1-28-2012

The Medical Device Dongle: An Open-Source Standards-Based Platform for Interoperable Medical Device Connectivity

Philip Asare
University of Pennsylvania, asare@seas.upenn.edu

Danyang Cong
University of Pennsylvania, cdanyang@seas.upenn.edu

Santosh G. Vattam
University of Pennsylvania, vattam@seas.upenn.edu

BaekGyu Kim
University of Pennsylvania, baekgyu@seas.upenn.edu

Andrew King
University of Pennsylvania, kingand@seas.upenn.edu

See next page for additional authors

Philip Asare, Danyang Cong, Santosh G. Vattam, BaekGyu Kim, Andrew King, Oleg Sokolsky, Insup Lee, Shan Lin, and Margaret Mullen-Fortino. 2012. The medical device dongle: an open-source standards-based platform for interoperable medical device connectivity. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium (IHI '12), Miami, FL*. ACM, New York, NY, USA, 667-672. DOI: 10.1145/2110363.2110438

Conference Site: http://sites.google.com/site/web2011ihi/

The Medical Device Dongle: An Open-Source Standards-Based Platform for Interoperable Medical Device Connectivity[*]

Philip Asare
Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, United States
asare@seas.upenn.edu

Danyang Cong
Santosh G Vattam
Computer and Info. Science
University of Pennsylvania
Philadelphia, United States
{cdanyang, vattam}@seas.upenn.edu

BaekGyu Kim
Andrew King
Computer and Info. Science
University of Pennsylvania
Philadelphia, United States
{baekgyu, kingand}@seas.upenn.edu

Oleg Sokolsky
Insup Lee
Computer and Info. Science
University of Pennsylvania
Philadelphia, United States
{sokolsky, lee}@seas.upenn.edu

Shan Lin
Dept. of Computer Science
Temple University
Philadelphia, United States
shan.lin@temple.edu

Margaret Mullen-Fortino
University of Pennsylvania
Health System
Philadelphia, United States
margaret.fortino-mullen@uphs.upenn.edu

ABSTRACT

Emerging medical applications require device coordination, increasing the need to connect devices in an interoperable manner. However, many of the existing health devices in use were not originally developed for network connectivity and those devices with networking capabilities either use proprietary protocols or implementations of standard protocols that are unavailable to the end user. The first set of devices are unsuitable for device coordination applications and the second set are unsuitable for research in medical device interoperability. We propose the Medical Device Dongle (MDD), a low-cost, open-source platform that addresses both issues.

Categories and Subject Descriptors

J.3 [**Computer Applications**]: Life and Medical Sciences; D.2.12 [**Software Engineering**]: Interoperability; H.4.3 [**Information Systems Applications**]: Communications Application

General Terms

Design, Standardization

Keywords

interoperability, medical device connectivity, IEEE 11073-PHD, plug and play

[*]This research was supported in part by NSF CNS-0834524, NSF CNS-0930647, NSF CNS-1035715, and NIH 1U01EB012470-01.

Post-print version. This paper has been originally published at:
2012 ACM Health Informatics Symposium
Miami, Florida, USA
January 28–30, 2012

1. INTRODUCTION

Emerging medical applications rely on device coordination [7]. Such coordination requires that medical devices be able to connect over a network, and the process is made much simpler if the devices employ a common set of connectivity and communication protocols. Unfortunately, many of the existing medical devices in use were not originally designed for network connectivity [8]. However, a good number of these devices provide a communication port (usually RS-232 or USB) for data exchange with a single computer. Such devices can be adapted for network connectivity by providing a peripheral that connects to their output ports. The devices would then be able to share their data or be controlled over the network using a common set of protocols. This integration could then support distributed applications such as automated data collection, remote monitoring and vital sign analysis, and inter-device coordination.

Such peripherals exist [9], and even though they use a standard protocol like IEEE 11073-PHD [4], these peripherals are geared more towards personal health devices and less toward enabling the coordination applications we are interested in. Also, the implementation of the protocols are unavailable to the end-user, making such peripherals unsuitable as test beds or platforms for research in interoperability since the user has no way of modifying the protocols. The aim of the Medical Device Dongle (MDD) is two-fold: first, to enable existing devices connect and communicate using a standard protocol to allow/support development of distributed medical applications; second, to provide a platform and test bed for research in medical device interoperability. Most of this paper focuses on the first aim; however a discussion on the second aim is provided in section 7.

