A VERSATILE MOBILE DEVICE ARCHITECTURE FOR MANAGING DATA FROM BODY AREA NETWORK MEDICAL SENSORS

P. Andreou, V. Kilintzis, A. Mourouzis, I. Chouvarda, N. Maglaveras
Lab of Medical Informatics, Medical School, Aristotle University of Thessaloniki, Greece

In this paper we present a versatile mobile device architecture for storing and presenting semantically enriched vital signs, biosignals and bioparameters from wireless Body Area Network medical sensors. The architecture focuses on reusability and scalability by defining a simple communication between the central platform module and the device specific add-on modules. Following this architecture the platform provides a repository for medical sensor measured data independently of the sensor vendor. The platform is developed for android and a device specific module add-on was also developed as an example for Bioharness Zephyr. The steps for deploying additional device specific add-on modules for other sensors are also presented.

Keywords: sensors, android, data management.

1 INTRODUCTION

The provision of healthcare services to anyone, anytime, and anywhere begins with the need of acquiring the required physiological parameters outside of a controlled environment such as a hospital. With wearable and body area medical sensors and medical software patients can constantly monitor their health status, detect deteriorations, and initiate actions for improving health.

The technological advance in the domain of wearable and body area network (BAN) biomedical sensors is significant in a month to month scale. New sensors that are more accurate, smaller, lighter and cheaper that are capable of measuring more bio parameters are introduced each year. Easy to buy and to use biomedical sensors can measure a variety of vital signs and signals. The increased volume and disparity of the collected data raises new management and processing challenges.

The proposed solutions regarding patient health monitoring are various. Wagner et al. propose an Android based BAN that handles data acquisition and forwarding. Lifeware focuses on data collection from body area sensors to support a health care application. Moron et al. propose management and monitoring of chronic diseases through a smart-phone based tele-care system collecting data from Bluetooth-enabled wearable medical sensors. A very interesting approach is MyHealthAssistant is an event-based middleware running on an Android smart phone to monitor a user's daily activities. Also a more complete approach is presented by Philip et Al. where through the use of RESTful web services the sensor measurements are stored to the cloud and shared with applications.

The common practice regarding management and exploitation of medical sensor’s measurements through custom applications, as illustrated in the above examples, is either to use the vendor’s application to view and –if possible- export the data in order to use them or, if the vendor provides an SDK, develop the communication management module and incorporate it in the custom app. In this work we aim to overcome the obstacle of having to rebuild the custom application in order to implement a new vendor specific communication protocol through the introduction of device specific module add-ons. A central platform handles the storing and presentation of the acquired data while different android applications can be built for different sensor devices in order to handle communication with them and acquire the measured physiological parameters.
The structure of this paper is as follows. In Section 2 we present the overall design of the proposed architecture identifying the different modules and their interaction. Furthermore, we also present the platform specific design decisions and the schema of the main storage database. In the Results section we present screen-shots of the implemented platform along with definition of requirements for a new device specific add-on module. Finally, in Section 4 conclusions, open issues and proposed future work are discussed.

2 METHODS

2.1 Architecture

The proposed mobile device based architecture focuses around modularity scalability and user-friendliness. A depiction of the extensible platform architecture along with its device add-ons is available in Figure 1. Three distinct parts are discernible from the overall architecture, the Biomedical body area network sensors, the module services related with each separate device and the generic platform, which is acting as an orchestrator of the whole operation.

The Bluetooth biomedical BAN sensors monitor the patient’s vital signs in an unobtrusive manner. Bluetooth pairing between the sensor and the android mobile device is a prerequisite for the sensor management by the generic platform.

The device add-ons are attachable platform modules corresponding to each type of biomedical sensor and run as background services in order to provide seamless data transfer to the platform. These are not offered as standalone solutions but accessed and executed only through the generic platform.

