OPENHEALTH
The OpenHealth FLOSS Implementation of the ISO/IEEE 11073-20601 Standard

Santiago Carot-Nemesio, José Antonio Santos-Cadenas, Pedro de-las-Heras-Quirós and Jorge Bustos
GSyC/Libresoft, Rey Juan Carlos University, Tulipan S/N, Móstoles, Madrid, Spain

Keywords: FLOSS, ISO/IEEE 11073-20601, Personal health, Android, Bluetooth, MCAP, HDP, Continua health alliance.

Abstract: The OpenHealth project aims to provide the first complete Free/Libre Open Source Software (FLOSS) implementation of the ISO/IEEE 11073-20601 personal health standards over Bluetooth transport. A manager has been implemented in Java, tested both in Linux desktops and on the Android platform against thermometer agents. The MCAP and HDP Bluetooth profiles have also been implemented in BlueZ, the offcial Linux Bluetooth protocol stack. This implementation has been successfully tested with a Nonin Onyx II 9560 pulse oximeter, the first medical device that implements the HDP Bluetooth profile and the ISO/IEEE 11073 data protocol.

1 INTRODUCTION

The OpenHealth Assistant project (Andago Ingeniería S.L., 2009) being developed by Andago Ingeniería and the GSyC/LibreSoft research group at Rey Juan Carlos University aims to deploy a full socio-sanitary platform to fulfill the needs of current public health assistance services. Figure 1 shows a diagram of the architecture of that project.

The Manager (GSyC/Libresoft Research Group, 2009) described in this paper [1] is one of the components of this project, aimed at managing and recollecting vital measures sent from medical devices (agents) connected through body area networks (BAN) such as Bluetooth, and providing support for sending the received measures from the manager application to the hospital backoffice or to online personal health record applications in an interoperable and standard way.

The ISO/IEEE 11073-20601 (Institute of Electrical and Electronics Engineers Inc., 2008c) belongs to the ISO/IEEE 11073 family of standards for device communication. It defines a common abstract communication model based on agents and a manager to transmit transport-independent personal health data. The new standardized Bluetooth Health Device Profile (Bluetooth Special Interest Group

Figure 1: OpenHealth Assistant Platform.

Inc., 2008a) (HDP) in combination with the IEEE 11073-20601 specification provides a robust, standards based framework to allow interoperability between Bluetooth Health Devices. Figure 2 represents the protocol model used by this profile.

Advances in mobile computing bring us the possibility to improve socio-sanitary assistance by running the manager on smartphones, tablets, subnotebooks,

[1]This work has been partially funded by the Catedra Internet del Futuro Andago-URJC and by the Spanish Ministry of Industry's Plan Avanza

Carot-Nemesio S., Antonio Santos Cadenas J., de-las-Heras-Quiros P. and Bustos J. (2010).
OPENHEALTH - The OpenHealth FLOSS Implementation of the ISO/IEEE 11073-20601 Standard.
In *Proceedings of the Third International Conference on Health Informatics*, pages 505-511
DOI: 10.5220/0002766705050511

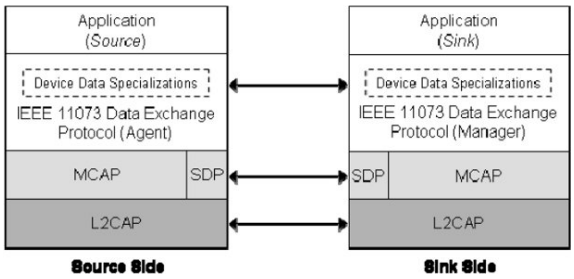

Figure 2: Protocol Model.

etc. These are adequate devices for hosting a manager due to their diminishing prices, their widespread usage, their small form factor and their multiple communication channels.

The wireless communication model for the ISO/IEEE 11073 family uses the new MCAP and HDP Bluetooth specifications. HDP and MCAP profiles require new developments in lower layers of the Bluetooth stack, specifically in L2CAP, that are not available in current commercial Bluetooth stacks.

We decided to implement HDP/MCAP for the Linux kernel, with the aim to facilitate the inclusion of this code in multiple mobile platforms that use this kernel, such as Nokia/Maemo, Android, Intel/Moblin. The official Linux Bluetooth protocol stack (Bluez, 2009) was a good choice. Developments made by other members of the BlueZ project in parallel with our project made it possible to use the L2CAP special transmission modes that HDP/MCAP requires.