2. RELATED WORK

The Continua Health Alliance [1] is a consortium that provides design guidelines for developing medical equipment to meet the IEEE 11703 standards for interoperability. They provide a number of tools, guidelines, and references designed for developing devices to meet these standards; however, these resources are available to members only. Also, Continua's efforts target the industry, particularly device vendors, and more specifically telehealth applications. Our work is targeted at the research community and is aimed at providing lower cost options for investigating new coordinated medical device applications (both in the hospital and for telehealth) and exploring medical device interoperability issues.

Intel® provides an evaluation kit [6] for developing embedded devices that interact with medical devices. Its kit provides the binary code for Continua's software stack for Microsoft Windows Embedded 7 for free, as well as a trial version of the Windows Embedded operating system. This solution can get expensive since an Intel® Atom development kit costs $1,300 at minimum. The kit also targets the development of device managers only and assumes that the medical devices it will connect to are network-capable and Continua-certified. Our work, on the other hand, targets already existing medical devices which were not original developed for connectivity and hence do not follow any interoperability protocol.

The OpenHealth project [5] is aimed at provide an open implementation of the IEEE 11073 protocol for manager devices that wish to interact with Bluetooth-enabled medical devices (like those certified by Continua). It provides the Bluetooth Health Device Profile (HDP) which is required for interacting with Continua-certified devices, as well as an API for building applications (mostly for the Android platform) that interact with such devices. This work only targets building manager applications mostly for personal health applications in body area networks. Our work targets applications both in the hospital and for personal health, and our device manager is designed to be platform-agnostic and is aimed at supporting device coordination applications.

3. MDD OVERVIEW

Applications of interest require that devices connected to the patient interact with a central point called the supervisor. This supervisor usually acts as in intermediary between the medical devices and the software applications that use them in a coordinated manner. This *multiple-devices-single-manager* architecture necessitates two versions of the MDD: a manager MDD for the supervisor, and an agent MDD for each medical device. The MDD can be implemented as a physical peripheral connected to a device or as logical software components running on a device (with access to the device's network interface). The MDD is based on the IEEE 11073 protocol and is designed to support other medical devices and interoperability protocols, especially those geared towards maintaining patient safety like those used in the Medical Device Coordination Framework (MDCF) [7]. We chose 11073 because it is an IEEE standard and supports the multiple-devices-single-manager model. The network architecture is based on the older point of care (PoC) standard, while the connectivity and communication protocol is based on the personal health devices (PHD) protocol. We chose PHD for the main protocol because it has better and more recent documentation than the previous PoC standard. It is important to note that we did not implement the full standard, but only those parts we deemed sufficient for medical device connectivity and communication. In particular, we focused on the design and development of

1. The manager and agent association finite-state machines for maintaining connectivity.
2. GET, SET, and EventReport services for message exchange and command execution.
3. The Medical Device System object from the domain information model (DIM) for device description.
4. The Medical Device Encoding Rules (MDER) from the communication model for encoding messages.

We did not implement any device specializations (though we will be using the specified data formats for each device) nor did we implement any of the other encoding rules since the MDER is the only encoding rule required by the standard. We call our implementation 11073-MDD. We limited our implementation to providing connectivity and communication because these are the most well-documented parts of the protocol and we believe that this is the most basic requirement for interoperability.

4. MOTIVATION/USE CASES

We describe two scenarios that motivate our work: the first is in the hospital ICU context and the second applies more to the out-patient context.

ICU Monitoring. Electronic health records (EHR) have enabled digital recording and storage of data into a patient record; however, ensuring accuracy of the data remains a challenge. Medical devices are often purchased with a vendor-specific server that collects data from the device to make it available to the practitioner. If the EHR provider is different from the device and server provider, chances are that these two systems are using incompatible communication protocols and formats, especially if these are proprietary. The hospital IT department is then tasked with the job of bridging these two systems. The other problem is with the devices that have no network connectivity. Data from such devices are usually entered manually. This creates room for error in data entry and makes the whole data collection process inconsistent. With the MDD, all devices in a patient's room can be connected to a device manager in the room. This device manager can then interact with the patient EHR to ensure that all the data from devices are collected in the EHR over the network. This creates less administrative problems, reduces the room for errors, and ensures that data is communicated in a standard format regardless of the manufacturer of each device. It also enables other applications besides the patient's EHR to have access to the medical devices and use them as needed.

