Exploiting Mobile Computing in Health-care

Usman Arshad[*$], Cecilia Mascolo[*] and Marcus Mellor[$]
[*]Dept. of Computer Science, University College London, Gower St, WC1E 6BT London UK
[$]Capula Elan, Palmerston Court, Palmerston Way, London, SW8 4AJ London UK

***Abstract*—In this paper we report about our experience in using mobile computing middleware in context of health-care. The dynamicity and variability of context and conditions make this environment very suitable for the use of mobile and wearable computing techniques. The use of small and portable devices can be very beneficial in terms of efficiency and vital support to patients. However, the many challenges that this environment presents need to be addressed, possibly by a general mobile computing framework that could be used in different mobile settings. We will discuss these issues in the paper along with the description of the prototype we have developed.**

1 Introduction

Small and portable devices, such as mobile phones, PDAs and the like, have recently been pushed on the marked and have allowed for complex cooperation and communication patterns that were not foreseeable some time ago. The ability to carry these devices, share the content using communication networks, possibly wireless, and synchronize their content with more standard devices have been seen as essential features. However, these technologies still come with some limitations and differences with respect to the standard computing platforms we have used for years. Resources (such as battery, bandwidth, memory) are by orders of magnitude smaller, and patterns of use change depending on the network availability and on the location of users.

Applications have begun to be developed for these devices in order to allow data synchronization, Internet browsing, and cooperation. The most targeted domains have been Mobile Commerce and E-shopping, however it is becoming clear that the use of mobile technologies will become quite pervasive in our lives and that we need to support development of applications in different areas.

In particular, we have recently been involved in the use of mobile computing in health care setting. The healthcare environment is quite unique in that it brings with it several constraints and requirements that any deployable application would have to adhere to. The most obvious issues are those of patient confidentiality and doctor- patient relationship. It is vital that medical data is kept confidential via the use of potentially several vertical layers of security.

Healthcare data is mission critical and errors introduced by an application can have potentially fatal consequences. For example, in the case of a medical prescribing tool, a miscalculation or misprint in the dosage of any particular drug to be administered could have serious consequences for the patient. Hence, it is important that some kind of data integrity assurance is supported by the application.

The clinical health-care working environment is 'highly mobile', with clinicians constantly moving around from patient to patient while performing their duties. This constrains the shape and performance of computing devices that can be used in such an environment. Many organizations have attempted to implement computing solutions to aid clinicians with their every day duties using a networked desktop environment. In practice, such solutions have failed to reach their target users, and instead simply served as administration systems. The reason behind this is the fact that in such a highly mobile environment, clinicians make decisions at the 'point of care' i.e., the patients bedside, and they do not have the time to leave and potentially queue up to refer to a desktop based application. Hence, the ideal platform for use in such an environment is the PDA or tablet PC platform, which the clinician can carry around and use wherever necessary.

In this paper we will introduce the platform we have developed for use in health-care scenarios and show the general principles that have driven our design decisions. We will show how the platform is planned to be used, and how it can improve the efficiency of patient care.

The paper is structured as follows: Section 2 describes the mobile computing environment and the challenges it presents. Section 3 introduces our case study, looks at the motivation and describes our adopted architecture. Section 4 describes the implementation in details. Section 5 discusses the interesting outcomes of our work and shares some of the experience gained from working in a mobile computing environment. Finally, Section 6 concludes the paper and outlines our planned future work.

2 Mobile Computing

Point of care delivery is vital for the success of any application in the clinical healthcare environment. Hence, the use of a PDA platform utilizing wearable technology is ideal and is adopted by the work described in this paper. Wearable and mobile technologies introduce considerations and constraints that have to be dealt with when developing software. This section highlights some of these and explains how they impact upon development.

2.1 Network Model

Mobile networks can be roughly classified into two types, ad-hoc or nomadic. An ad-hoc network is one that is formed by two or more mobile enabled devices that are in reach of each other and can form a compatible network. This model is suited to a peer-to-peer scenario there is no guarantee of any device remaining available or indeed being available at all. It provides no guarantee of resources or services, and has to deal with changing conditions such as disconnections, and resource availability.

A nomadic network is one that includes a fixed network infrastructure of base stations. Mobile devices are able to rely on the resources made available on the fixed infrastructure as long as they remain within reach of network coverage. This model is suited to a client-server approach where client would be the mobile device and the server would be available via the fixed infrastructure. This model still has the problem of disconnections to the network when mobile devices go out or range. The work described in this paper is based on a nomadic model using a client server approach, although the application we are developing has extensions of use in emergency areas where base stations are not in reach: we will speak more about this extension in the future work section at the end of the paper.

