2012 Ninth International Conference on Wearable and Implantable Body Sensor Networks

Health-Dev: Model Based Development of Pervasive Health Monitoring Systems

Ayan Banerjee, Sunit Verma, Priyanka Bagade and Sandeep K.S. Gupta
School of Computing Information Decision Systems Engineering
Arizona State University, Tempe, Arizona
Email: {abanerj3, sverma14, pbagade and sandeep.gupta}@asu.edu

***Abstract*—Implementing requirements verified body worn medical sensors and smart phones, acting as base stations, in Body Sensor Networks (BSNs), is of extreme importance for development of reliable pervasive health monitoring systems (PHMS). Models of BSNs have been used to analyze designs with respect to requirements such as energy consumption, lifetime, and network reliability under dynamic context changes due to user mobility. This paper proposes *Health-Dev* that takes a high level specification of requirements verified BSN design and automatically generates both the sensor and smart phone code. Case studies related to energy efficiency and mobility aware network reliability show whether the resulting implementation satisfies the requirements set forth in the design phase.**

***Index Terms*—pervasive health monitoring system, automatic code generation, AADL, smart phone, body sensor network**

I. Introduction

Pervasive health monitoring systems (PHMS) consist of miniature sensors or medical devices deployed on body forming a body sensor network (BSN) that sense and process physiological signals and transfer information to a Internet data server via wireless link through a smart phone base station [1]. Operation of medical devices in PHMS has to be guaranteed to meet safety, lifetime and reliability requirements. In this regard, model based development of PHMS software has received considerable focus [2]–[4]. Despite these efforts there has been several recent cases of failures of life saving medical devices such as drug delivery systems, as reported by Food and Drug Administration (FDA) Maude database (http://www.accessdata.fda.gov/). These failures can be attributed to the errors in converting the models to implementations. This paper considers automated software development from BSN models to reduce the potential of an implementation error.

BSN applications in pervasive health-care are characterized by dynamic context changes induced by the mobility of the user. These context changes can cause the BSN system properties such as the quality of the wireless link and the available energy from the scavenging sources, to change and in some cases adversely affect the system operation. In this regard, a model based verification approach is employed for designing a PHMS [2]. This approach uses high level architectural models of BSNs for different contexts and performs mathematical analysis on the model parameters to verify the requirements (Figure 1). However, translating these verified models into actual code requires manual software development for the sensors and the smart phone. The errors in manual implementation may lead to violation of requirements by the

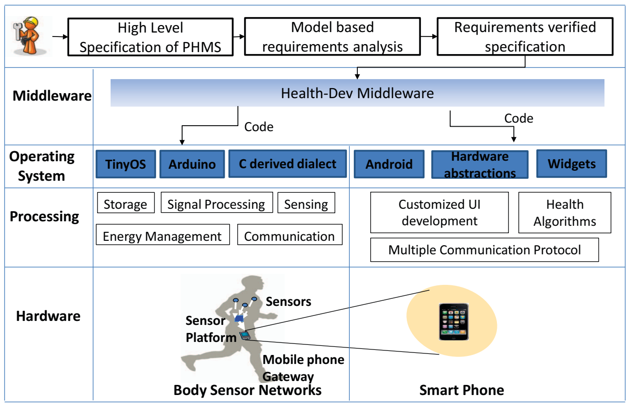


Fig. 1. **The *Health-Dev* abstracts software and hardware details of different PHMS components to a developer.**

implemented BSN. In this regard, this paper proposes *Health-Dev*, a framework for model based development of PHMS, which takes high level architectural models of BSNs and automatically generates downloadable code for sensors and the smart phones. Case studies further, show that the implemented BSN meets the requirements considered in the modeling phase.

Automated programming of sensor platforms and smart phones is a challenging task due to diversities in the hardware components as well as the software stack (Figure 1). They are typically application-specific and their operations are limited to basic sensing, simple processing, and single-hop wireless communication to the smart phone. Hence, there is an immense diversity in the hardware architectures for these sensors each having different types of sensors, micro-controllers, and radios. Further, the operating system of these platforms maybe event driven e.g. TinyOS or Linux based e.g. iMote2 and consequently the programming abstractions are also diverse for these platforms.

On the other hand, the smart phones have a totally different software development paradigm. Typically, plug-ins containing hardware abstractions of a smart phone are available in high level languages such as C or Java. The hardware abstractions are uniform across different platforms, however, the programming languages vary a lot in syntax and semantics. *Health-Dev* abstracts these diversities by using a comprehensive code repository and an efficient model parser, which maps the high level model components to OS and phone specific code.

