CABLELESS COMMUNICATION BETWEEN MOBILE PHONE, DATABASE AND A SCALE USING THE MSD FORMAT

*J. Orlowski, *K. Biskup, *B. Jettkant, **P. Dültgen, and *B.Clasbrummel

*Research Group Telemedicine, BG-Hospital Bergmannsheil Bochum, Germany
**Chair of production systems, Ruhr-Universiy Bochum, Germany

joerg.orlowski@rub.de
karsten.biskup@rub.de

Introduction

In the health care sector decision makers and health professionals are increasingly concerned with utility, quality and costs of medical-therapeutic procedures. Telemedicine makes use of tele-communication and information technology during the diagnosis and treatment of patients at home. The birth of telemedicine is accompanied with space travel [1].
Medical technology applications and equipment are characterized by an increasing functional integrity. The individual systems are mostly incompatible and implemented with manufacturer-specific, non-standardized technical solution. This leads to inflexibility and prevents patient-scaled and individual customization of the devices
The wireless, safe transfer of medical data is becoming ever more important under today's conditions. However, proprietary protocols and isolated solutions from several suppliers make it impossible to receive and correctly interpret data by any medical device. For this purpose, the MSD format was developed, which puts all required information in a uniform format.
To demonstrate the technical feasibility, a P900 from Sony Ericsson was programmed such that data from a scale sent to a Symbian mobile phone via bluetooth converted to the MSD format and transmitted into a database. XML documents from the server are transmitted to the phone hand, parsed and displayed.

Materials and Methods

Figure 1: the setup: scale, mobile phone, Linux server

First we list all individual components of our mobile home care setup with their specifications. As a simple corresponding example scale values are wireless transmitted and transformed into a MSD format on a mobile phone and then stored in an external server database.

Scales with RS-232 interface

We have looked for low cost (less than 200 US$) scales with a measurement range up to 120 kg and a RS-232 interface.on the international market.
The platform scale DE-150K-50N from KERN & SOHN GMBH is designed as a multifunction balance, intended to be used manually, i.e. the material to be weighed is carefully placed in the center of the plate. The weighing value can be read off after a stable weighing value has been obtained. With a weighing range of 150 kg, a reproducibility of 50 g, a linearity of 150 g, and plate dimensions of 315 x 307 x 73 mm it can be used for non dynamic weighing of persons individual mass. A warm-up time of 5 minutes stabilizes the measured values after switching on. The accuracy of the balance depends on the local acceleration of the fall. The metrology features must be checked at regular intervals. For battery operation the balance has an automatic power off function which can be activated and deactivated.
A data output is realised via an interface RS-232-C (9 PIN D-Sub). As a condition for the data transfer between the balance and a peripheral device both devices have to be set to the same interface parameters (for instance 8-bit ASCII Code, 1 start bit, 8 data bits, 1 stop bit, no parity bit, 9600 baud). The balance must be disconnected from the mains before connecting or disconnecting additional equipment to or from the data interface. This scale can be set to four different modes:
Pr PC: a reading will be transmitted, only if the PRINT key is pressed and the weight is stable.
AU Pr: if a stable reading comes up, the reading will be sent once automatically.
AU PC: weight readings will be sent automatically and continuously, no matter the weight reading is stable or unstable.
rE Cr: Remote commands s/w/t will be sent from the remote to the balance as code. When the balance received the s/w/t command, following data will transmitted.

ISSN: 1727-1983

Another low cost but durable scale is one of the HD Series from Fa. Myweight with 30 year warranty. These scales are designed for shipping, weight checking, and all light industrial uses. The HD-300 comes with an AC Adaptor and RS-232-cable but it can also be powered by AA batteries for portable weighing. The HD-300 can be used standalone or read out by a PC. This scale can be set to a UPS® and USPS DAZZLE® compatibel mode. The HD-300 has a built in RS232 port with single direction interface protocol (9600 baud, 1 start bit, 8 data bits, 1 stop bit, no Parity).
If the mode "SCI.0" is selected and button "DATA" is pressed or the carriage return (0d hex) is received then the scale transmits 14 bytes ASCII data each time: weight, unit, stability, over/down or low voltage message according to the following protocol: The string begins with the start Byte1 ":" and terminates with Byte 14 a carriage return (0d hex).
Byte 2: "W" for weight, "M" for Message OVER
Byte 3: signature
Byte 4 – 9: weight value in fixed decimal format
Byte 11,12: weight unit, lb or kg
Byte 12 : "S" if stable or " " if unstable
Byte 13 "L" means the low voltage