2.2 Mobile communications

Mobile communications are weak and prone to disconnections. This means that traditional applications that rely on network access will not work unless they use an always-on technology, and even then, they could encounter pockets of no coverage. In our view it is important for applications to take temporary disconnections into account, and adapt accordingly. This involves the use of reflection to detect network availability and the ability of an offline mode of operation when disconnected. This marks a move towards thick client devices with embedded intelligence.

The situation is complicated further when potentially many concurrent users have access to the same piece of data. In such a situation if some people are working offline on locally cached copies the data while others are working online, inconsistencies can quickly develop. Hence, there is a clear need to be able to identify such inconsistencies and deal with them accordingly. With weak communications, data transfer rates have to be taken into account. Application developers need to consider the time it takes to download data to a device or to upload from it and whether this time delay makes the application unusable. Of course, data rates depend on the type of mobile technology that is to be used, but is even more significant in cases where a user is charged for connection to network on a time basis.

2.3 Resources

Mobile devices such as PDA's are extremely resource-constrained in terms of memory, processing power, battery lifetime and screen size. Applications for such devices need to be resource conserving and lightweight enough to achieve a level performance deemed usable. The application developer also needs to take into account the strain put on these resources during runtime, and there are often tradeoffs to be made as to where to execute processes and store information, whether it be locally on the mobile device or remotely on a more powerful device. More details on these issues can be found in [6] and [7].

The way we address these issues is discussed in Section 4. The fact that PDAs have a very small screen and a different means of navigation than traditional systems means that careful consideration needs to go into the development of a usable user interface.

In the next section we will describe our approach to the use of mobile computing in health-care.

3 Our approach

3.1 Scenario

The motivation behind the work described in this paper is to aid the clinician by empowering him/her with mission critical information at the point of care. This is achieved by developing a PDA based application that uses mobile technology in a Client-Server Nomadic setting. The application being developed combines an Electronic Patient Record (EPR) system with an e-prescribing application. The purpose of an EPR system, is to replace traditional paper based methods of documenting patient records with electronic records that are available to multiple users concurrently. The records are dynamic in that they change on a regular basis, and global consistency is important. One example of the dynamic nature of patient records is current medication that is prescribed to a patient. The e-prescribing application provides the clinician with a database of drug information including the complexities of drug interactions and contra-indications. The clinician is able to prescribe drugs to his patients and the application flags up potential problems caused by drug interactions.

3.2 Architecture

Figure 1 shows the architecture we have adopted: a client-server architecture separated by a mobile network. The server connects to a central data-store, which contains data shared by the client PDA devices. The communication between the client and the server is via a packet based communication protocol. A session is initiated by a clinician logging in to the system via the client *User Display*. This initiates the *Downloader* component to contact the server and retrieve patient record data from the database. Once downloaded the clinician is able to navigate the data via the application. If data are modified by the clinician, the *Updater* component ensures that the changes are reported to the server and that the data-store is updated accordingly. It also makes sure that the change is communicated to all other clients logged onto the system.

One of the key aspects of our approach is ability of the client to work offline, i.e., if the client PDA is out of reach of the network any changes made by the clinician are cached locally until such a time at which the network is once again available. To enable this the client needs to be network aware. This is achieved by the server *Presence Broadcaster* regularly

broadcasting one-way messages on the network, which are picked up by the *Presence Receiver* on the client. As long as the client receives these messages, it assumes it is still connected to the network. This information is fed into the updater, which makes the decision as to whether to cache the changes or send them to the server.

