WORK LISTS FOR THE TRANSPORT OF PATIENTS
A Case for Mobile Applications in Health Care

Andreas Holzinger, Jürgen Trauner
Institute of Medical Informatics, Statistics and Documentation (IMI), Research Unit HCI4MED
Graz University Hospital, Auenbruggerplatz 2/V, A-8036 Graz, Austria

Stefan Biffl
Complex Systems Engineering Lab, Institute of Software Technology and Interactive Systems (IFS)
Vienna Univeristy of Technology, Favoritenstrasse 11, A-1040 Wien, Austria

Keywords: Mobile User Interfaces, Medical Workflow optimization, Mobile Computing in Hospitals.

Abstract: In many hospitals, the workflow involved in transporting patients is supported by interactive work lists on desktop computers that provide the status of a work order and detailed information on request. However, the nurses, who organize and conduct the transportation, need to go to the desktop in order to update the work list status; which has been found very time consuming. In this paper we report on the development of a mobile solution prototype that provides interactive transport work lists on a PDA and we discuss some design and implementation issues. Despite the limited screen size, end-users were able to use the PDA with no more problems than their usual desktop user interface as the PDA user interface can be customized with property files to fit exactly the requirements of the end-users in the hospital.

1 INTRODUCTION

The transport of patients within hospitals is often supported by work lists on desktop computers (Gale & Gale, 2000). The central information is the status of a work order. Whilst such a non-mobile system is useful for work orders that do not change, the highly dynamic nature of hospital work often results to changing situations (Holzinger & Errath, 2007). Consequently, the nurses, who organize such transportations, are required to go to the desktop computer in order to update the work list status – only *to find out that their next patient is waiting where they just came from.* In this context, a desktop solution is time consuming, and consequently wastes personal resources. In our project example, at the Danube Hospital Vienna, nurses have the core responsibility of transporting patients between the Radiology department and other medical departments: 14 different rooms, each with a separate list of appointments.

At the beginning of their shift, the nurse prints out the list of patients to be transported on that day. Approximately 12 nurses are responsible for transportation per shift, which means that some must work with two lists. The list contains the name and the date of birth of the patient, for identification; the station where the patient lies; which type and when the checkup is planned; and whether the patient has to lie in bed or can walk. Usually, the nurse does not transport the patients in the order given on the list. Each transport is arranged with the assistant at the medical unit. Some changes in order are made by the nurse, for example, when more than one patient is at the same station or waiting at the station where another patient was returned. Then they may be fetched in order to save time. However, nurses are not automatically notified when patients are added during their shift, so either the nurses are informed about the new additions by the assistant or, less often, by telephone.

2 MATERIAL AND METHODS

In order to ensure platform independency of our mobile prototype, our first choice was Java. Fortunately, there is a free Virtual Machine for every Windows Mobile 2003 device (and later versions), called Mysaifu JVM (Freebeans, 2008).

Holzinger A., Trauner J. and Biffl S. (2008).
WORK LISTS FOR THE TRANSPORT OF PATIENTS - A Case for Mobile Applications in Health Care.
In *Proceedings of the International Conference on e-Business*, pages 454-459
DOI: 10.5220/0001909004540459

Although this is *not* a Java 2 Micro Edition, it is a Java 2 Standard Edition (J2SE) conformant Virtual Machine, which is built as a Source Forge project under the GNU Public License Version 2 (GPLv2). For this project we used version 0.2. This version supports both the Abstract Windowing Toolkit (AWT) and Swing. We applied small sample programs in order to ensure full functionality; the use of a standard edition JVM has, in our opinion, some benefits: first, it is very simple to test the application before copying the files to the PDA. Second, J2ME is *not just* J2ME; which means that there are many different versions *highly dependant on the performance of the device* (Sun, 2008), (Knyziak & Winiecki, 2005). Because of these issues and of other restrictions of J2ME, the J2SE was the best way to implement this application, in order to provide compatibility with most of the current systems – which are usually developed in Java (Bruno, 2005). The general idea was to use an already available connection to the existing server. The main connection type is the Bright Side Framework (BSF) (Brightsidefactory, 2008), which transports XML content over the HTTP protocol with dynamic instantiation and invocation of available interfaces. However, in such specialized circumstances the JVM for the PDA is *not* fully compatible with the version for the PCs. The other connection was based on Remote Method Invocation (RMI), facing the problem that dynamic interfaces were not supported on the PDA. This required us to develop another connection for the PDA. Due to the communication protocol should not be changed in the first version we built a relay application.