This paper describes a Java based Free/Libre Open Source (FLOSS) implementation of the ISO/IEEE 11073-20601, as well as a FLOSS implementation of the HDP/MCAP Bluetooth profiles, that allows us to run a manager on Android devices. Both of them are available in the project's website [2]. We decided to target specifically the Android platform because it fulfills all our requirements: it is Open Source, based on a Linux kernel, and it is a mobile platform available in multiple devices.

Figure 3 shows a diagram of the architecture of the stack we have implemented. Parts of it will be described in sections 2, 3 and 4.

Section 2 describes the Java based ISO/IEEE 11073-20601 we have implemented. Then, section 3 details the Android Manager Service, which is a layer on top of the manager designed for the Android OS, implemented as an Android service that enables other Android applications to control the behavior of the agents connected to the manager. Finally, in section 4 we first introduce the HDP and MCAP Bluetooth specifications which have been recently adopted by Continua Health Alliance for the wireless transmis-

[2]http://openhealth.morfeo-project.org/lng/en

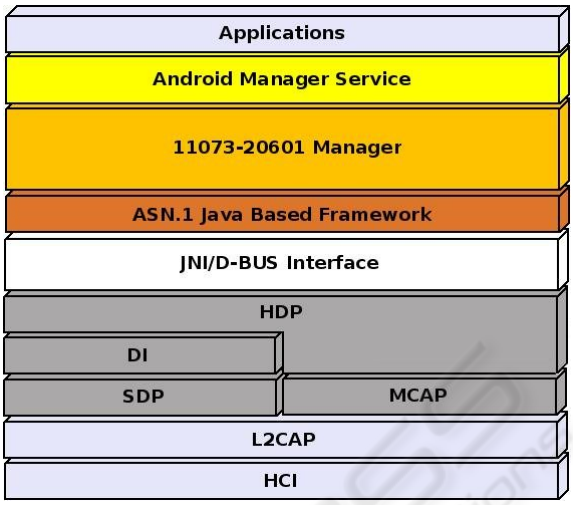

Figure 3: Android Manager Service + ISO/IEEE 11073-20601 Manager + BlueZ (HDP/MCAP) architecure.

sion of standard data records in compliance with the ISO/IEEE 11073 standard, and then we detail our particular implementation of HDP/MCAP for the BlueZ Linux Bluetooth stack. The JNI/D-BUS interface is the only component shown in figure 3 that has not yet been finished. The bottommost 2 layers shown in the figure, L2CAP and HCI, have not been implemented by us.

Experimental tests and results are reviewed in section 5. The paper concludes reviewing related work, future work and conclusions.

2 MANAGER IMPLEMENTATION

The Manager is the main software component of the architecture. It plays the sink role to collect the data sent from one or more agent systems, and it manages the communication process in order to ensure that the personal data exchange occurs according with the optimized data exchange protocol.

All protocols defined under ISO/IEEE 11073-20601 use abstract syntax to provide an specification of the structure of data items without reference or requirement for a specific implementation technology for medical devices communication. ASN.1 (International Telecommunication Union, 2002), together with specific ASN.1 encoding rules, facilitates the exchange of structured data by describing data structures in a way that is independent of machine architecture and implementation language.

The ISO/IEEE 11073-20601 uses optimized encoding rules for ASN.1 known as Medical Devices Encoding Rules (MDER). MDER was specified in ISO/IEEE 11073-20101 (Institute of Electrical and

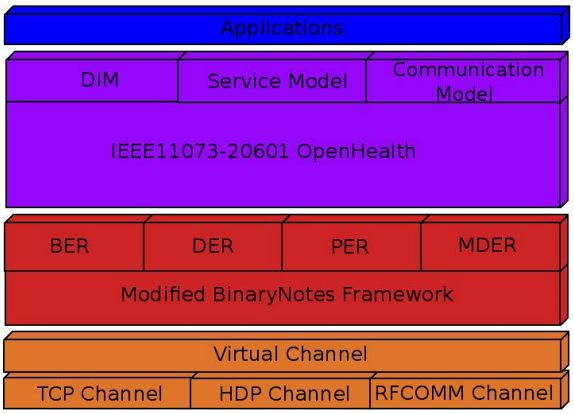

Figure 4: Manager architecture.

Electronics Engineers Inc., 2004) to minimize bandwidth in transferences of dynamic personal health data between agents and managers.