Remote Patient Monitoring for Primary and Emergency Care. Advances in treatment of chronic illnesses allow patients to have active lives, including the ability to travel. People often develop illness in locations far from their primary physician, requiring them to seek care in an emergency room. The sick patient may not be able to provide all of the necessary information or provide a comprehensive medical history. This lack of information effects the practitioners ability to ensure safe care. Medical device connectivity would provide a mechanism for the patient to be monitored and have their data transmitted in real-time to their primary care physician, enhancing the physicians ability to make appropriate treatment decisions. The MDD enables such capabilities. If medical devices in hospitals, ambulances, and clinics were equipped with the MDD, then the patient could connect to them with their smart phone or a dedicated device (running a logical version of the MDD and an application to coordinate the devices), which would in turn transmit the data to their EHR and alert the primary care physician that the patient has just visited a health facility or had been in an ambulance.

5. ARCHITECTURE

As mentioned previously, the MDD consists of an agent and a manager portion, each with a collection of software components, called MDDWare, that work together to ensure interactions between the agents, manager, and other applications in a plug-and-play manner. As shown in Figure 1, the agent MDD communicates with the medical device over standard data interfaces like USB and RS-232 using the vendor's proprietary format. The agent converts communication from the medical device into the 11073 to be sent to the device manager over standard network interfaces like Ethernet, Bluetooth, or WiFi. The agent converts any messages from the device manager back to the vendor's format to be sent back to the medical device. The manager also provides an interface that allows medical applications and other devices to indirectly interact with the medical devices through the manager.

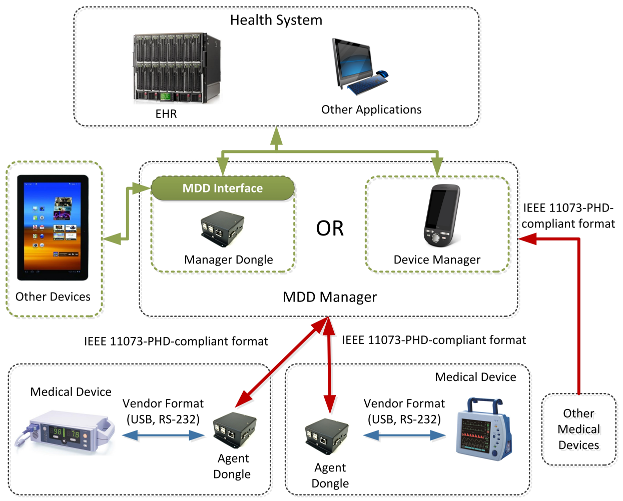

Figure 1: General Device Connectivity Architecture

5.1 Agent-side MDD

Figure 2 shows the agent-side software architecture. The MDDWare is the same for each MDD. The other components vary based on the medical device the MDD is connected to.

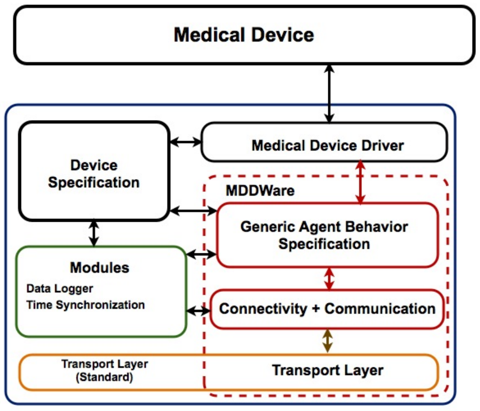

Figure 2: Agent Software Architecture

Medical Device Driver. The medical device driver acts as the intermediary between the MDDWare and the medical device. The characteristics of the device are specified in the medical device specification. The driver reads this specification in order to know which requests can be handled by the device. When data is requested, the driver parses the messages from the medical device, translates the data into the right format and passes this data on to the generic agent behavior specification (GABS) module to be sent to the manager in the right 11073 packet format. The driver also receives messages on behalf of the medical device from the GABS and translates these messages into the vendor's format to be sent to the medical device. If the driver receives a request from the GABS that the device cannot handle, it sends an error message to the GABS to be sent to the manager. A driver can be specific to a particular device or could be for a family of devices which are differentiated by their device specifications.