3 BACKEND LESSONS LEARNED

A new concept was necessary, employing a relay, in order to establish the connection to the server. On the server side the connection was built using RMI or BSF (Sims, 2004), while, on the PDA side the communication is based on an object stream. The BSF and the RMI connections were given the means to access the data on the server. Both communication connections had to be adapted to fit the requirements of this application. The communication is no longer made over a static connection; this is due to the fact that the relay has to be able to open many connections. On top of this, some additional functions were necessary. The most interesting change is the function which returns the sessionID. The communication process between the relay and server, and relay and PDA client, is provided by SASTransferObjects. These made it necessary to add a function to transport them without alteration. Example: the logon data is stored within an object but the function takes the username and the password as its arguments, therefore it is necessary to prevent direct access to this data.

On the other hand, there is the client. The newly built communication method is also sub-classed from the SASServerProxy. This makes it possible to change the method of communication just by changing the properties file. The first idea was to use an object stream for both directions; however there was some difficulty with this method. Generally, the communication with objects from the client to the relay does not work – because it is far more complicated as it normally is on PCs, due to the different JVMs. As already mentioned, the Mysaifu JVM is not yet fully compatible with the Java defaults, since it calculates the version numbers and does not allow for these to be fixed, as implemented in the source code. Every class which has a fixed version number causes an error at the relay when it is de-serialized. The InvalidClassException, states that the serial versions are different but in reality they are the same and only one number has been wrongly calculated. The solution for this is to change the version number before de-serialization. Any data which is sent through a stream is sent as a series of bytes. In an object stream, this is done with a special protocol which is publicly available (Sun, 2004). There is an exact description of how the class and the serialVersionUID are printed out. Some general information is followed by the class and super class information, which is succeeded by the class name printed in UTF. Directly after the name, the version number is written to the next eight bytes, followed by some flags and the variables, which of course can also be classes.

Once the object serialization stream protocol is known, this ceases to be a problem, since the right version numbers are known by the relay.

After the serialVersionUID of any class defined in the properties file is changed, the objects from the received byte stream can be regenerated and it is then possible to send a SASTranferObject, including classes, from the PDA client to the relay. However, in the other direction, the communication from the relay back to the PDA client is less simple.

The reason for this is mainly that the JVM on the PDA throws an InvalidClassException for each class. Theoretically, it would also be possible to change the version numbers on all the classes transferred on the PDA, practically, there are too many and the cost would be too high; therefore it was decided to send strings as a series of bytes. A

very simple protocol was invented where the string is divided into fields with field terminators. The PDA splits up the field and rebuilds the resulting set. This method has proved successful. The disadvantage for the relay, was the necessity of rethinking the data encryption technique, however, this actually turned out to be an advantage. Since the communication is built up with such easy methods, the necessary number of libraries decreased and now requires less than 100 KB.

The WPA encryption, which is supported by the WLAN hardware, is generally sufficient to secure the transmitted data. However, since the client on the PDA is used within a hospital and includes the transmission of sensible patient data (Weippl, Holzinger & Tjoa, 2006), (Miller, 2004), (Mupparapu & Arora, 2004) we decided to include additional encryption. The first technique we considered was an asymmetric method: its security has remained unbroken up until now. However, as we practically expected, a performance test of the RSA algorithm showed that this was not possible in our case. For the test, a very short stream of about 50 bytes was used. The decryption of these few bytes took almost a second. Within the real application, the amount of data sent is often more than one kilo byte. The time delay caused by this encryption and decryption would be unacceptable. Other asymmetric algorithms are as slow as the RSA algorithm, which also made them unsuitable for our client application on the PDA (Salomaa, Rozenberg & Brauer, 1996). An additional, very fast, symmetric encryption, which has not been patented, was implemented in order to ensure against a possible deactivation of the WPA encryption. There are many symmetric encryption algorithms. The One-Time Pad would have been best, but it is not practicable. DES and 3DES are old and also quite slow. IDEA, RC5, RC5a and RC6 are protected through patents. A5 is already broken. The result of this is that only RC4, Blowfish, Twofish and AES were suitable. Of course, there are some other (proprietary) algorithms; however, AES is a common and very fast algorithm.