The lack of Open Source implementations of medical encoding rules for ASN.1 motivated us to extend the GNU/LGPL BinaryNotes (Abdurakhmanov, 2008) ASN.1 framework for Java, implementing the support for the new encoder and decoder rules for medical devices on this framework. Although MDER is mandatory by both the agent and manager, devices can optionally establish other encoding rules beyond MDER, such as PER and XER. BinaryNotes provides native support for BER, PER and DER. The Manager uses the BinaryNotes libraries to code the applications protocol data unit (APDUs) using the above encoding rules in addition to the MDER rules that we implemented. Other closed source implementations of ISO/IEEE 11073-20601 managers only provide support for medical devices encoding rules.

Figure 4 illustrates the main components integrated in the OpenHealth Manager implementation, corresponding with the main components of ISO/IEEE 11073-20601.

The Domain Information Model (DIM) constitutes the central core of the manager implementation. It defines a set of classes for modeling agents like objects containing data sources as vital signs, events and alert reports, etc. It defines their relationships and the methods that a manager can use to control the behavior of the agent system.

The personal health device service model defines the mechanism for data exchange services. These services are mapped to messages coded using ASN.1 that are exchanged between agent and manager.

Finally, the communication model is implemented through a finite state machine (FSM) to synchronize the messages exchanged on Agent-Manager peer connections over different conditions. Although current implementation only provides support for the thermometer agent specialization (Institute of Electrical and Electronics Engineers Inc., 2008b) in the basic FSM interaction, we expect to provide support to pulse oximeters (Institute of Electrical and Electronics Engineers Inc., 2008a) too in the near future.

To keep transport independence-media at the data exchange protocol level, we have incorporated to the current manager design an abstract communication model based on virtual channels. A virtual channel can manage simultaneous channels at the same time with each Agent-Manager peer connection. Furthermore, a channel in a virtual channel can be either TCP, RFCOMM or HDP, thus providing a common communication interface to the manager application for sending and receiving APDUs from/to the agent system.

3 ANDROID MANAGER SERVICE

Android applications are written in Java, what makes it easy to integrate the manager on a device running Android OS or other Java virtual machine. The Android Manager Service provides an API that enables programmers to develop new manager applications in Android by using the services specified in the ISO/IEEE 11073-20601 standard in a highly abstracted way.

The *service* mechanism provided by Android allows to run applications in background for an indefinite period of time, running in parallel with other basic system services. This feature brings us the possibility to integrate the Java manager described in section 2 in Android as an Android service, waiting in background for agent connections. Furthermore, it's possible in Android to connect external applications to a service, what facilitates the programming of applications that interact with agents.

Figure 5 shows the Android Manager Service architecture. The Android Manager Service exposes notifications about internal events received from agents through transports such as Bluetooth, and it provides an action service interface to manage the state of current agents connected with the Manager.

Because the Android Manager Service is designed for mobile devices, we try to minimize the energy consumption by avoiding polling be made from external applications. All communications from/to the Manager Service is done using a callback service. The Manager API is defined using AIDL (Android Interface Definition Language) to generate code that enables two processes on an Android-powered device to communicate by using inter-process communication (IPC). The AIDL IPC mechanism is interface-

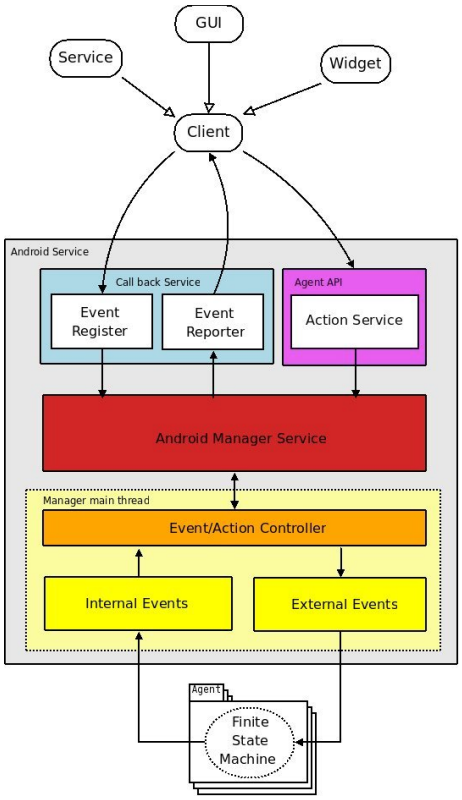

Figure 5: Android Manager Service Architecture.

based, similar to COM or CORBA, but lighter. It uses a proxy class to pass values between the client and the implementation.