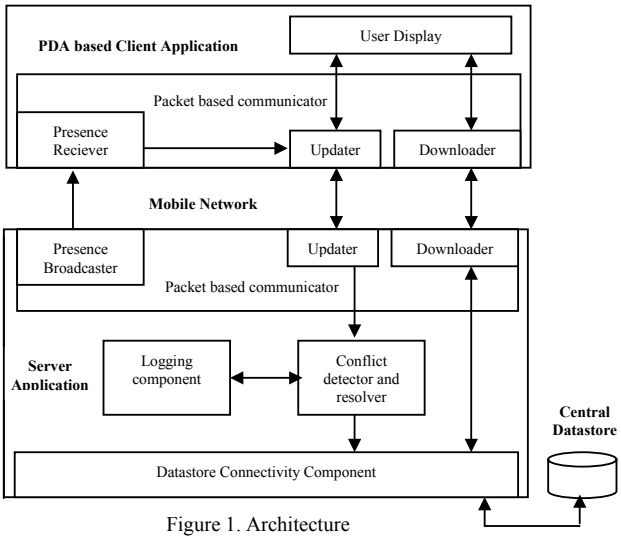

Figure 1. Architecture

While offline, a client may miss updates made by other clients. A process to receive these missed updates once the client is again online is put in place. Further to this, while offline, the clinician may make updates that conflict with updates made by other clients. The *Conflict Dectector and Resolver* component and the *Logging* component take care of this. Further details are provided in Section 4.5.

4 Implementation

We now describe the details of the platform we have developed. The server is a Java application that provides access to an Oracle database. The client device is also a Java application. The server retrieves data via an API that returns the data in the form of an XML document. Similarly changes to the data are committed to the database by passing the API an XML document representing the changed data. The reasons for choosing XML are discussed in Section 4.4. The Server exists as part of a fixed network infrastructure that is extended to the mobile clients using an 802.11b wireless LAN.

4.1 Presence Information

In order to be network aware, the platform needs a mechanism to gain presence information. The clients use two approaches to gain this information

1) Active approach – The clients are responsible for checking if the network is accessible to them. The client would send something similar to a ping command to the server expecting a response to indicate accessibility. This approach is efficient, as minimal messages have to be sent over the network. However the client has to do more work.
2) Passive approach – In this approach the Server is the active component and continually broadcast a message over the network attempting to reach clients. The clients simply listen for this message continually and as long as it is received on a regular basis, they assume the network is available. This approach has the disadvantage of putting extra load on the network

We use the passive approach in our scenario at the expense of the increased network load. This method is preferred as it requires minimal work on the part of the client, and given the scarce resources available on the client it is preferable to push work onto the server end. Further to this, it is expected that the client is likely to require a lot of access to the network, hence increasing the number of times the ping-like message would have to be sent.

4.2 Working offline

Applications for mobile devices can fall into three main categories, when dealing with disconnections.

1) Always-on Applications – These are dependent on network availability and require constant interaction with other networked machines, for example web based applications. These applications would fail completely if the network was unavailable, and hence are only deployed with always on communications such as GSM, or are limited to the afforded boundary of coverage.
2) Hot Sync Applications – These are commonplace on PDA platforms. In hot sync, it is not critical that the network is constantly available, and the user can continue to use the application when disconnected. When the user reconnects to the network, a synchronisation process takes place to filter across relevant information generated as a result of being disconnected to the network. The synchronisation process often occurs as a result of the user taking affirmative action to execute the process. A typical sync application could be a synchronization of an email inbox between a laptop and a PDA.
3) Auto update applications – These applications fall in between the other two approaches. Being connected to the network is important in such applications, however the user is able to work offline when the network is not available. Once back in reach of the network an automatic synchronisation process can take place. These applications appear similar to hot sync applications, but there are distinct differences. As mentioned, Sync applications often perform synchronisation as a result of an affirmative action on the part of the user, whereas in auto update applications, synchronisation remains transparent to the user. Further to this auto update applications often involve data that is shared by multiple users, hence synchronisation on a regular basis is important so modifications made to the shared data is filtered throughout the system, whereas hot sync applications usually involve synchronisation between a small number of devices (usually two) often belonging to a single user.

Our application follows the auto update model, which exploits transparency as far as possible, including the implementation of an automatic conflict resolution component (see Section 4.7). Transparency makes the application more usable, as the target users will not want to worry about whether they are working online or offline.

At the time of writing, many applications particularly those that depend on network availability, have tended to treat PDAs as *thin* clients, i.e., where the PDA is nothing more than a basic IO device. However our approach employs the PDA as a *thick* client with the embedded intelligence to store information locally when it is disconnected from the network.

As explained in Section 4.1 the server continually multicasts a presence message to the clients. In normal (online) operation any modifications made to the data by clients are immediately communicated back to the server, which makes the appropriate modifications in the central data store, logs the modification, and then multicasts the modifications to filter them throughout the system. The offline mode of operation is triggered when a presence message is not received in a given period of time.

Once working offline, the middleware still accepts modifications made to the data, but caches them locally. As soon as the presence messages are again being received, the client knows that it can once again enter into online mode, however it is vitally important that the following three process are carried out first:

- The client needs regain missed updates from the server to maintain data consistency in the system.
- The client needs to commit the changes it has made locally to the server
- A conflict check needs to be carried out to ensure that the modifications made by the client while it was offline do not conflict with modifications made by other clients.