This new standard algorithm has already been analyzed for security vulnerabilities, which practically ensures that there is no chance of cracking the algorithm within the next few years. Therefore, the AES algorithm was chosen. The implementation of the algorithm was not done within this project. A free implementation of the ACME Laboratories was chosen to encrypt the data. In this project, it is used with a 16 byte (128 bit) key which is calculated from a pass phrase which has to be specified in the property file. The used block size is 32 bytes, the maximum supported by this algorithm. This is acceptable because, although the calculation time needed for the encryption, decryption and the data lookup on the server is up to 5 seconds, the old list is left on display until the new one is loaded. It is possible that the customer might decide that no encryption on top of WPA encryption would be needed and change this in the property file to save time. It is also possible to change the whole algorithm, because this part was built to support other algorithms. The test with non-Standard English characters, such as the German “äöüß”, was also interesting. These characters are encoded differently on the PDA. For example, on the PC the “ü” is represented by a byte with the value of -4. On the PDA, the value for the same character is -127. Other languages with additional character tables will have similar problems. To add support for these languages, an adjustment of the characters can be made after the transfer. Which changes have to be performed can again be specified in the property file which contains all the general settings. This makes it simple to adopt the system for many different character sets.

4 FRONTEND LESSONS LEARNED

At first the nurse logs on; the server then checks the logon information. The BSF communication framework provides the necessary security. After logon, the transport list is shown with the default filter. In order to provide the ability to change this, two option panels are available. Our main design criteria was to build the mobile interface as similar as possible to the existing non-mobile interface on the PC, which the nurses are used. We followed previous experiences on mobile interface design (Holzinger, Sammer & Hofmann-Wellenhof, 2006), (Holzinger, Searle & Nischelwitzer, 2007), (Nischelwitzer et al., 2007) and general experiences on usability engineering (Holzinger, 2005) in order to ensure an end-user centered user interface. On mobile interfaces, generally the keyboard size problem refers to the fact that using the on-screen keyboard significantly reduces the remaining screen size. The log on and the Options panel use Scalable Vector Graphics (SVG). The layout manager expects the whole screen to be available for the widgets. Constraints specify the place and the position as a percentage of the screen. This generally worked, due to the independence from the physical screen size. However, when the keyboard is selected, the program is notified to refresh its screen – to fit the

new, smaller screen; this new rendering causes each component to become smaller, which makes the options panel very difficult to read. After the removal of the keyboard, the PDA must again adjust the size of all components. This takes approximately 3 seconds, slowing down the overall application time considerably, which was perceived as extremely unpleasant by our test users. This problem can be solved by using another layout manager, however, the start up problem still has to be solved, where we followed a three step approach:

The first step is to review the needed libraries. Maybe not all classes of the library are needed. A new library should be built, which only contains the necessary classes. The reason for this is simple, to shrink the size of the libraries which have to be loaded, thereby decreasing load time. This is not very difficult, since the class files contain the information of their super class and interface, the dependency within a package can be discovered and the methods' argument types can also be seen.

The second step is the loading process itself. This prototype loads all elements which are shown on the display at start-up. This mean that all three panels, the logon, the transport list and the options, are all loaded before anything is painted to the screen. A better solution would be to load only the first panel, where the user enters their username and password. To do this, the user needs a few seconds to enter their account details: meanwhile the transport list can be loaded in the background. The benefit of this is simply that the user can start to work after a shorter loading time. From the user's point view of while using the application, this does not make a big difference as the next panel is already loaded when it is needed. The same strategy can be used for the options panel. While the user looks at the screen and uses the functions of the transport list panel, the options panel can be loaded. The only disadvantage of this technique is from the programmer's perspective: it is more work to load something in the background than it is to load everything at once. In this case, it is necessary to use separate threads to control which panels will be needed next and to check whether it is already loaded, or to load it before it is needed, in order to avoid delay. We had to ensure that the system waits until the required panel is loaded; in real-life this will happen very rarely.

The third step is to use separate frames for each panel. This helps to reduce the required memory, because after the user is logged in, the logon frame can simply be removed from memory.

The *not default mistake* means that it is default on the PDA that the close button in the title bar is used for the okay and cancel buttons. An X in the title bar means cancel and the symbol for okay is simply represented by the two characters 'OK'. Because of this, the cancel button generally will not be used very often; and therefore could be removed to the title bar. Experienced users will be acquainted with this. Also, inexperienced users will not have many problems because they normally only need the OK button which is same as on the PC. When the cancel button is needed, a short glance over the rest of the screen will solve any difficulty that may arise.

The options scroll is really one of the bigger problems. This view shows the different options which the nurse can set up. As already described in the options paragraph above, every time the user activates a scrolling action the whole window must be redrawn. If there are only a few components on screen, then the PDA's processor is sufficient to perform the necessary actions of the layout manager. Therefore, it is better to split the panel up into two windows, or panels, so that the scrollbar is no longer needed. A button is necessary to switch between the two panels. This makes sense because in this way it is possible to put the more important widgets on one panel and the remaining components on the other panel, which may not need loading as often. Another possibility is to reduce the number of possible options to a minimum by removing the less important ones and therefore the necessity for a second panel. Other known errors refer to usability difficulties rather than real problems, avoiding these makes it easier for the user to work with the program. A typical example includes the information displayed about the last update or a logout button.