The platform has several roles; it acts as the sensor specific add-on manager and also handles data transfer, storage and presentation of the acquired information. As soon as the Bluetooth sensors are paired with the mobile device the add-on module services are available via the platform on a first-come-first-serve (FIFO) basis allowing their simultaneous management and execution upon user’s request. The platform collects the date from the add-on module services and stores them locally.

The platform’s graphical user interface (GUI) design is pretty simple and straightforward including a two set tabbed pane comprised of a list of devices for access and management and a data representation tab for providing feedback to the user.

A thorough analysis is presented, concerning the interaction between the different modules and the system platform and how these coordinate to provide feedback to the user.
Using android OS’s Bluetooth discovery functionality, the mobile device identifies Bluetooth sensors that are paired. All available paired sensors are listed to the GUI of the application platform for further selection and management. For each Bluetooth device that there is a known add-on module compatible with the platform, an entry in a local look-up table exists. The look-up table has the device’s name and the corresponding add-on module’s identifier. All paired devices that are on the look-up table are available for management via the platform as long as their corresponding add-on module is installed. If the device is on the table but the add-on module is not installed the user is redirected to the device add-on page on Google Play in order to download the specific add-on.

Regarding the add-on modules, each device has a particular application that can be developed by an independent developer and can attach to the application platform. The sole purpose of the add-on is to ensure the seamless transfer of Bluetooth data to the platform for later processing. Each module functions as a separate background service allowing the multiple of them to run simultaneously on demand.

A close observation of the architecture diagram shows that the different device modules are accessible through the platform’s user interface, which allows them to start or stop their seamless data transfer. As shown in the architecture, a content provider interface has been implemented inside the platform. This internal module defines the common communication framework, upon which, the whole communication between the add-on modules and the platform is performed. The content provider utilises the database adapter for accessing and manipulating the platform based backend database content. The communication between the add-on modules and the content provider is based on specific URIs. Custom URIs are used to allow the communication between the add-on module and the tables of the backend database that data are stored.

The platform’s cloud synchronization module utilizes Google Play services client to allow automatic synchronization of the locally stored data to the cloud. This is performed if the user has data synchronization option enabled.
The GUI platform module handles the communication with the user. Apart from the management of the device specific add-on modules the user can view the measured data in the form of plot graphs for each particular biomedical vital sign that is stored in the database.

Overall, the modular architectural design ensures increased performance during simultaneous execution of different modules without compromising scalability. Also the GUI module that manages add-on modules and presents the measurements provides a user friendly interaction with the platform.

### 2.2 Implementation

Current implementation has utilised the open-source Android SDK platform v.23 in combination with the SQLite database of the Android OS.

Implemented as an Eclipse plugin the android platform’s software development kit is a versatile, open-source and extensible set of tools and libraries that is suitable for the purposes of this particular project. Moreover, Android OS currently has the highest usage share among mobile operating systems. Current API level chosen is 16, since according to official Android usage statistics it is the most highly used platform version (Jelly Bean) and one of the latest ones.

SQLite database RDBMS was preferred for our development purposes, since data storage had to be delivered in a structured and highly configurable and accessible manner that could apply to all platform modules. Consequently, the platform includes a single database (“Platform.db”) comprised of three different tables, Devices, Measures and Data, which are universally accessed by every platform add-on. The tables have a predefined homogenous structure, allowing flexible storage of data and meta-data information about the various biomedical sensors and measurements. An overview of the database schema that is globally defined for all modules depicted in table 1.

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
</tr>
</thead>
</table>
| Data | `ID` – autoincrement, primary key.  
`timestampStart` - in ms  
`timestampEnd` – in ms  
`fs` - sampling rate (Hertz)  
`measureID` – measurement id from Measures table  
`deviceID` - device id from Devices table  
`values` - comma separated N values OR single value |
| Measures | `measureID` - primary key, auto increment  
`Measure` -measurement label |
| Devices | `measureID` – measurement id from Measures table  
`deviceID` – identifier of sensor device  
`deviceName` – device label |