At the front end, *Health-Dev* uses Architecture Analysis and Design Language (AADL), a high level architecture specification language, to specify the BSN [2], [4]. *Health-Dev* includes constructs to specify the number of sensors in the BSN, the sensed signals, the signal processing algorithms, the communication protocols and the routing algorithms. Further, it allows specification of the smart phone User Interface (UI), which supports display of the sensed data, sending control commands to the sensors such as `start` and `stop` sensing and change the sampling rate, and execution of the common signal processing algorithms. The AADL model is then parsed to form an XML based meta model that is used to generate readily downloadable sensor and smart phone side code. The *Health-Dev* operation is demonstrated using two case studies: 1) sensor radio duty cycling for energy efficiency, and 2) dynamic transmission power control to support reliable radio communication, under user mobility. Further, the quality of the code generated by *Health-Dev* is evaluated against the BSNBench benchmark [5], with respect to different metrics such as the RAM requirements, the code size, and the energy consumption. *Health-Dev* is written as an Open Source AADL Tool Environment (OSATE) plug-in and hence can be readily used on verified BSN models [2].

The rest of the paper is organized as follows - Section II differentiates *Health-Dev* from existing development tools; Section III discusses the architecture of *Health-Dev*; Section IV discusses its implementation; Section V evaluates and validates *Health-Dev*; and Section VI concludes the paper.

II. Related Work

Model based development of sensor code has been a major focus of research. However, most of the work have concentrated on either developing APIs or TinyOS specific sensor code generation. There exists several work on API development for TinyOS to support signal processing, intelligent storage, and energy efficient communication [6]–[8]. However, to use these APIs the user has to write TinyOS code for the sensors, introducing possibility of errors. Further, the authors in [7], [8] do not consider the interfacing of the sensors with smart phone. The mobile middleware [6] provides a basic API for the smart phone to control the sensors. However, this again requires user effort to write the code.

Researchers have also focused on code generation for sensors from a high level specification such as Simulink [8], or graphical specifications [9]–[11]. The framework proposed in [8] comes closest to *Health-Dev* since they generate code for TinyOS based sensors and also support algorithm execution sequences. However, the supported algorithms are not related to physiological signal processing (required for a PHMS). RaPTEX [9], Viptos [10], and TOSDev [11] consider the sensor nodes as only a sensing and communication device and do not allow physiological signal processing. Hence, a framework that can support intuitive high level specification for both sensors and smart phones with OS and hardware independent code generation, along with physiological signal processing is missing in the literature. *Health-Dev* fills this gap.