The last update is necessary because if the nurse is somewhere in the hospital without a wireless connection to the server it is not possible to update the transport list. And because the nurse usually gets an update of the list every five minutes it is possible that they may think that the program does not work anymore because the list had not changed for more than ten minutes. In this case, the time of the last update indicates a connectivity problem, which requests that the nurse move towards an area with a wireless connection. This will mainly be an issue at the beginning, later support will be expanded to the entire hospital. The logout button is a time saving service for the nurses to enable them to share PDAs without the necessity of restarting the PDA application each time it changes hands.

When the nurse goes off duty they just log off and the next nurse can login immediately. Another thing which is not directly an error is the cancel button. Because this button has been moved to the title bar it is no longer a real widget which needs

space within a panel or window. So the okay button should be moved to the middle or to the right of the available space to fit the user's expectations. Our last idea was to use different panels for the different screens and to perform any cancel actions using the "X" in the title bar. However, this was not possible. The reason for this is simple, that the default action on the PDA is to hide the window and to show the next one which is below the current window. On the PC it is possible to execute own code when the closing button is pressed. On the PDA this is not possible because no event is generated to perform an action which would prevent the window from being hidden. So some changes were necessary. The first was that every panel gets its own window, except the two option panels, which share one. The options window is the only panel which has a default cancel operation, performed by pressing the "X" on the title bar. This is possible because then the transport list window, which is the window below, is shown. The problem that switching the windows takes longer than switching the panels could also be eliminated by preloading the windows. If a window is needed, it is just shown on the screen because it is already in the memory, which prevents the shimmer. The other two windows no longer have the close button in the title bar to prevent user mistakes. Standard buttons are used to perform the needed actions. This means that the exit button, which was just removed, is again added to the logon panel. Also the logout button within the transport list panel can not be removed. In the transport list panel, one very important button is absent. There is no way to change the direction of the transportation. The patients have to be transported to the radiology and they must also be transported back to their stations. This means that two lists must be displayed. To save space on the screen the button is added instead of using one options button. The loss is that there is now no way to get directly to each of the options panels. To minimize the effects of this disadvantage the options dialog always shows the panel which was used the last time. A next/back button is used to switch between the two options panels. Other modifications which are needed are the logout and direction buttons and the change of the last update time string. The change of the two buttons has two reasons. The first is that the application should be consistent. In the options dialog, the position of the OK button to go back to the transport list is at the bottom left. The logout button is the same within the transport list window. The second reason is that the direction is like a headline. And so its logical position is at the top of the window. An extra window is used for the list, which is the main part of the application. In the top left of the window, there is a button to select the direction. The title of the button shows the currently selected direction. Possible values are "To Radiology" and "To Station". By pressing it, the direction is changed. On the right of this button, there is a label which tells the user when the last update of the list was made. This is necessary because of the automatic update of the list. The time interval of the update is specified in a property file of the client. On starting, the list consists of the patients name, the time to transport, from where and if the patient can walk, which are displayed in two rows. After the selection of one of the items, the item pops out and shows additional information, such as the patient's date of the birth. When one patient is transported, the nurse double-clicks on the patient in the list to mark this as transported. This is shown by the light gray background. After selecting another item the marked item pops in and the color of the font of the selected item turns dark gray. By the next update of the list, the update will have been completed on the server and the item is removed from the list. It is possible to declare a minimal time in the property file, during which an item is not updated to prevent mistakes, it is only necessary to make a second double-click on the marked item in order to change the item state back to normal.

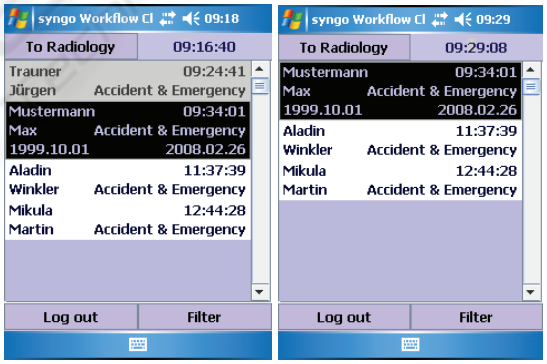

Figure 1: Transport List: Before and after performing a transport.