Bluetooth Cable Replacement

The popular RS-232 standard interface is used in many medical and industrial controls and devices. But cables are limiting the users freedom and the installation at home is time expensive.
Bluetooth adapters are active devices with integrated microcontrollers for RS-232 are nearly perfect wireless solutions for RS-232 communication cable replacement. Due to the low radiation and high security in data transfer, Bluetooth is useful for medical applications, measurement devices or embedded systems as well.
Such BlueSerial adapters as from Fa. Hantz und Partner GmbH, Gundelfingen, are working independent of Operating System allowing a RS-232 cable replacement up to 100 m depending on the Bluetooth radio Class (1 or 2) and the antennas used.
The max. range is obtained in an open field and the penetration of walls depend on the thickness and the material of which they are made. But typical 40 m can be reached in house.
An automatic detection of RX/TX and DTE/DCE is integrated. Hardwarehandshake could be switched off for "dumb" devices like sensors. This adapters offer an one-time-setup and pairing between two or more Bluetooth enabled devices. There is an automatic device recognition and profile detection as well. New adapters can be wireless configured using a second Bluetooth adapter on a PC or notebook. There is no need to program own Bluetooth profiles or stacks. This features allow an easy integration of new RS-232 BT modules into own systems and devices to get fast and reliable connections.
Fa. Stollmann Entwicklungs- und Vertriebs-GmbH, Hamburg has developed serial BlueRS+ Bluetooth adapters as stand-alone modules for medical data acquisition with baud rates from 300 bps up to 230 kbps and four simultaneous connections. They have been certified as Bluetooth products.
The adapters have a Bluetooth software integrated and can be configured via the serial interface or via the Bluetooth connection. Using well known AT commands more than one Bluetooth connection can be polled sequentially from a terminal device.
The functions of these adapters can be set into an automatic mode or into a multipoint serial bus emulation. To establish connections from a single adapter to several terminals Stollmann is offering the standardized TS 101 369 (GSM 07.10) protocol.
The power consumption depends on the operating status. Full power of up to 70 mA current is relatively short required only during transmission phases. In between they go automatically (<15 ms) into a latency phase and consumes at minimum 4 mA in the power down modus. The module can be reactivated via the serial interface or via Bluetooth.
So far some medical application have been developed with the Stollmann Bluetooth interfaces, e.g. an online ECG.
For a high optimized integration into established standard devices external and internal serial Bluetooth modules and chipsets are available.
Single-chip solutions or additional integration in customer solution can be made based on their own portable Bluetooth upper layer stack. Windows device drivers and software for serial communications up to seven concurrent BT connections are available.

MSD Format

In 2001 the initiative "Implantable and Extracorporal Modular Microsystem Platform (IMEX)" inside VDE was started to analyze widely used interfaces between components of modular constructed medical micro systems [2]. Based on this evaluation the need for a new data format to be used among micro systems and external evaluation and storage devices was identified. A scalable data exchange format was designed for data transmission between components of medical micro systems and their communication partners. These partners can be other micro systems, communication gateways or computers. Its flexibility allows using established communication hardware, protocols and objects. The communication hardware is thereby not taken into consideration. This specification of the MSD-format (micro system data format) has been developed and published in the framework of the International Electrotechnical Commission's (IEC) Industry Technical Agreement (ITA) program [3]. The new data format should meet requirements from the new nature of information delivered by micro systems.
The MSD format is scalable, thus supporting a wide range of granularity from one single measuring value to large aggregates of data. Existing data formats can be transparently embedded into MSD files and messages. MSD-structured data can be transparently transmitted using any existing communication system [4]. Additional information indicating the origin of data, compression, encryption, transformation method, time

ISSN: 1727-1983

base and other information can be added in a header. The dynamically structured MSD format can be coded both binary and XML-consistent [6,7,8]. We attach great importance to data integrity.
The *container* is the top-level structure generally related to the handling of data exchange between the transmitter, the gateway and/or the receiver, which indicates information about sender, address etc. in a network. The *sections* are embedded in the container in a certain order. They represent a sorted collection of data. The data itself are embedded into blocks. A section may contain blocks which are produced by different individual parts of a systems. The *block* contains the raw data and a supplementing description (heading) as well as provisions for data integrity. The *block* ensures the safe transport using information about the data structure, sequence processing priority and life cycle.