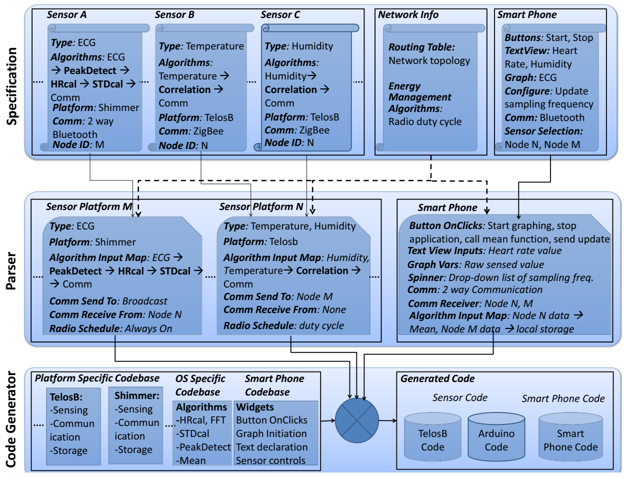

Specification
Sensor A
Type: ECG
Algorithms: ECG → PeakDetect → HRcal → STDcal → Comm
Platform: Shimmer
Comm: 2 way Bluetooth
Node ID: M
Sensor B
Type: Temperature
Algorithms: Temperature → Correlation → Comm
Platform: TelosB
Comm: ZigBee
Node ID: N
Sensor C
Type: Humidity
Algorithms: Humidity→ Correlation → Comm
Platform: TelosB
Comm: ZigBee
Node ID: N
Network Info
Routing Table: Network topology
Energy Management Algorithms: Radio duty cycle
Smart Phone
Buttons: Start, Stop
TextView: Heart Rate, Humidity
Graph: ECG
Configure: Update sampling frequency
Comm: Bluetooth
Sensor Selection: Node N, Node M
Parser
Sensor Platform M
Type: ECG
Platform: Shimmer
Algorithm Input Map: ECG → PeakDetect → HRcal → STDcal → → Comm
Comm Send To: Broadcast
Comm Receive From: Node N
Radio Schedule: Always On
Sensor Platform N
Type: Temperature, Humidity
Platform: Telosb
Algorithm Input Map: Humidity, Temperature→ Correlation → Comm
Comm Send To: Node M
Comm Receive From: None
Radio Schedule: duty cycle
Smart Phone
Button OnClicks: Start graphing, stop application, call mean function, send update
Text View Inputs: Heart rate value
Graph Vars: Raw sensed value
Spinner: Drop-down list of sampling freq.
Comm: 2 way Communication
Comm Receiver: Node N, M
Algorithm Input Map: Node N data → Mean, Node M data → local storage
Code Generator
Platform Specific Codebase
TelosB: -Sensing -Communication -Storage
Shimmer: -Sensing -Communication -Storage
OS Specific Codebase
Algorithms -HRcal, FFT -STDcal -PeakDetect -Mean
Smart Phone Codebase
Widgets Button OnClicks Graph Initiation Text declaration Sensor controls
Generated Code
Sensor Code
Smart Phone Code
TelosB Code
Arduino Code
Smart Phone Code

Fig. 2. ***Health-Dev* architecture**

III. *Health-Dev* Architecture

Health-Dev consists of three modules: 1) specification, 2) parser, and 3) code generation (Figure 2).

A. Specification Module

At the front end it provides the user with an interface to provide a high level specification of the BSN. In the specification module, the user provides three types of information:

Sensor specification: In *Health-Dev*, the user specifies the computation, and communication requirements for each type of sensed signals. Multiple sensors can be implemented in the same sensor platform depending upon the hardware capabilities of the platform. For each sensed signal a separate specification module is maintained, which has two ports: i) input data, denoting the raw sensed data and 2) output data, denoting the data that is to be reported via the wireless network. In this module, three types of information are maintained:

1) Sensor properties: This class of information includes the type of sensor, e.g., temperature or humidity or electrocardiogram (ECG) signals, the sampling frequency of the signals, sensitivity, and the platform type. *Health-Dev* supports most commercially available sensor platforms such as TelosB (xbow.com), Shimmer (shimmer-research.info), BSN v3 (http://vip.doc.ic.ac.uk/bsn/a1892.html) (TinyOS based), iMote2 (bullseye.xbow.com) (both TinyOS and Linux based), and Ardruino (http://www.arduino.cc/), (Linux based OS).

2) Sensor subcomponents: Each sensor has two classes of subcomponents related to the computation performed on the sensed value and communication of data (Figure 3). For computation in a sensor, *Health-Dev* provides *algorithm subcomponents*. It has an input port and an output port. This subcomponent also has a property to specify the name of the algorithm. A *codebase* of algorithms are maintained in the code generator module. If the algorithm name matches any one of the available algorithms then the code from the codebase is used in the implementation. Otherwise, the user has to implement the algorithm in the form of a function in the

TABLE I
SPECIFIABLE PROPERTIES IN *Health-Dev*

Sensing	Computation	Communication	Phone
1. Node id	1. Physiological processing (BSNBench, heart rate calculator, ECG models)	1. Bluetooth or ZigBee	1. Buttons, text views
2. Platform		2. destination & source address	2. graphs
3. Sensor type		3. packet contents	3. execution sequences
4. Sampling frequency	2. Execution sequence	4. radio duty cycle	4. Command to sensors

platform specific programming language and include it in the codebase. The communication subcomponent has properties to specify the communication protocol, which can be either Bluetooth or ZigBee, multi-hop or single hop communication, and the frequency of data packet transmission. This subcomponent also has four ports pertaining to the *source id* (from which a packet is received), *received data*, *destination id* (to which data has to be sent), and *send data*. Further, there are data subcomponents in the sensor to specify its *node id*, and its one hop neighbor node ids. Sensors components with the same node id are implemented in the same platform.

3) Sensor connections: To facilitate specification of an algorithm (as an execution sequence) and type of information transfered to the network, *Health-Dev* provides two types of connections. Using the algorithm (lines 7 to 10 in sensor side code in Figure 3) the user can specify the inputs of each algorithm, which can be either the raw signal or the output of another algorithm. For example, the raw ECG samples from the sensor can be passed through a peak detector, heart rate calculator (HRCal), and heart standard deviation calculator in sequence (Figure 2). The output can be transmitted to the smart phone using the communication subcomponent. The communication connections can then be used to specify the transmission and reception parameters (lines 11 and 12 in Figure 3). Multiple inputs to the algorithm can be specified by connections from multiple sensor outputs or from other algorithm outputs to the same input port of the given algorithm (lines 7 and 8 in Figure 3).