When the button "Filter" in the transport list panel is pressed, the first options panel is shown. The nurse can specify different options for the filter; e.g. the transport type or the examination date. The transport list button behaves as a back button to the transport list panel; when pressed the expected panel is shown and the contained list is updated to fit the current filter settings. The "Next" button is used to get to the second options panel. The "X" is abort. The second panel is built as the first one. The whole available space is used for components which are used to specify different filter criteria. Examples for

filter options on this panel are the patient name or the handling of already completed transports. At the bottom of the panel there is again a button to return to the transport list and update it with the specified options to match the selected filters. The "Back" button switches to the first options panel.

5 CONCLUSIONS

It is not trivial to develop both a useful and usable mobile application. The general idea to use a J2SE Virtual Machine was abandoned, due to lack of compatibility. The server communication could not be done trough the available methods. This made it necessary to build a relay to forward the inquiries to the server. That made it possible to convert the result to a string which is sent as a byte stream. Due to this sensible data, encryption was necessary. Primarily the WPA encryption is used, which is secure and is supported by the PDA's hardware. Additionally, the data is encrypted by the application. After a comparison the AES implementation of the ACME Laboratories was deemed the optimal choice. Due to the fact that it is the new standard algorithm for symmetric data encryption, the algorithm can be graded as secure. The user interface of the PDA client also caused some troubles. It is not possible to build a slightly more complicated interface when using the default layout managers. It was necessary to program a custom interface to fit all our needs within an adequate time. Another problem was that the PDA had a different window handling. It is not possible to receive any window events within a Java application and that must also be considered. Another big problem on the PDA, which is not an issue on the PC, is the scrolling of complicated interfaces. The scrolling time is much too long because a simple scroll request required a few seconds of time to perform. Everything had to be optimized to save as much computing time as possible. However, some of the challenges of migrating desktop applications to mobile technology are the limited screen size and the originality of the technologies that need to work together correctly for a useful and usable solution. In conclusion, the PDA prototype can reduce the time needed for transports and the end-users were able to use the PDA similarly to their non-mobile interfaces.

REFERENCES

Brightsidefactory (2008), Bright Side Framework Overview. Online available: http://www.bs-factory.org/components/remotingDoc/architecture.html, last access: 2008-06-10

Bruno, E. J. (2005) NetBeans 4.1 & Eclipse 3.1 - Development platforms for J2SE, J2EE, J2ME. *Dr Dobbs Journal,* 30, 8, 14-23.

Freebeans (2008), Mysaifu JVM. Online available: http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html, last access: 2008-06-10

Gale, M. E. & Gale, D. R. (2000) DICOM modality worklist: An essential component in a PACS environment. *J of Digital Imaging,* 13, 3, 101-108.

Holzinger, A. (2005) Usability Engineering for Software Developers. *Comm of the ACM,* 48, 1, 71-74.

Holzinger, A. & Errath, M. (2007) Mobile computer Web-application design in medicine: research based guidelines. *Universal Access in the Information Society International Journal,* 6, 1, 31-41.

Holzinger, A., Sammer, P. & Hofmann-Wellenhof, R. (2006) Mobile Computing in Medicine: Designing Mobile Questionnaires for Elderly and Partially Sighted People. *Springer LNCS 4061*. Berlin, New York, 732-739.

Holzinger, A., Searle, G. & Nischelwitzer, A. (2007) On some Aspects of Improving Mobile Applications for the Elderly. Springer *LNCS 4554*. Berlin, Heidelberg, New York, 923-932.

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

Miller, A. (2004) PDA security concerns. *Network Security,* 2004, 7, 8-10.

Mupparapu, M. & Arora, S. (2004) Wireless networking for the dental office: current wireless standards and security protocols. *J Contemp Dent Pract,* 5, 4, 155-162.

Nischelwitzer, A., Pintoffl, K., Loss, C. & Holzinger, A. (2007) Design and Development of a Mobile Medical Application for the Management of Chronic Diseases. In: Springer *LNCS 4799*. Heidelberg, Berlin, New York, 119–132.

Salomaa, A., Rozenberg, G. & Brauer, W. (1996) *Public-Key Cryptography.*

Sims, B. (2004) Moving from liability to viability. Hospitals, health plans and physician practices can outsmart hackers with policy, a comprehensive security infrastructure and wireless monitoring. *Health Management Technology,* 25, 2, 32-35.

Sun (2004), Object Serialization Stream Protocol. http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/protocol.html, last access: 2008-06-10

Sun (2008), CLDC HotSpot™ Virtual Machine. http://java.sun.com/j2me/docs/pdf/CLDC-HI_whitepaper-February_2005.pdf, 2008-06-10

Weippl, E., Holzinger, A. & Tjoa, A. M. (2006) Security aspects of ubiquitous computing in health care. *Springer Elektrotechnik & Informationstechnik, e&i,* 123, 4, 156-162.