All three parts can be organized, once again into similar modules. *Message*, *Container* and *Block* follow the basic construction: Header block (static and dynamic), Data object and Data integrity.

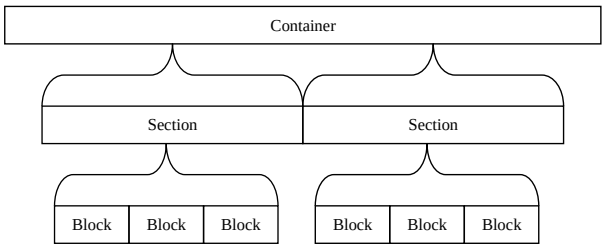

Figure 2: global MSD format structure

The notation of the format can be chosen. The notation ranges from binary / hexadecimal order to a XML compliant order. In case of using a binary order the Header of the Block begins with 3 information bytes. The first one (Data Structure) contains information, as whether it is a *Container*, *Section*, or *Block*, byte order, indication of handshaking etc. Every byte ends with one extension bit. The second byte contains among other things the priority. This is important, when several different data streams are sent simultaneously. A LifeTime ensures that the receiving end confirms the receipt within the LifeTime, otherwise the packet can not be considered as have been received. With this, real time application can be established. A set TimeDivisor defines the underlying time unit as part of a second. Finally sender's name can be set as a string. Clarity can be guaranteed using serial numbers or MAC addresses. Actual details such as date stamps are attached to the dynamic part of the header protocol. The third byte contains further information whether specific details are available in the dynamic part, for instance whether data was compressed or a checksum had been attached. The dynamic header varies in the length, depending on the status bits set in the static header. It always begins with the version number and the type of encoding.
A continuous numerical 32-bit signed integer BlockCounterNumber identifies the unique sequence of the Blocks. Since synchronization of all communicating partners is not feasible, the sequence of individual Sections cannot be determined, using UTC time stamp. The BlockType indicates the type of the data object using four consecutive letters. A common file extension, generally identical with Windows/ DOS file extensions permits automated processing. The next bytes are about priority and LifeTime, access right ad-ministration, all based on UNIX. Almost important is the length of the data object for validation of the complete transmission. The value refers exclusively to the decoded user data in the data block (BlockDataObject).
Subsequently, the actual data object follows. This is later stored in the database as referenced file. The checksum upon the data object is calculated and stored as an array of char. The width of the array is determined by the applied CRC method (CRC8-CRC128).
It is possible to store a crypto-signature in the Header of *Container*, *Section* and *Block*. The encryption can be added by means of the cipher-status information. Only the file contents are encoded. Symmetric, public key and hybrid algorithms are supported.
A new and important supplement to the Container/Section/Block is the specification of the address. The address of a sender, receiver, router and so on is defined as a character string without \0-termination. Beside all alphanumeric 7 Bit ASCII characters all 32-bit Unicode characters are allowed, however exclusively in hexadecimal representation. Other special characters are used for operators. Lists of comma-separated addresses in brackets ( ) can also be created. It is possible to give the address in different formats, i.e. in decimal, binary or hexadecimal form, as IP format or as a substring. In doing so, the respective format must be clearly identified using a prefix. As to be shown, operators are used for routing [< > / \ ], for delimiting of addresses or lists [, | ( )] or for priority definition [:]. In all these cases, the operator associativity is from right to left. The path is the description of the route. The layout of this "way description" is as follows: Addresses(s) Operator(to from) Address : Priority (LISP like).

Since the deployed medical devices can also act as actuator applications, Block Execute defines the executable commands for vendor-specific control data or instruction tables. Remote maintenance-related problem definitions can be declared using the Remote Section. By means of the Block Remote Information-Header, the device is able to detect whether the transmitted data is a command for remote maintenance or not.
As a result a unified format is recommended: scalability from 8 bit to 64 bit processors, amount of data from some characters up to several 100 MB data, embedding of existing data formats (like JPG, DCM…), identification of communication partners [9,10], timestamp and date in UTC, chronological order of messages, prioritization of messages, integrity of messages, differentiation between data, commands, execution and remote access, encryption and dig. Signature, path of transmission, sender name, address, router, priority, timeout values for transmission and execution (real time), event markers, handshake support

ISSN: 1727-1983

for transmission, support for actuators, sensors and data transmission.