Generic Agent Behavior Specification. This module is responsible for implementing the 11073 protocol. Once it detects that it is connected to the medical device and the network, it initiates the association procedure with the manager. It is also responsible for going through the configuration procedure with the manager if the device is not recognized by the manager. During operation, it handles the GET and SET service requests from the manager, and generates EventReport messages on behalf of the device driver. It is responsible for encoding messages from the device driver into the right 11073 packet format and decoding messages from the manager before passing on the requests to the medical device driver.

Connectivity and Communication. This module is responsible for tracking the connectivity states of the device and alerting the GABS of state changes so the appropriate actions can be taken. It is also responsible for generating packets (when prompted by the GABS) in the MDER format as specified by the 11073-PHD standard before sending messages via the transport layer.

5.2 Manager-side MDD

Figure 3 shows the manager-side software architecture.

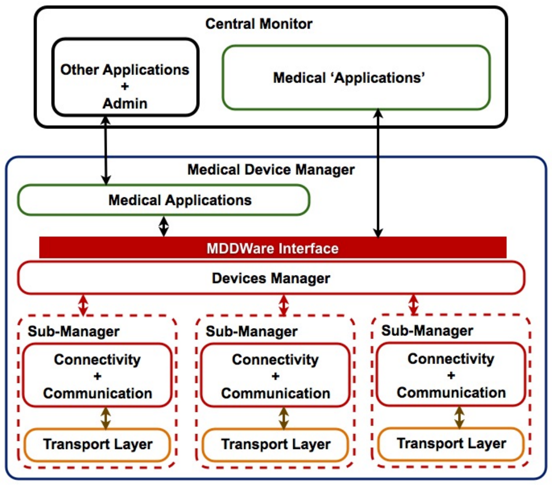

Figure 3: Manager Software Architecture

MDD Device Manager. The MDD Device Manager performs the following managerial and control tasks:

- Maintain a list of devices that are currently connected to the patient.

- Create sub-managers for each of the MDD Agents connected to the above mentioned devices.
- Maintain the sub-managers through the duration of connectivity.
- Provide an interface between connected devices and medical applications that want to interact with them.
- Destroy or remove the sub-managers once a device is disconnected.

MDDWare Interface. The MDDWare Interface sits on top of the MDD Device Manager and can be considered part of the manager. It is designed to provide a transparent interface for any application that requests the services of the manager. Applications that need to access data from a specific device connected to a patient can request this information from the MDD Device Manager using the MDDWare Interface. The MDDWare Interface and the MDDWare exist on the same MDD manager device. The services of the MDDWare interface can be requested either locally, by an application located on the MDD manager device itself, or remotely, from an application located on a different device.

Sub-manager. Each submanger is responsible for *exactly one agent*. It maintains the connection and directly exchanges packets with the agent on behalf of medical applications or other modules. The number of sub-managers is determined by the manager. A sub-manger is destroyed once the agent connected to it permanently disconnects. The use of sub-managers allows the various connected agents to be in different states and to ensure that the states of the manager and agent for each device are synchronized. This allows for easier analysis of the state machines of the managers and agents.

6. CURRENT STATUS

Most of our efforts have been aimed at coming up with the system architecture. This was necessary to make sure that our architecture was flexible, scalable, and evolvable. It is important to note that even though the 11073 protocol specifies the expected behaviors of agents and managers, it is not clear from the specification what the best way to implement these components in software is. For example, interactions are typically shown for one agent and one manager, whereas in real implementations a manager may interact with more than one agent. Hence, is therefore not clear from the specification how to synchronize the state of the manager with the potentially varied states of the different agents it may be connected to. This prompted us to come up with the sub-manager-based architecture, where the device manager is responsible for maintaining global information about connected and known devices, and the sub-managers are responsible for interactions with the agents which conforms to the behavior specified in the protocol. We therefore consider our software architecture a main contribution of this work.