These processes are achieved by the conflict resolution and logging components (see Section 4.5).

4.3 Development on PDAs

One of the first things that became clear while prototyping on the PDA platform was the limitation of memory, which severely hinders the amount of data that can be downloaded to a device. PDA applications have to work around this limitation. In order to cope with it we decided to adopt a 'session based download' approach. For example, in the case of the EPR application we decided that a doctor would log into the system, and only the records of the patients assigned to that doctor would be downloaded to the device, the session being the use of the application by that doctor. We used this approach as opposed to downloading all the patients on a ward for example.

It is also important to consider memory usage at runtime, as the size of data to be stored is not the only consideration. Some runtime processes can also be memory intensive, for example, depending on which parser is used, parsing XML can consume a lot of memory. Further to this, XML parsing itself can be a processor intensive process. Since our application is centered on delivering data in XML, our choice of parser would have to give us maximum performance.

Another important consideration for PDA like devices is the extent of multithreading that is used. In our experience, the use of too many concurrent threads on such a device can soon make an application unusable. In fact, at time of writing, Palm have only just added support for multitasking and multithreading with the release of version 5 of their PDA operating system PalmOS. Microsoft's PocketPC operating system has supported multithreading and multitasking since release, however as the number of threads increases, memory consumption is also increased and the application becomes increasingly unusable.

Another constraining factor on the PDA platform is the small screen size. This has considerable effect on the kind of applications that can be deployed on such a device. The biggest change is using a stylus as opposed to a keyboard or a mouse. This change impacts the UI as many traditional UI widgets or components are not usable via such an input mechanism, hence the developer is restricted to simple components such as buttons and dropdown lists. One of the terms that is emerging to describe the desired features of a PDA based user interface is 'high poke ability'. This includes using only simple widgets, but also encompasses usability models such as ensuring that the user does not have to frequently tap the screen to use the application to the extent that it requires too much effort to get even a small amount of work done. There is a shift from traditional desktop applications, which strive to provide a complete and comprehensive application that offers maximum functionality and, to a system that is centred on the critical workflow of the user and includes only the core and most important functionality. In fact such restrictive and simple UIs make PDAs intuitively easier to use and easier to learn than desktop systems for non computer literate personnel.

Battery lifetime is another concern on PDA platforms, and indeed battery technology is one area, which has not seen any significant advancement and is not expected to in the near future. Hence application developers will have to accept that the workflow of their applications will have to conform to use for short stints of time followed by a period of recharging the device. Application sessions cannot be long lived such that they exceed the battery lifetime. This problem is significantly magnified with the introduction of wireless communication, which can have significant drain on battery power.

Our application is being developed using the Java programming language. Java is emerging as an important language in the mobile device arena, and much of this is due to its platform independent nature. With the market still growing, a whole host of mobile devices have emerged including mobile phones, PDAs, tablet PCs and laptop computers each being ideal for use in some scenarios but not others. Further to this many more such devices are likely to emerge which are specifically designed for a certain environment. Some companies such as Intermec already offer specialised PDAs, which could for example be ruggedized for use in an industrial setting. Another interesting trend at time of writing is the way mobile phones are striving to become more like PDAs, offering more complex applications, whereas PDAs, are trying to become more like phones trying to offer voice call capability. Such heterogeneity between different hardware and operating systems means that platform independence is highly desirable. With the realisation of this, Sun Microsystems have already put considerable effort into developing small lightweight java virtual machines and standards such as Personal Java and J2ME to aid this. In our application, the use of several of these devices as opposed to just one may well be desirable. Hence we have made a point

of decoupling the UI from the core functionality in order to make the application portable between these different devices by accommodating an appropriate interface given the form factor of the device.

4.4 Use of XML

The last section discussed how platform independence is desirable. XML gives us a mechanism to generically represent data, which is instantly transferable between heterogeneous platforms. Our application takes this one step further with the use of XSLT to cater for different platforms. For example, consider the difference between a PDA and a tablet PC. Clearly the tablet PC is much more feature rich and has a much larger screen hence can present a lot more information visually. One way to achieve this would be to pass all the information available from the server to the mobile device whether it be a PDA or tablet. On receiving the information the different user interfaces on the two devices could decide to display all the information, or in the case of the PDA, just a small portion of it. However this clearly violates the concept of trying to conserve memory and keep processing to a minimum. Hence our application uses XSLT at the point of data retrieval to filter out the information that is not required for the device to which the data is to be sent. This is facilitated by simply specifying a different XSL style sheet to the transformer.