Applications use the Manager callback service to get notifications about generic events in the manager such as new agent connections and disconnections. When applications want to know the detailed state of a certain agent, they register with the agent callback service. Later, applications will get notifications about the agent state changes and will receive notifications about measures sent from agents.

The Action Service provides an interface to the application layer that enables applications to invoke services specified in the Domain Information Model defined in ISO/IEEE 11073-20601. Applications use the Action Service to gain access to GET and SET methods over PHD-DIM classes and to send events to control the behavior of the agent system.

An Android service executes the code of the manager in a thread running in background, waiting for agent connections. The manager thread uses the Component Event/Action Controller to exchange events between agents and applications. Applications send events to control the finite state machine of the agents (we call these external events). The Event/Action Controller component is in charge of

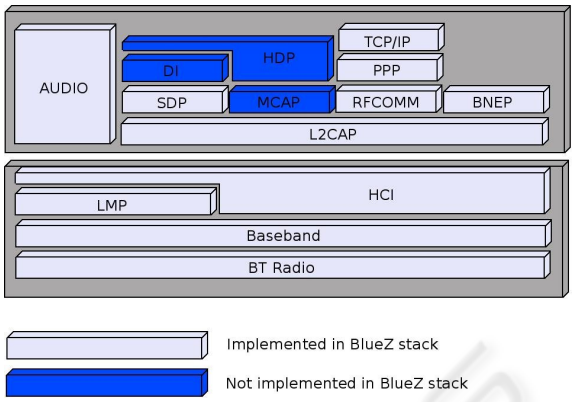

Figure 6: BlueZ Architecture.

delivering the external events by redirecting invocations to the appropriate agent. On the other hand, events triggered by agents (we call these internal events) are delivered to the applications registered for these particular events.

4 THE BLUETOOTH HEALTH DEVICE PROFILE IMPLEMENTATION

ISO/IEEE 11073 protocols can use several transports to send and receive personal health data. In this section we describe our implementation of the HDP, MCAP and DI Bluetooth protocol profiles. Figure 6 shows a block diagram of the bluetooth stack architecture, including the new components implemented by us that were not previously present in BlueZ.

The Bluetooth stack includes the Multichannel Adaptation Protocol (MCAP) (Bluetooth Special Interest Group Inc., 2008b), the Health Device Profile (HDP) and the Device Identification Profile (DI) (Bluetooth Special Interest Group Inc., 2007). None of them were implemented in BlueZ until today.

MCAP is a versatile L2CAP-based protocol that provides a simplified way to use and manage multiple data channels. The protocol supports different channel configurations depending on the application or transmission needs such as for example streamming or reliable modes.

HDP is a profile that simplifies the use of MCAP and announces to other devices the supported features and profiles as well as the channels used for each profile. For these announcements it uses the Service Discovery Application Profile (SDP) (Bluetooth Special Interest Group Inc., 2001). This way, the devices can discover the manager in order to connect with it, or

alternatively the manager can initiate a connection to a device, if it is needed, without having any previous information about the device.

DI specifies a method by which Bluetooth devices share information that may be used by another device to find icons or associated software, such as for example specific drivers. All this information is also published in the SDP record.

4.1 MCAP Impementation

Our MCAP implementation is compliant with the Bluetooth Special Interest Group specification, supporting all the obligatory features and some of the optional, such as channel reconnection. The only optional feature not supported by current implementation is the clock synchronization protocol.

MCAP specification requires the enhanced retransmission and streamming modes from L2CAP. None of them was supported by BlueZ when we began our MCAP implementation, but they have recently been developed in parallel with our work by BlueZ developers (Padovan, 2009), what has allowed us to test test our MCAP+HDP implementation.

The system architecture is designed to announce to the upper layer all the events occurred during the protocol execution using callbacks. This behaviour allows the HDP layer to receive all the events in real time, not needing to do polling or other CPU-intensive operations, what renders the implementation more efficient.

MCAP implementation is multithreaded. Once the MCAP session is opened, a new thread is launched, waiting for new connections. This thread also controls all the actions which are part of the protocol, such as the connection of new data channels, disconnections, reconnections, etc. The main thread listens the sockets where new connections are received and also serves control commands. This architecture allows the system to continue working normally while MCAP is running.

4.2 HDP Impementation

The current HDP implementation simplifies the use of MCAP by the user. It also registers all the necessary information on the SDP record. It uses both, callbacks and accept functions to notify the events to the user, depending on the role of the current HDP execution (source or sink).