We have moved past the main design stage and are at various stages of implementation of the different components. We are continually refining design details. The MDD currently consists of a TI Beagleboard (a low cost embedded platform) running Linux (Ubuntu 10.04 LTS). It has a number of USB ports (which can be converted to RS-232 and bluetooth using adapters) and an ethernet port. The MDDWare is implemented in Python. The parts of the MDDWare that have currently been implemented are:

- A full version of 11073-PHD association finite state machine (FSM) for both the manager and agent
- Basic device drivers for a pulse oximeter (Nellcor N-595) and blood pressure monitor (AND UA767-PC).
- Ethernet and Bluetooth transport layers (including a Bluetooth transport layer for Android)
- A basic version of the MDDWare interface (in Python and Android)
- A basic device description specification file

The MDD can be implemented on any other embedded platform that will support Linux and Python. Basic connectivity and communication tests have been run for our pulse oximeter agent. We have tested the behavior of the association state machines and the ability for the agent to respond to GET and SET requests. An Android application was implemented to view data from pulse oximeter and blood pressure monitor as part of these tests. This application is built on the top of manager MDDware and uses the connectivity configuration shown in Figure 1. The setup for this application is shown in Figure 4.

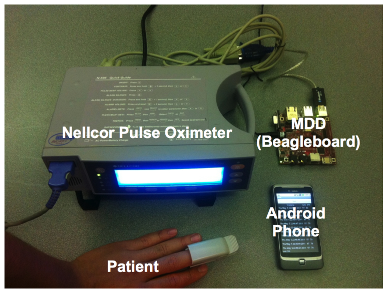

Figure 4: MDD Basic Implementation

Our Bluetooth tests were conducted in an uncontrolled environment with no latency or dropped packet issues. This is probably due to the fact that the pulse oximeter is a low data rate device (providing data at 0.5 Hz)—the blood pressure monitor is a data-on-demand device. We expect reliability issues to appear with higher data rate devices like ECGs. We also conducted Ethernet/WiFi tests on a public network with no latency of dropped packet issues.

7. WORK IN PROGRESS

We are working towards having a complete and validated implementation of the MDD. We are also working on applications that use the MDD in various device coordination scenarios. Our aim is to have a platform that is compatible with the 11073-PHD standard and a manager interface that allows for the easy development of applications that rely on device coordination.

Validation of 11073 compliance. We are not aware of any widely accepted methods or tools to validate IEEE 11073 compliance besides the Continua certifications process which is open to members only. We are looking in two directions to validate our compliance. One way is to validate using tools like Frontline's IEEE 11073 Sniffer + Analyzer [2] and NIST's ICS Generator and PDU Validation tools [3]. The Frontline 11073 Analyzer sniffs and analyzes the packets used for communication and validates if the packet conforms to the 11073-PHD standard in terms of the packet size and the packet structure. The NIST's ICS Generator generates the XML schema file that can be validated against the device specification schema as used by the MDD. The other way is to have the MDD interact with certified 11073-compliant devices.

Support for more devices. The goal of our platform is to provide as many medical devices as possible with interoperable connectivity. We are in the process of developing device drivers for a ventilator and ECG monitor. This process will help us provide guidelines on developing device drivers for the MDD so others can develop their own drivers for devices they have available to them. Since the MDDWare is the same across all MDDs, the MDD should be capable of interfacing with different medical devices so long as a proper device driver is provided.

Interfacing with other protocols. As mentioned previously, we have the MDDWare Interface layer on top of the device manager layer that provides a pluggable interface for any service or protocol that wishes to utilize the MDD service. We are, currently, working on provide support specifically for the MDCF [7]. An MDCF device can be interfaced with the MDD agent by developing a module that uses the MDDWare service on the MDD Agent as shown in Figure 2. The MDCF Manager can interface with the MDD Manager by utilizing the services of the MDDWare Interface as shown in Figure 3. Similarly, other applications can be run to using the MDD as a service, on top of it.

Support for medical applications. The MDDWare interface allows applications to interact with the MDD device manager in a transparent manner without having to know the internals of the functioning of the MDD itself. Thus applications can be written on top of MDDWare to utilize the MDDWare functionality. Specifically, we are working three applications.