Symbian Handy

Mobile phone will be able to transform XML data (e.g. MSD format) if additional software enhances the functionality of them written in an advanced language.
We decided to use the JAVA version „J2ME“ from SUN because the P900 can execute JAVA based programs. This version is adapted to the capabilities of mobile phones. SUN distribute a free development package with emulator, which significantly cuts development costs [11,12,13].
By using this development environment, it was possible to implement the functions for interpretation of the MSD format in XML and access the database.
To enable Java on mobile devices the developers must restrict many possibilities. For both, personal JAVA and J2ME simplified safety restriction are applied. Those security functions in personal JAVA are more restrictive as those for J2ME. Furthermore there is no guaranty running the same personal JAVA applications on different cell phones. Portability by using personal JAVA is not naturally given as field-tests show. We chose the JAVA Version J2ME from SUN, to have the possibility to run those JAVA programs on different mobile phones. The use of such SDKs instead of plain C++ increase the development effort extensive [14].
There exist a lot of tools to handle XML with J2M. In general a compromise has to be found between size and functionalities. For processing XML on embedded devices licensed parsers like NanoXML, TinyXML or kXML2 distributed under common public license, may be used to get the structured content. The first two reads the complete XML document into local memory and therefore high memory requirements are needed. The pull-parser kXML2 is best. The parsing results are extracted on fly from the input stream from server by the controlling application. Especially this enables recursive methods acting on the document's tree structure for parsing.

Server, Database

In the background we build up a native command line Linux server, connected to the Internet. As database we chose the free available and extreme stable running PostgrSQL All server applications for handling the data streams to and from the mobile phone are build with SUN's Java developer edition [9].

Results

First let's have a formal look on scale's output data imbedded in the MSD Format as a block.

```
<MSD_Block>
 <MSD_BlcHeader
    ByteOrder = 'Intel'
    Handshake = 'no'
    Version = '1.0.0'
    DataCoding = '2'
    Number = '1'
    Type = '.txt'
    TimeStamp = '2005-09-16 07:59:59'
    TimePart = '0'
    Path = 'h008822FF0002>h008822FF0001'
    Length = '9'
    Name = 'KERN DE-150K-50N MyMass'>
 </MSD_BlcHeader>
 <MSD_BlcData
  Data='89.67 kg'>
 </MSD_BlcData>
<\MSD_Block>
```

Figure 4: MSD format for scale in XML notation

Every Bluetooth device has its own unique identification, the Bluetooth device address. This is mostly represented as a 12 digit hex number e.g. 00-88-22-FF-00-02 which is converted to a more short machine readable MSD address string h008822FF0002. The Bluetooth module at the serial port of the scale with address 00-88-22-FF-00-02 transmits data to the Bluetooth chip set configured to the address 0-88-22-FF-00-02. The time stamp is represented in ISO Format, accepted by many databases and operating systems. Scale's US-ASCII text value (`DataCoding='2'`) is 9 Bytes long, including the sign. No CRC16 checksum was generated.

Figure 3: from server incoming XML parsed content

Current mobile phones are capable of processing data at acceptable speed and due to their JAVA, C++ and Bluetooth capabilities they can communicate with different devices. Another advantage is the free programmability, which allows to turn a phone into a device for medical home use. To demonstrate the technological feasibility, a Sony Ericsson P900 mobile phone has been programmed to receive and display formatted XML data from a relational database.
Files can be exchanged with a server via HTTP by J2ME applications but never directly by email, because this is part of the restrictions of the JAVA sandbox model. There are no restrictions in programming directly under C++.