Network specification: The network topology and communication energy management schemes are specified in this subcomponent. Routing information, such as data path through the network, can be included in a routing table, a separate file, and the filename can be specified in the routing info field. If a routing table file is provided then the user does not need to specify the destination and source id. The parser will extract those information from the routing table. If none is specified then by default a network wide broadcast scheme is employed. For communication energy management, three modes of radio operation are supported - always on, only on during transmissions, or duty cycled.

Smart phone specification: *Health-Dev* allows the user to specify the UI of the smart phone base station, the sensor nodes in the BSN that can communicate with the base station, and the algorithms to be run in the base station on the received data. The user can specify buttons, text views, graphs in the smart phone subcomponent. In the button *OnClick* functions *Health-Dev* allows starting and stopping graphs and running basic algorithms such as computation of mean and standard deviation of received data. Text view and graph view inputs are specified using connections in the smart phone subcomponents. Further, the button *OnClicks* can also be used to transfer control information to specific sensors. The control information includes start or stop sensing, and change sampling frequency. Currently in *Health-Dev* only Android based smart phones are considered, which communicate with the sensors using the Bluetooth communication protocol. Table I summarizes the different sensor and smart phone properties that the user can specify.

B. Parser Module

The parser module takes the specification of a BSN as input and generates meta models, which are used in the code generation module for actual implementation.

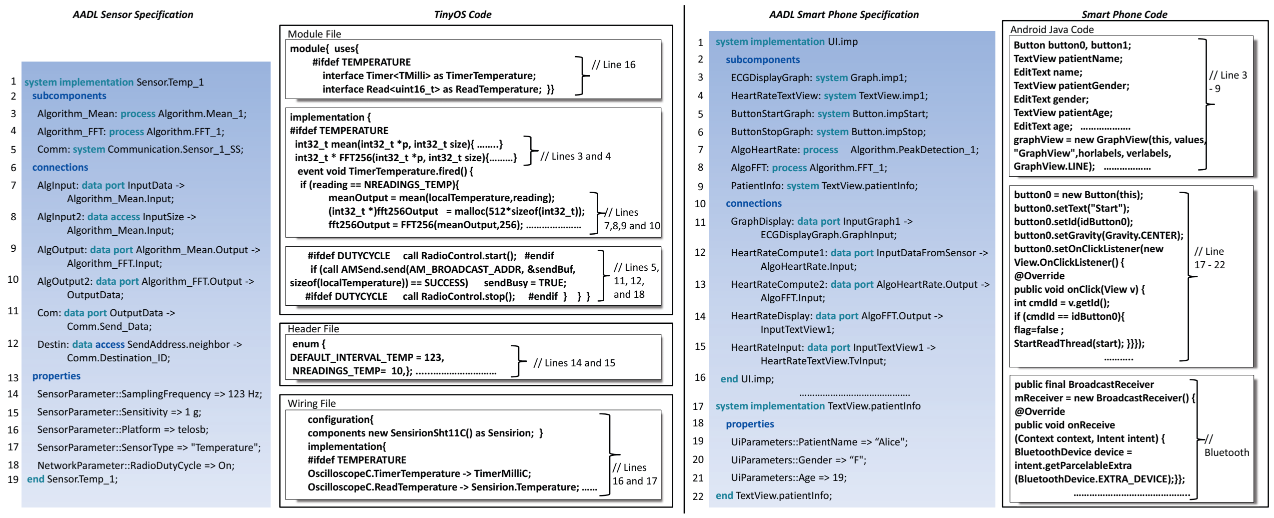
Sensor parser: The main function of this parser is to convert the platform independent specification of the sensors to a platform specific one. The parser extracts the node id and sensor platform information from each sensor specification and groups the sensors with same node id to the same sensor platform and creates a meta model for that platform or "mote". The parser then parses the sensor types and gives platform specific names to the sensors. For each algorithm the parser extracts the appropriate filename from the platform specific codebase and includes it in the meta model. It then parses the connections in the specification and for each algorithm extracts the input map. For example, the input to the correlation function in sensor platform N in Figure 2, are the temperature and humidity readings. They are specified in two different sensor components having the same node id. For each communication specification the parser extracts the destination node id, data to be sent, source node id, data received, and routing information, and includes them in the meta model.