XML also separates our application into a middleware layer and an application layer. This is achieved as the middleware only deals with XML data. The data is retrieved by the middleware at the server end from the enterprise database system via an API that returns the data as XML document. This document is then sent to the client device, and the middleware hands off to the application layer by passing it the XML document. Following this, any subsequent changes made by the client are communicated to the middleware as XML. Again the middleware sends the XML to the server, at which point it hands off to the application layer by passing it the XML, and so on. The point is that the middleware handles all data as a CLOB of XML only, and does not care about the semantics of the XML.

4.5 Conflicts and conflict resolution

Allowing users to work offline in a system in which data is concurrently shared between multiple users, introduces several complexities into an application. For example, changes have to be cached locally until the user is back online at which point they need to be filtered throughout the system. Also while the user is offline he/she may well miss data updates made by other users who were online. As a result the offline client then has an inconsistent view of the data, and may well make updates that conflict with updates that have already been made.

Our middleware deals with the complexities of working offline by using a server administered logging process and a user configurable conflict resolution process. The server maintains a global version number, which is incremented with each update to the data. The update, along with the version number, is logged by the server. Each of the clients maintains a local version number, which corresponds to the last update they received from the server. Comparison of the version numbers between the server and client reveals whether any updates have been missed, and hence resent. The sequence diagram in Figure 2 and description below depicts how this works.

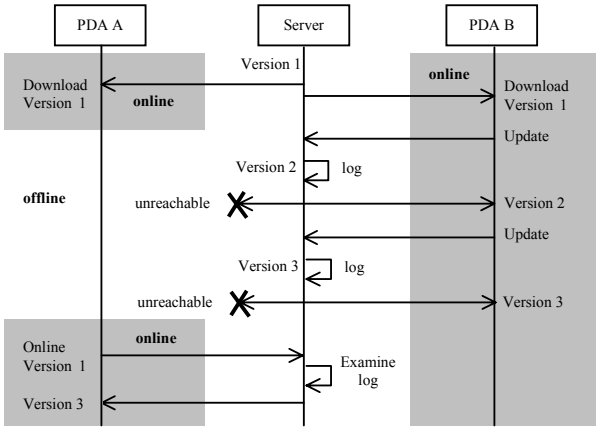

Figure 2. Missed updates

As the diagram shows, both client A and B are online to begin with, and individually download data from the server. At the point of download the server sends the global version number, which is recorded by the client. Client A then goes offline, and Client B then makes two successive updates. The server updates its global version number accordingly and multicasts the update along with the new version number to all the clients. Since client A was offline, it does not receive the updates, and hence its version number remains at 1. Client A then goes online once more, and on doing so sends a message to the server which includes its version number. The server is able to compare this to its global number and thus knows that client A missed two updates. The server is able to consult its log and resend the updates to client A.

The conflict resolution process is carried out on a per update basis, so each time an update is sent back to the server, it is compared against entries in the log for potential conflicts. If a conflict does occur the resolution process can then be started. The definition of a conflict depends on application logic, and hence the middleware needs to provide a way for the application to specify rules to resolve conflicts. Our middleware achieves this through the use of XML. When the server receives an update, it takes the XML for that update along with the XML for previous updates from the log and passes them to the conflict resolution module. This module then compares the two pieces of XML for conflicts according to rules defined by the application. If a conflict occurs the middleware first tries to automatically resolve them, otherwise the user can be notified accordingly.

5 Discussion

Performance is an important consideration for applications on resource-constrained devices. Given the fact that XML is text based, and the application will deal with transferring reasonable amounts of the data over the mobile network, means that there may well be a considerable delay while the transfer is being downloaded. Hence, it was decided to

employ compression before sending the data over the network.

The disadvantage to this is that we are now imposing on the processing power of the mobile device during the decompression process. However, the advantage gained was far more significant. Using simple zip format, a compression advantage in transmission time of the order of ten was seen. This varies according to the content of the XML, however given the data is similar to that of the English language other compression techniques that exploit the entropy of this type of data could show even more improvement.

The other potential overhead with using XML is the parsing of the data. Many of the fully featured parsers in existence would simply put too much drain on the memory and processing power. However there are several 'small footprint' parsers that are available that offer basic parsing functionality. We opted for NanoXML [3], but other similar parsers exist such as TinyXML[4] and MinML[5].