ISSN: 1727-1983

Currently, the P900 includes Bluetooth and Advanced Messaging APIs for J2ME and Symbian C++ [15].
All incoming structured XML information from the server (e.g. health care information) can be parsed on the phone in JAVA. The Scale values can be transported via Bluetooth to the mobile devices, programmed in C++ and via GSM to a server.
Access to the Bluetooth serial port emulation protocol RFCOMM is provided by the Symbian OS socket architecture [16,17]. RFCOMM, emulates the RS-232 serial port, data is sent in stream and it is quite useful for easy communication between devices. This part explains where to find and how to write a simple cel-phone Symbian C++ program using Bluetooth RFCOMM for a point to point connection.
First you need to do a discovery using the `OnFindDevices` method well known as part of the menu selection. Functions of the TBTSockAddr class allow a client application to get and set the socket address using the 48-bit Bluetooth address structure
Second you do a service discovery with the search pattern ID 0x1101 for the serial port service. The Bluetooth device discovery process has first to identify the address of the device offering the RFCOMM service and the channel on the receiving device. The client application then composes the address of the device it wishes to connect to.
Third start the Bluetooth socket connection, open the socket and set the port and device address to those found previously. In the point-to-point access the `CMessageClient class` connects and opens a socket on the RFCOMM. Once a connection has been accepted, `CMessageServer` must ensure that the RFCOMM entry in the SDP database is marked as being used to avoid a second client connecting to the same port. It is simply an updating of an attribute value in the service record.
Forth if always connected, we are now able to receive or send data. Once a connection with a client device (here the scale) has been established, the receiving phone needs only to call the `RecvOneOrMore` function on the socket the client has connected to. This will call `CMessageServer::RunL` function when data has been received. Sending data (e.g a carriage return to the scale) over an RFCOMM connection is as simple as writing. The `CMessageClient::SendMessageL` function performs this task.
Fifth to gracefully disconnect from the device calling the `Disconnect` method of the socket object.
To reduce the energy, shutting down of the Bluetooth socket is done by calling `close` on the `RSocket` object. Before doing this, all RFCOMM services have to be removed from the SDP database.
Bluetooth connection can be made using API's from the JAVA specification request (JSR) 82. It's a challenge using this JSR due to their up-to-dateness. With the published J2ME wireless toolkit JSR82 in version 2.2 beta or higher Bluetooth connection can be used without further installations. Helpful examples of programming Bluetooth with JSR are given in the documentations.
After getting scale's data and generating the MSD data format on the mobile phone we have to show how this strings can be written to a server. Examples in Java can easily be found in the corresponding documentation but not one in Symbian C++. Therefore we place our tested code (Symbian60) here:
First define a new `ClientSocket` using 4096 byte send and receive buffer. A `Listener` is then implemented, only used if a connection is well established. The `OniTCPClientSocket1Connect` as a method is then called doing this. At last basic settings like port and IP of the server must be defined.

```
void CSocketAppContainer::InitComponentsL()
{
iTCPClientSocket1 =
CTCPClientSocket::NewL(4096,4096,0);
TEventT < CSocketAppContainer >
iTCPClientSocket1_OnConnect( this, &
CSocketAppContainer::OniTCPClientSocket1Connect
);
iTCPClientSocket1->
SetOnConnect( iTCPClientSocket1_OnConnect );
iTCPClientSocket1->
SetServerName( _L( "82.83.151.53" ) );
}
```

To show a simple text on phone's display whats going on with the connection we make use of an other method:

```
void CSocketAppContainer::
OniTCPClientSocket1Connect(CBase * aSocket)
{
 iEikLabel1->
 SetTextL(_L("connection established"));
}
```

A data transmision can be done with the function `WriteL` similar as "write" in java. First step: define a descriptor containing the data, then change the display to show someone that something happens and finaly transmit the interesting data.

```
void CSocketAppContainer::
OniMenuItem2ViewCommand( TInt aCommand )
{
 _LIT8(Puffer,"0123456789012345678 9");
 iEikLabel1->SetTextL(_L("transmitting data"));
 iTCPClientSocket1->WriteL(Puffer);
}
```

It's clear at last we have to close the transmission, and to change at least the message on the display.

```
void CSocketAppContainer::
OniMenuItem3ViewCommand( TInt aCommand )
{
 iEikLabel1->SetTextL(_L("disconnected"));
 iTCPClientSocket1->Disconnect();
}
```

Understanding this data transfer example one should be able to program own applications (e.g. exception handling, byte counter, check sums, error messages and so on).
The development of 2ME applications for cel-phones are typically easier than those in corresponding C++ or Java. In addition, these applications can be programmed to be compatible with other J2ME-enabled mobile phones.
Present PDAs, on the other hand, are not suitable for programming in JAVA because the JVM (JAVA virtual machine) provides only rudimentary functions here and no further development progress has been made.