Smart phone parser: The smart phone parser in *Health-Dev* is specifically for Android development and is primarily concerned with setting up the UI. The UI in Android is an XML specification with specific keywords defined in the Android Java widget. The smart phone parser reads the UI specification and converts it to the corresponding XML specification.

C. Code Generation Module

The code generation module takes the parser output and uses codebase to generate code for sensor platforms and smart phone. There are three types of codebases:

Sensor platform specific codebase: The sensor platform program generally follows an event driven paradigm, where there are event handler functions for sensing and communication. In these event handlers, computation tasks can be posted for execution after the processing of the events. This is especially true for TinyOS, which is the most popular OS used in sensor platforms. Further, the OS has several platform independent hardware abstraction modules that can be used to handle sensing and communication hardware. These abstractions are then wired to appropriate hardware units to



AADL Sensor Specification

```
1  system implementation Sensor.Temp_1
2    subcomponents
3      Algorithm_Mean: process Algorithm.Mean_1;
4      Algorithm_FFT: process Algorithm.FFT_1;
5      Comm: system Communication.Sensor_1_SS;
6    connections
7      AlgInput: data port InputData ->
              Algorithm_Mean.Input;
8      AlgInput2: data access InputSize ->
              Algorithm_Mean.Input;
9      AlgOutput: data port Algorithm_Mean.Output ->
              Algorithm_FFT.Input;
10     AlgOutput2: data port Algorithm_FFT.Output ->
              OutputData;
11     Com: data port OutputData ->
              Comm.Send_Data;
12     Destin: data access SendAddress.neighbor ->
              Comm.Destination_ID;
13   properties
14     SensorParameter::SamplingFrequency => 123 Hz;
15     SensorParameter::Sensitivity => 1 g;
16     SensorParameter::Platform => telosb;
17     SensorParameter::SensorType => "Temperature";
18     NetworkParameter::RadioDutyCycle => On;
19  end Sensor.Temp_1;
```

TinyOS Code

Module File

```
module{  uses{
      #ifdef TEMPERATURE                                        // Line 16
         interface Timer<TMilli> as TimerTemperature;
         interface Read<uint16_t> as ReadTemperature;  }}
```

```
implementation {
#ifdef TEMPERATURE
 int32_t mean(int32_t *p, int32_t size){ ........}             // Lines 3 and 4
 int32_t * FFT256(int32_t *p, int32_t size){.........}
  event void TimerTemperature.fired() {
   if (reading == NREADINGS_TEMP){
          meanOutput = mean(localTemperature,reading);          // Lines
          (int32_t *)fft256Output   = malloc(512*sizeof(int32_t));  7,8,9 and 10
          fft256Output = FFT256(meanOutput,256); ...................
```

```
     #ifdef DUTYCYCLE     call RadioControl.start();   #endif      // Lines 5,
      if (call AMSend.send(AM_BROADCAST_ADDR, &sendBuf,             11, 12,
sizeof(localTemperature)) == SUCCESS)     sendBusy = TRUE;          and 18
    #ifdef DUTYCYCLE     call RadioControl.stop();    #endif  }   } }
```

Header File

```
 enum {
DEFAULT_INTERVAL_TEMP = 123,                                    // Lines 14 and 15
 NREADINGS_TEMP=  10,}; ..............................
```

Wiring File

```
    configuration{
    components new SensirionSht11C() as Sensirion;  }
    implementation{
    #ifdef TEMPERATURE                                          // Lines
    OscilloscopeC.TimerTemperature -> TimerMilliC;               16 and 17
    OscilloscopeC.ReadTemperature -> Sensirion.Temperature; ......
```

AADL Smart Phone Specification

```
1  system implementation UI.imp
2    subcomponents
3      ECGDisplayGraph: system Graph.imp1;
4      HeartRateTextView: system TextView.imp1;
5      ButtonStartGraph: system Button.impStart;
6      ButtonStopGraph: system Button.impStop;
7      AlgoHeartRate: process    Algorithm.PeakDetection_1;
8      AlgoFFT: process Algorithm.FFT_1;
9      PatientInfo: system TextView.patientInfo;
10   connections
11     GraphDisplay: data port InputGraph1 ->
              ECGDisplayGraph.GraphInput;
12     HeartRateCompute1: data port InputDataFromSensor ->
              AlgoHeartRate.Input;
13     HeartRateCompute2: data port AlgoHeartRate.Output ->
              AlgoFFT.Input;
14     HeartRateDisplay: data port AlgoFFT.Output ->
              InputTextView1;
15     HeartRateInput: data port InputTextView1 ->
              HeartRateTextView.TvInput;
16   end UI.imp;
                    ........................................
17 system implementation TextView.patientInfo
18   properties
19     UiParameters::PatientName => “Alice";
20     UiParameters::Gender => “F";
21     UiParameters::Age => 19;
22 end TextView.patientInfo;
```