*Table 1. Overview of the database schema*

### Communication and Interaction

As far as the scope of platform communication with each separate device module is concerned, the GUI of the platform manages the initial start-up of a device specific service module. However, a unified interface is required that would allow each application to be attached with the platform’s backend. The communication of the add-on modules with the platform’s database is achieved via the content provider interface module. The content provider allows cross-app utilization of the database tables, which otherwise would have been inaccessible. Specifically, a Content Provider provides encapsulated data to applications, thus allowing the sharing of data between applications. Device specific applications communicate with the platform’s backend through a method named `getContentResolver()` which enables access to the database model through three different URIs one for each table of the backend database.
Finally, data representation takes place at the graphical user interface where a set of radio buttons representing different biomedical measures are available for selection and retrieval from the database in the form of graphs. The graphs may represent data stored in the database from a previous acquisition time or they can present currently acquired measurements.

Cloud Synchronization

Cloud synchronization module's functionality is to act as an external storage of the database for backing up device specific data and allow the use of them from cloud based applications. As a result, whenever internet connection is enabled and the user has the synchronization option enabled, database is synchronizing its data to the cloud. The module uses Google Play Services (Google Cloud Save) to transmit and store the data to the users Google account. The cloud synchronization module also handles the update of the look-up table that defines the link between the biosensors and the corresponding add-on modules via Google Cloud Messaging service.

3 RESULTS

3.1 Main Platform functionality

A step by step example of the use of the platform is depicted in Figure 2. In this example an add-on module was specifically implemented for Zephyr Bioharness sensor in order to acquire heart rate, respiration rate, peak acceleration, posture, temperature and also 1-lead ECG vital signs.
Upon starting the platform application the list of sensors that have module add-ons implemented for the platform are shown (Figure 2.a). For those having add-on modules installed the functionality to start the specific service is given via a simple check box tick (Figure 2.b). Whenever the user wants to view stored data, he/she selects the Reports tab and the appropriate device. The list of corresponding measurements is then presented (Figure 2.c) and then the selected measurement is presented in a graph (Figure 2.d).

3.2 Adding device specific module add-ons

As mentioned above in this work we present the complete architecture along with an implemented example of a device specific add-on module for Zephyr BioHarness physiological monitor. The proposed architecture offers the possibility to attach other biomedical sensors by developing the independent add-on modules.

Each device specific module is a standalone android application. In this application there must be two distinct parts. The first refers to the connection and acquisition of data from the biomedical sensor and the second should handle the communication with the platform via the content provider interface module in order to store the acquired data.

Regarding the communication with the biomedical sensor the developer must refer to the documentation of the specific sensor. About the communication with the platform the developer must implement a simple interaction module. This platform related module must call when it needs to store data to the platform by calling the following three methods.

The `insertToDB()` is used to store a measurement value into the platform, the `insertLookupMeasure()` is used to define a type of measurement and `insertLookupDevice()` to define the specific biosensor.

The actual code of the methods can be copied by the BioHarness module add-on example.
4. DISCUSSION AND FUTURE WORK

In this work we present a versatile mobile device platform developed for android based on a modular architecture that can be used to collect and present data from various biomedical body area network sensors. The architecture is scalable through the easy addition of sensor specific add-on modules that can be developed independently of the main platform and dynamically enable the use of the measurements acquired by the platform. An example module add-on was developed for Zephyr BioHarness physiological monitor that can be started from the main platform and present in graphical form the measurements taken by the specific sensor.

The architecture could be extended either by adding to the platform interface for sending the data either to 3rd party local applications or to a cloud storage service where the acquired data could be analyzed and exploited by more complex applications. In this manner, the fragmentation of data in separate sensor-specific cloud storage services could be avoided. In addition, cloud storage service could be part of an integrated Personal Health Record or Personalised Health Services which integrates both clinical and self-recorded data.

Besides, integration with a cloud service, next steps of this work include the wider implementation with a series of widely adopted sensors, elaboration of notifications alerts and intelligence and a usability testing for the adoption of this platform by users, both patients, and developers.

References