The implementation completely hides the MCAP control channels mechanism, providing an abstraction model based on a notification service which notifies new MCAP Communication Links (MCL) like new available remote devices. Of course, the creation of new MCL's is made possible by providing the bluetooth address of the remote devices. This action will create a new MCL that will allow applications to request new data channels.

The API provides mechanisms to start data channel connections to a previously connected device. Connections can also be received from remote devices. Depending on the HDP role of the device, it waits for a new connection calling to an *accept* function (source role) or the new connections are automaticaly notified when they are received (sink role).

Our HDP implementation also manages the creation of new data channels and controls the correct configuration of them before the connections are notified to the user. A check is made to know if the data channel uses reliable or streamming mode depending on the configuration parameters that were previously negotiated.

HDP also takes care of the correct registering in the SDP record of all the necessary information, such as the PSM where the MCAP profile is waiting for connections, the supported devices and protocols, etc.

5 EXPERIMENTAL RESULTS

The implementation described in the previous sections has been tested in order to check its correct behavior and compliance with the standards.

A first test tried to determine if the manager implementation was Continua compilant. This test was run by Andago Ingeniería personnel. The test consisted on a communication between our ISO/IEEE 11073-20601 manager and a simulated thermomether agent provided by the Continua Enabling Software Library (CESL), through a TCP connection. This test was very satisfactory because the response of the manager was completely correct including: the MDER encondig library, the manager state machine transitions and the APDUs exchanged between manager and agent.

Once the integration of the manager in Android was done, the work effort shifted to the wireless communication. In this area we developed an emulation of a thermomether agent running on Android, using Bluetooth RFCOMM communications to exchange data with our manager. We also developed an adaptation of our manager to Bluetooth RFCOMM. This way we were able to run both, the manager and the agent, in a couple of Android devices. The communication carried out correctly. Figure 7 shows the manager and an agent running on Android devices.

In the latest devolopment stage the MCAP and

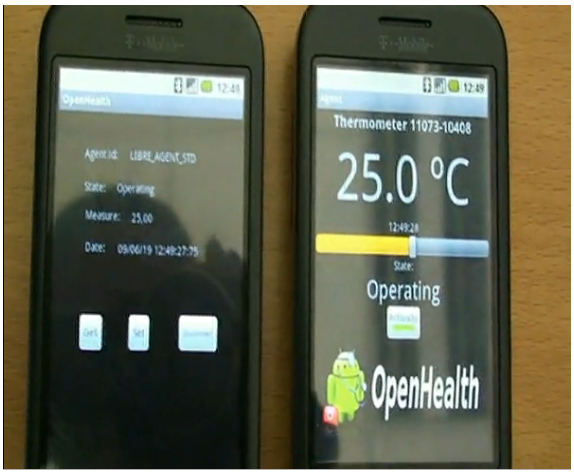

Figure 7: Communication through a RFCOMM connection. Agent (right), manager application (left).

HDP implementation were implemented. The first test of these modules intended to communicate two Linux desktop computers using multiple MCAP data channels. Then we tried the same communication model but using HDP. Finally we tested the implementation between a computer and a Nonin Onyx II 9560 Pulse oximeter (Nonin Medical Inc., 2009). These tests were all satisfactory. We ensured that the MCAP commands were correctly interpreted according to the MCL state machine transitions. The HDP implementation was also tested, comfirming that the SDP record was correctly registered and that the communication management done by HDP was also performing the correct actions. Figure 8 shows the Nonin Onyx II 9560 pulse oximeter communicating with the computer.

6 RELATED WORK

As far as we know, this is the first Java FLOSS implementation of ISO/IEEE 11073-20601 that will run on and HDP enabled Bluetooth stack. Also, this is the first FLOSS HDP/MCAP Bluetooth available implementation. In private communications, members of the Bluetooth community have told us that several companies have said to be working on Open Source implementations of either the manager or the HDP/MCAP Bluetooth stack for Linux OS, but as far as we know, no code providing both parts (manager and HDP Bluetooth support) has been publicly released, nor papers describing it have been published.

Mindtree (Mindtree, 2009) corporation announced and demonstrated on october 2009 a closed source manager running on Android, on top of its closed source HDP enhanced Bluetooth stack.