Smart Phone Code

Android Java Code

```
Button button0, button1;
TextView patientName;
EditText name;                                   // Line 3
TextView patientGender;                          - 9
EditText gender;
TextView patientAge;
EditText age;   ...................
graphView = new GraphView(this, values,
"GraphView",horlabels, verlabels,
GraphView.LINE);   .................
```

```
button0 = new Button(this);
button0.setText("Start");
button0.setId(idButton0);
button0.setGravity(Gravity.CENTER);
button0.setOnClickListener(new                   // Line
View.OnClickListener() {                         17 - 22
@Override
public void onClick(View v) {
int cmdId = v.getId();
if (cmdId == idButton0){
flag=false ;
StartReadThread(start); }}});
                   ...........
```

```
public final BroadcastReceiver
mReceiver = new BroadcastReceiver() {
@Override
public void onReceive
(Context context, Intent intent) {               //
BluetoothDevice device =                         Bluetooth
intent.getParcelableExtra
(BluetoothDevice.EXTRA_DEVICE);}};
              ..........................................
```

Fig. 3. **AADL specification of a sensor and smart phone and their TinyOS and Java code**

generate executable code. In the codebase, for each platform, a basic module file for handling all available type of sensing and communication hardware and their event handlers are stored. The different types of sensing and communication protocols are differentiated using preprocessing directives. Similarly a wiring file is generated with all possible hardware modules differentiated by pre-processing directives.

OS specific codebase: A collection of algorithms related to physiological signal processing and health data statistics analysis are kept in this codebase. The algorithms include the BSNBench suite [5], a benchmark specifically designed for BSNs consisting of FFT, peak detection, and statistics operation. Further, it includes physiological signal specific algorithms such as heart rate calculator from ECG signal or photo-plethysmogram (PPG) pulse width calculator, signal normalization, and correlation computation.

Smart phone codebase: This codebase consists of parameterized declaration and call back codes written in Java for Buttons, Text View, and Graph Views. The Java widgets provided by Android are also kept in the smart phone codebase.

IV. *Health-Dev* IMPLEMENTATION

Health-Dev is implemented using the AADL as the specification language and the OSATE to develop the parser and the code generator modules. Figure 3 shows the AADL specification of a single sensor and the smart phone along with their respective implementations in TinyOS and Android. Note that *Health-Dev* also supports code generation for non-TinyOS based sensors such as Arduino, but for the simplicity of discussion we use the TinyOS example.

A. Generation of the Sensor Code

Each sensor can be implemented using the *system* construct in AADL. The sensor system has two *ports*: `InputData`, which is the raw sensed signal and `OutputData`, which is the data to be transmitted to the base station. The algorithms and the communication radio are subcomponents of the sensor system. The algorithm is implemented as a *process* and has two *port*s corresponding to the input and the output. An execution sequence in the sensor can be specified using *connection*s between algorithms as shown in the Figure 3, lines 7 through 9. The communication radio is implemented as a *system* with four *ports*: `SendData`, the data to be transmitted, `ReceiveData`, data received, `DestinAddress`, the destination address, and `SourceAddress`, node from which data was received. These ports can be populated using the *connection*s construct (Figure 3 AADL sensor specification).

Given these AADL specifications the OS and platform specific codes are generated using **Algorithm 1**. The algorithm first searches sensor components in the AADL file and checks the node ids to group sensor components to sensor platforms. For each sensor platform a new copy of the basic files with all possible sensors and communication protocols differentiated using pre-processing directives were generated from the code repository. The code generator searches for the sensor type and sets the corresponding directive to “true”.

