000, 17 – 24.

[11] HANSON, V. L. 2004. The user experience: designs and adaptations. In W4A: Proceedings of the international cross-disciplinary workshop on Web accessibility, New York, NY, USA. May 2004. 1-11.

[12] Introducing JSON, JSON. 2008. www.json.org

[13] iPhone Developer Center, Apple. 2008. http://developer.apple.com/iphone/

[14] JAFARI, R., BAJCSY, S., GLASER, B. GNADE, M. SGROI, and SASTRY, S. 2007. Platform design for health-care monitoring applications. In Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, Boston, MA, USA, June 2007, pp. 88–94.

[15] Java ME at a Glance, Sun MicroSystems. 2008. http://java.sun.com/javame/index.jsp

[16] JOVANOV E., PRICE J., RASKOVIC D., KAVI K., MARTIN T., and ADHAMI R. 2008. Wireless Personal Area Networks in Telemedical Environment. In Proc. 3rd Int. Conf. Inf. Technol. Biomed. Arlington, VA, Nov. 2000, 22-27.

[17] LO, B., THIEMJARUS, S., KING R., and YANG, G.Z. 2005. Body Sensor Network - A Wireless Sensor Platform for Pervasive Healthcare Monitoring. In Adjunct Proceedings of the 3rd International Conference on Pervasive Computing, May 2005, Munich, Germany, 77-80.

[18] MILLER, S. K. 2001. Facing the Challenges of Wireless Security. In IEEE Transactions on Computers, July 2001, 46–48.

[19] MORRIS, S.J. and PARADISO, J.A. 2002. A compact wearable sensor package for clinical gait monitoring. Motorola Journal, 2002. 7-15

[20] NetBeans, NetBeans.org. 2008. http://www.netbeans.org/

[21] NOSHADI, H., AHMADIAN, S., DABIRI, F., NAHAPETIAN A., STATHOPOULUS, T., BATALIN, M., KAISER, W., SARRAFZADEH, M. 2008. Smart Shoe for Balance, Fall Risk Assessment and Applications in Wireless Health. Microsoft eScience Workshop Indianapolis, IN, December 2008.

[22] OTTOAND, C., MILENKOVIC, A., et al. 2006. System Architecture of a Wireless Body Area Sensor Network for Ubiquitous Health Monitoring. Journal of Mobile Multimedia. 1(4), 307–326.

[23] PENTLAND A. 2004. Healthware: Medical Technology Becomes Wearable. IEEE Computer, 37(5), May 2004, 42-49.

[24] POTLAPALLY, N., RAVI, S., RAGHUNATHAN, A., and LAKSHMINARAYANA G. 2002. Algorithm exploration for efficient public-key security processing on wireless handsets. In Design, Automation and Test in Europe (DATE), Nice, France, Mar. 2002, 42-46.

[25] SHNAYDER, V., CHEN, B., LORINEZ, K., FULFORD-JONES, T. R. F., WELSCH, M. 2005. Sensor Networks for Medical Care. In Harvard University Technical Report TR-08-05, 2005.

[26] WARREN, S., LEBAK, J., YAO, J., CREEKMORE, J., MILENKOVIC, A., and JOVANOV, E. 2005. Interoperability and Security in Wireless Body Area Network Infrastructures. In Proceedings of the 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Shanghai, China, 2005. 3837-3840.

[27] WU, W., AU, L., JORDAN, B., STATHOPOULOS T., BATALIN M., KAISER, W., VAHDATPOUR, A., SARRAFZADEH, M., FANG M., CHODOSH, J. 2008. The Smart-Cane System: An Assistive Device for Geriatrics. Third International Conference on Body Area Networks (BodyNets 2008), Florence, Italy, March