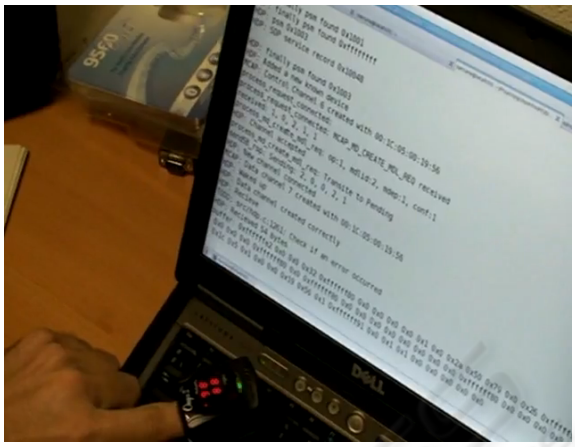

Figure 8: Communication between a Nonin Onyx II 9560 pulse oximeter and a Linux computer using our HDP/MCAP implementation.

7 CONCLUSIONS AND FUTURE WORK

The ISO/IEEE 11073-20601 Open Source manager we have implemented has been tested with the thermometer agents provided by the Continua Health Alliance CESL (Lamprey Networks, 2009) reference implementation, and with our own implementation of thermometer agents.

On the other hand, the MCAP/HDP implementation has already been tested with Nonin Onyx II 9560 pulse oximeters implementing HDP.

As work in progress we are integrating the ISO/IEEE 11073-20601 code with the HDP/MCAP BlueZ enabled stack, and testing it on Android devices from which we manage the Nonin pulse oximeters. We have begun developing a D-Bus (freedesktop.org, 2009) interface for HDP/MCAP in order to ensure an easy interoperability with Linux desktop applications using BlueZ. The D-Bus interface will facilitate the interoperability between HDP/MCAP and the ISO/IEEE 11073-20601 manager because D-Bus is accessible directly using Java through a JNI interface.

Alongside our partner Andago Ingeniería, member of the Continua Health Alliance, we are exploring both, the Continua Health Alliance Certification program and the Bluetooth qualification tests.

As of today, stack vendors have not updated stacks with HDP/MCAP profiles. No current mobile phone Bluetooth stack (including the Windows stack) implements it. Our work on HDP/MCAP, alongside the work done in parallel by BlueZ developers implementing enhanced retransmission and streamming

modes for L2CAP, could potentially have a positive impact on the industry, as ours would be the first Open Source reference implementation of ISO/IEEE 11073-20601 over HDP/MCAP bluetooth transport that could be freely used to test new devices appearing on the market.

The first medical device that implements the HDP Bluetooth profile and the ISO/IEEE 11073 data protocol is the Nonin Onyx II 9560 pulse oximeter, announced in June 2009, and only recently available in the market. This device is the one we are using for testing our implementation. The rate of appearance of new devices on the market is slow because there are no ISO/IEEE 11073 / HDP stacks available. On the other hand, developers of stacks are awaiting the appearance on the market of new devices. We expect to be contributing in part to solve this deadlock situation with our Open Source stack.

REFERENCES

Abdurakhmanov, A. G. (2008). Binary notes. http://bnotes.sourceforge.net/.

Andago Ingeniería S.L. (2009). Open health assistant. http://openhealthassistant.andago.com/inicio.

Bluetooth Special Interest Group Inc. (2001). Service discovery application profile.

Bluetooth Special Interest Group Inc. (2007). Device identification profile (di) 1.3.

Bluetooth Special Interest Group Inc. (2008a). Health device profile.

Bluetooth Special Interest Group Inc. (2008b). Multichannel adaptation protocol.

Bluez (2009). Bluez project. http://www.bluez.org/.

freedesktop.org (2009). D-bus. http://www.freedesktop.org/wiki/Software/dbus.

GSyC/Libresoft Research Group (2009). Openhealth project. http://openhealth.morfeo-project.org.

Institute of Electrical and Electronics Engineers Inc. (2004). Part 20101: Application profiles base standard.

Institute of Electrical and Electronics Engineers Inc. (2008a). Part 10404: Device specialization pulse oximeter.

Institute of Electrical and Electronics Engineers Inc. (2008b). Part 10408: Device specialization thermometer.

Institute of Electrical and Electronics Engineers Inc. (2008c). Part 20601: Application profile optimized exchange protocol.

International Telecommunication Union (2002). Itu-t recommendation x.680. http://www.itu.int.

Lamprey Networks (2009). Continua enabling software library. http://www.lampreynetworks.com/.

Mindtree (2009). Mindtree announcement. http://tinyurl.com/yha2fhq.

Nonin Medical Inc. (2009). 9560 Onyx II. http://www.nonin.com.

Padovan, G. F. (2009). Streaming and enhanced retransmission modes for bluez project. http://www.bluez.org/.