In the next step, the code generator searches for algorithm subcomponents in each sensor component. Once an algorithm subcomponent is found a codebase query is made. The corresponding algorithm definition is then inserted in the implementation section of the TinyOS code. The algorithm inputs and execution sequence are determined by parsing the connections in the sensor component. In this step the code generator searches through all the connections and groups them into classes based on their destination names. The inputs to a given algorithm A will be the sources of the connections whose destinations are all A’s input. Depending upon the type of input, the function call is written accordingly in the TinyOS code (Figure 3). An important problem with regard to function calls is the correct sequencing of algorithms. This is done by the code generator during parsing the inputs of the algorithms. While parsing the inputs of any algorithm A, the code generator checks whether any input matches the

Algorithm 1 Sensor Code Generator

```
Use OSATE component switch to parse sensor components
for each sensor component do
    Group sensors with same NodeIds on the same platform
    Query the platform specific codebase for base code
    Parse sensor type and set corresponding directive to "true"
    Parse communication protocol and set directive to "true"
    Use the OSATE subcomponent switch to parse algorithms
    for each algorithm in the sensor component do
        Query the OS specific codebase for $algorithm$ declaration
        Declare the algorithm at the beginning of the base code
    end for
    Use OSATE connection switch to parse all connections
    Group the connections according to their destination names
    for each algorithm in a sensor component do
        Aggregate incoming connections and create input map
        for each input do
            if it is an output of another algorithm B then
                Find the function call for algorithm B
                if No such function call found then issue an error
                elseWrite function call after algorithm B function call
                end if
            end if
        end for
        Make function call at the timer event handler for the sensor
    end for
end for
```

output of any other algorithm B. If it does match then the corresponding algorithm call is searched in the generated code. If the algorithm B call exists, then the generator writes the function call for algorithm A after that of algorithm B. If it does not exist then the generator issues an error in semantics.

Considering that the AADL input has N sensor components, M algorithms in each sensor components on an average, and P inputs to each algorithm on an average, the complexity of the generator code is $O(NMP)$. Experimental runs with three sensors each with two algorithms and each algorithm having two inputs, shows an average execution time of about 1s.

B. *Generation of the Smart Phone Code*

The smart phone code generation follows a similar procedure as the sensor code generation. The smart phone application on execution presents the user with an interface showing the available sensors in range. The user can then select the sensors that can communicate with the smart phone or setup a multi-hop connection. The UI also provides selection of algorithms and their inputs. However, the current version of smart phone code only supports algorithms with one input. Complex execution sequence of algorithms in smart phones are currently not supported. Further, the smart phone UI can be used to send start, stop and sampling frequency change commands to the sensor. A sample smart phone UI developed using the *Health-Dev* is shown in Figure 4.

V. *Health-Dev* Evaluation and Validation

In this section, we first evaluate *Health-Dev* generated code in terms of the code size, the RAM usage, and the energy consumption and compare with BSNBench [5], benchmark for BSNs. Then we show that the generated code indeed satisfies the requirements using two examples.

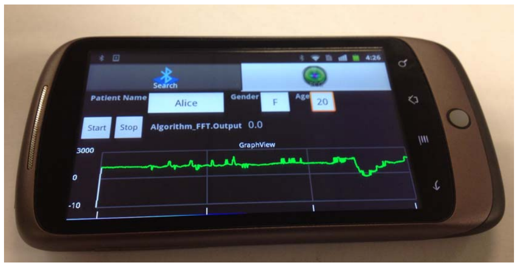


Fig. 4. **Smart phone UI developed using *Health-Dev***

TABLE II
EVALUATION OF *Health-Dev* CODE ON TINYOS BASED SENSORS

Task	*Health-Dev* RAM Size	BSNBench RAM Size	*Health-Dev* Code Size	BSNBench Code Size
FFT	3541	4403	19342	17430
Peak detect	1243	1279	20068	14488
FIR	484	1075	17490	11582
Differential encoding	4443	4275	17538	11512
Out of range	1241	1087	18478	13076
Statistics	4441	2285	17084	14270

A. *Evaluation*

Table II shows the comparison of the *Health-Dev* generated code and the BSNBench code. The code size of *Health-Dev* is always greater than the BSNBench. This is because *Health-Dev* aims to develop generic codes that can be easily modified to serve a different purpose, hence it has many more lines of code delimited by preprocessing directives. On the other hand BSNBench code is optimized for the specific functionality. An interesting fact is that often the RAM consumption for *Health-Dev* code seems to be lower than BSNBench code. This is because in *Health-Dev* variables used in physiological signal processing algorithms (FFT, peak detection, and FIR filter) are declared within the functions itself to make them modular resulting in very less number of global variables. The internal variables are loaded into and evicted from the stack on function call and return respectively. However, in BSNBench the internal variables of a function are often declared as global and reused to optimize RAM usage. Hence, the minimum amount of RAM required for BSNBench tasks is often higher than that of *Health-Dev* implementation but has less variance. However, the RAM usage of *Health-Dev* implementation has more variance and increases during a function call. The energy consumption for the *Health-Dev* generated code was found to be similar to that of BSNBench code, with variances attributed to battery power fluctuations and measurement errors.