One of the notable consequences of working in the mobile environment is the different tradeoffs that have to be made. For example consider the location where data should be stored. It could be stored locally on the client device giving the advantage of fast access. However, clearly this is going to impact on the limited memory available on mobile devices. Alternatively the data could be store remotely at the server and downloaded as required, however this would incur a time overhead as the data is downloaded to the device. Thus there is a trade-off here between memory consumption and communication overhead, and the decision depends on the semantics of the data in relation to the application. For example data that requires frequent accesses would be better stored on the client then the server.

Another trade-off comes as a result of the weak processing power on the mobile device. It would often be advantageous to migrate processes to a more powerful machine such as the server in our application. This would save time in executing the process, although it would introduce a communication overhead. It would also ease the processing load on the client device. So again we have a trade-off between the communication overhead and the load on the device.

The healthcare community is showing a strong trend in the adoption of mobile technology, particularly in the use of mobile device such as PDAs [1,2]. Hence there are a vast amount of applications being developed for such devices. Most of the established applications are stand alone with the mobile devices such as drug databases and patient monitoring tools. However there is currently a lot of work being done into looking at systems that utilize mobile technology especially in the area of EPR. These include the systems described by [8], [9] and [10]. However current systems depend on the availability of the wireless network and do not address the issue of disconnections. Other work includes location aware ad-hoc applications such as connecting a PDA to a remote monitor at a patient's bedside to explain diagnosis.

6 Conclusions and Future Work

In this paper we have identified the complexities introduced when working in a mobile environment. In particular we have looked at resource constrained mobile devices such as PDAs, and shared our experience with developing on this platform. Our case study has shown one example of how small portable devices can be employed to bring significant benefits over traditional workstation based systems. Introducing mobile communications to the devices opens the door to a whole range of novel application.

We found that in a PDA based development environment the application programmer has to be more aware of the implications of mobile computing and often adapt the usage model of the application to accommodate these. Our case study has presented a generic architecture that for data sharing client server applications that could be employed in many scenarios. In particular we have emphasized the ability of our application to work in an offline mode of operation, and having a mechanism to detect and resolve conflicting changes to data using application specific rules.

The future direction of our work is looking to extend the principals learned, into an ad-hoc peer to peer setting. This is a more dynamic environment in which the mobile devices cannot rely on resource availability at all. Further to this in an ad-hoc setting, all devices may be potentially resource constrained. The natural extension of our case study into an ad-hoc environment is in the use of ambulatory care. For example data may be recorded in an ambulance on the way to the hospital and on reaching the destination transferred to a clinicians PDA or automatically integrated into the hospital system.

References

[1] M. Ancona, E. Coscia, G. Dodero, M. Earney, V. Gianuzzi, F. Minuto, and S. Virtuoso, "Ward-In-Hand: Wireless access to clinical records for mobile healthcare professionals." *TEHRE 2001 m-Health Conference, First Annual Conference on Mobile & Wireless Healthcare Applications,* November 2001

[2] G. Dodero, V. Gianuzzi, E. Coscia, S. Virtuoso, Wireless networking with a PDA: the Ward-In-Hand project, Proc. *Workshop on "CORBA and XML: towards a bioinformatics integrated network environment", Genova*, 17-18 May 2001. 115-118.

[3] NanoXML http://web.wanadoo.be/cyberelf/nanoxml/

[4] TinyXML http://www.gibaradunn.srac.org/tiny/

[5] MinML http://www.wilson.co.uk/xml/minml.htm

[6] C. Mascolo, L. Capra, W. Emmerich. Mobile Computing Middleware. In *Tutorial Notes of Int. Conf. Networking 2002*. LNCS 2497. Springer.

[7] C. Mascolo, L. Capra, S. Zachariadis, W. Emmerich, XMIDDLE: A data-sharing middleware for Mobile Computing. Wireless Personal Communications 21:77-103. Kluwer 2002.

[8] O. Portale, Healthcare: the mobile opportunity, 08 November 2002 *http://www.sun.com/mobility/enterprise/feature_story.html*

[9] D. K. Vawdrey, E. S. Hall, C. D. Knutson, A self-adapting, transport-aware mobile patient healthcare insfrastructure. *http://www.poketdoktor.com/PoketDoktorArchitecture.pdf*

[10] Sybase, Mobile computing in healtcare, *http://was.sybase.com/mec/2959healthcare.pdf*