The first is a time synchronization module for the MDD. Time synchronization is an essential requirement for medical device interoperability and for applications that require device coordination. Unsynchronized data can provide misleading information about the patient's state. Since each MDD is part of local network and may not have direct access to the internet to synchronize, we are developing a synchronization module based on standard synchronization methods like NTP to synchronize agent MDDs with their manager.

The second is an ICU data entry application that is aimed at providing intelligent error-free data entry for ICU nurses. It interacts with the device manager in an ICU room to auto-populate the nurse data entry form as much as possible and pushes this back to the central record. This reduces the errors associated with manual data entry and allow different checks to be included at different points. For example, the application can tell if there are medical devices in the patient room that should not be there or if any medical device is not there or unresponsive over the network.

The third is an application much like the one described in Section 4. It allows the patient's smartphone or personal device to interact with almost any medical device and have their data sent to their primary care physician. This helps the physician keep better track of the patient's health and makes it easier to involve the physician in care that the patient receives remotely.

These three applications will aid us in refining our MDDWare interface to enable even more applications to be developed that make use of the MDD.

Medical device interoperability research. One main aim of the MDD is for it to be a test bed for further exploration of medical device connectivity and interoperability. In particular, we hope that it could be a testbed for evaluating 11073 standard itself. The 11073 standard is a work in progress and is evolving. The MDDWare will provide a test bed for further exploration and testing of the standard. In addition, it will also act as a test bed for other protocols and standards that can be interfaced with 11073. This is possible because the MDD is open source with the source licensed under LGPL and the source is available to use [10], modify and also redistribute, thus facilitating any improvements or additions. Users can therefore either redesign software components and their interfaces or build modules to interact with the existing components as part of interoperability investigations. One direction we are currently looking at is ensuring patient safety while allowing interoperability. Another direction we hope to venture into is the security issues associated with enabling interoperability.

8. CONCLUSION

We described the Medical Device Dongle, its hardware and software architectures, the current status of the project, and its future directions. The MDD is designed to provide two benefits: it is a peripheral and collection of software that enables interoperable connectivity for researchers interested in medical device coordination applications; and it is a platform on which interoperability can be investigated for researchers interested in developing interoperability protocols and investigating interoperability issues for medical devices. The choice of hardware and the open-source nature of the project provides researchers with an accessible and low-cost platform to use in such endeavors. The aim is to use the feedback provided by users of the platform to improve our own work on medical device connectivity as well as to improve the MDD for those interested in using it for their research activities.

9. REFERENCES

[1] Continua Health Alliance. *http://www.continuaalliance.org.*

[2] Frontline IEEE 11073-20601 protocol analyzer. *http://www.fte.com/support/IEEE11073-download.aspx.*

[3] NIST medical device communication testing project: test tools. *http://xreg2.nist.gov/medicaldevices/testtools.html.*

[4] ISO/IEC/IEEE health informatics–personal health device communication–part 20601: Application profile–optimized exchange protocol. *ISO/IEEE 11073-20601:2010(E)*, pages 1 –208, 1 2010.

[5] GSyC/Libresoft. OpenHealth project. *http://openhealth.morfeo-project.org/.*

[6] Intel®. Evaluation kit with IEEE 11073 Continua-certified software stack for medical applications. *http://www.intel.com/p/en_US/embedded/applications /medical/evaluation-kit/overview.*

[7] A. King, S. Procter, D. Andresen, J. Hatcliff, S. Warren, W. Spees, R. Jetley, P. Jones, and S. Weininger. An open test bed for medical device integration and coordination. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 141 –151, May 2009.

[8] K. Lesh, S. Weininger, J. M. Goldman, B. Wilson, and G. Himes. Medical device interoperability-assessing the environment. In *Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, HCMDSS-MDPNP '07, pages 3–12, Washington, DC, USA, 2007. IEEE Computer Society.

[9] C.-Y. Park, J.-H. Lim, and S. Park. ISO/IEEE 11073 PHD standardization of legacy healthcare devices for home healthcare services. In *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pages 547 –548, Jan. 2011.

[10] PRECISE Center. Medical device dongle (MDD)project. *http://rtg.cis.upenn.edu/mddongle/.*