ISSN: 1727-1983

Conclusions

The MSD standard defines a modular data format to be used for transmission and storage of sensor-, actuator- or micro system-related information. Potential areas of application include medical, environmental and industrial data acquisition, process monitoring, automation and control. The use of the MSD format in mobile terminal devices was demonstrated successfully by the example of a scale to which a Bluetooth transmission and receiving unit has been added.
The MSD format provides the basis for an open interface across system borders, for communication between different medical devices and applications.
The most important advantages of the MSD Format can be summarized as follows. Arbitrary data contents can be sent and is enclosed and expanded by additional information. Missing information are skipped. Future manufacturer-specific protocols can be simply integrated. The XML-consistent tagging allows a simple and structured reading into the source code. Furthermore the contents can be better imported into the database. The MSD data format, which is currently going through international standardization committees, is characterized by extreme flexibility and expansion capability. The authors hope for a wide acceptance of the standard and its successful application not only in medicine.
To confirm such a home care scenario a study will be necessary to show the practical capability.

Acknowledgements

The IMEX joint-project is promoted by the Bundesministerium für Bildung und Forschung (BMBF) from June 2003 to Feb 2005 grant no 16SV1594. The Televisite development is part of the project Teltra, supported by BMBF/DLR grant no 01EZ0016

References

[1] BISKUP, K., BOLZ, A et al. (2002): ‚Poststationäre Patientenbetreuung durch Televisite', Biomedizinische Technik. Biomedical Engineering Supp 2002; 1: 354-355

[2] 'Bestandsaufnahme der Mikrosystemtechnik für telemetrische Anwendungen in der Medizin', Institute of Healthcare Industries, Steinbeis Hochschule Berlin, Studie 2003

[3] IMEX MSD-Format, verfügbar als ITA bei IEC unter: http://www.iec.ch/searchtech/italst-e.htm

[4] B. JETTKANT, P. DÜLTGEN, K. BISKUP, B. CLASBRUMMEL (2004): 'Micro System Data Format MSD for Interoperability between Wearable Devices', Biomed. Tech. 49, 2004, p. 244 - 255

[5] ORLOWSKI, J., FRIELING, C., MANSOUR, O., BISKUP, K., JETTKANT, B., CLASBRUMMEL, B. (2004): 'Terminmanagement für die Telemedizin, ein Zusammenspiel von XML, XSL, Datenbanken und Cocoon', BMT, Vol. 49-1, pp. 242-243, 2004

[6] Extensible Markup Language (XML) 1.1, W3C 2004, http://www.w3.org

[7] FRIEDRICH, S., KLOTH, A., SCHÜLER, J., SOMMER, K. (2003): 'XML-basierte Frameworktechnologie', http://imp.blubbmon.de

[8] E.T. RAY (2004): 'Einführung in XML', O'Reilly, 2004

[9] MOMJIAN, B. (2001): 'PostgreSQL, Einführung und Konzepte', 2001

[10] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, Internet Engineering Task Force Working Group (20049 http://www.mhonarc.org/~ehood/MIME/2045/rfc2045.html

[11] M. KROLL, S. HAUSTEIN (2003): 'J2ME Developer's Guide', Markt+Technik, 2003

[12] S. ESCHWEILER (2003): 'Java 2 Micro Edition', vmi, 2003

[13] M. DE JODE (2004): 'Programming Java 2 Micro Edition on Symbian OS', Symbian Press, 2004

[14] N.N., 'Developing P800/P900 applications, API summary', technical document, Sony Ericsson Mobile Communication AB, 2004 http://developer.sonyericsson.com

[15] A.N.KLINGSHEIM (2004): 'J2ME Bluetooth Programming', Master's Thesis, , University of Bergen; 2004

[16] 'Symbian OS: Designing Bluetooth Applications In C++ Version 1.1', Nokia Corporation. 2005

[17] CHRIS DOUBLE (2004): 'Developing for a Bluetooth GPS on Symbian Series 60 cellphones with C++', http://www.double.co.nz/symbian/bt/borland_symbian_bluetooth.html

ISSN: 1727-1983