B. *Validation*

We consider two case studies: 1) radio duty cycling of sensors, and 2) mobility aware transmit power control. Plug-ins were written in the OSATE framework to mathematically analyze the AADL models. The *Health-Dev* plug-in was then used to generate the code for the verified BSN model.

Radio duty cycling: In this case study, we consider that the sensor is sensing ECG signals and is performing peak detection, and FFT, representative of the ECG based signal processing [5]. In the model based verification stage, the power consumption of the sensor platform in radio on ($P_{ROn}+P_{proc}$) and radio off (P_{proc} only computation power) stages were profiled. The total energy consumption of the sensor platform (E_{sensor}) in time t at a given duty cycle of $x\%$ is given by -

$$E_{sensor} = ((P_{ROn} + P_{proc})\frac{x}{100} + P_{proc}\frac{1-x}{100}) \times t \qquad (1)$$

We take the profiled values of the power consumption as reported in [5] and limit the energy consumption of the sensor to the maximum amount available (150 mW for 6 hrs) by scavenging from human body heat [2]. Considering that the application has to be sustained 24 hrs from 6 hrs of scavenged energy, the allowable duty cycle was found out to be 8.2% from Equation 1. The same processing operation was specified in AADL with the same radio duty cycle and the generated code was downloaded in TelosB motes. The average power consumption, measured over a single operation cycle, was 10.84 mW, which is much less than the average power available from scavenging sources ($150 \times 6/24 = 37mW$). Hence, the implementation satisfies the energy requirements.

Mobility aware transmission control: In a PHMS, the packet error rate (PER) varies considerably due to mobility. As a strategy for reducing the PER, a sensor can estimate the link quality in terms of path loss due to fading on each transmission. If the path loss is above a threshold then it increases the transmission power of the radio else it keeps the transmission power at the lowest value. The BSN system with this radio power control schedule was specified in AADL. In the model based analysis phase, the Ricean flat fading model for bit error rate (BER) as a function of path loss was assumed [12], as recommended by IEEE Task Group 6. Eight levels of transmission power were considered for the sensor ranging from -25 dBm to 0 dBm, typical of the CC2420 radio, and the path loss was varied from 10 dB to 70 dB. Given the BER, the PER was calculated using the equation $PER = 1 - (1 - BER)^L$, where L is the packet length. To simulate human mobility a Levy walk model [13] was used. It was found out from the simulation that the lowest transmission power level -25 dBm, gives a worst case PER of 0.82, while a transmit power of -15 dBm gives a worst case PER of 0.05. For the Levy walk model, with outdoor excursion probabilities ranging from 0.2 to 0.9, alternating between the -25 dBm and -15 dBm transmission levels was enough to keep a PER at 0.12. We assumed that for reliable network operation a PER of at most 0.2 can be tolerated. The resulting model was then passed through *Health-Dev* to generate the code. Experiments were performed in the university campus and outdoor movements were simulated by short excursions outside the department office. The PER was found to be 0.03, which clearly satisfies the requirements in the design phase.

VI. Conclusions

In this paper, we proposed *Health-Dev* to generate code for BSNs that meet the system requirements of energy efficiency, and reliable communication under dynamic user context changes. *Health-Dev* takes requirements verified models specified in AADL as input and automatically generates code for both embedded sensors and smart phones. The implementation of the BSN is shown to also match the system requirements set forth in the modeling phase. *Health-Dev* has a codebase of functions for physiological signal processing that allow rapid prototyping of health-care applications. *Health-Dev*, however cannot currently implement adaptive thresholding algorithms. Further, there is no easy method to specify algorithms that are not in the codebase in *Health-Dev*. We will address these short comings in our future work.

Acknowledgments

The authors thank the reviewers and the shepherd Dr. Emil Lupu for their valuable comments and Google for providing Nexus One smart phones. The research is funded in part by NSF grant CNS-0831544 and IIS-1116385.

References

[1] K. Venkatasubramanian, G. Deng, T. Mukherjee, J. Quintero, V. Annamalai, and S. Gupta, "Ayushman: A wireless sensor network based health monitoring infrastructure and testbed," in *Distributed Computing in Sensor Systems*, vol. 3560. Springer Berlin / Heidelberg, 2005.

[2] A. Banerjee, S. Kandula, T. Mukherjee, and S. K. S. Gupta, "BAND-AiDe: A tool for cyber-physical oriented analysis and design of body area networks and devices," *ACM Trans. on Embedded Computing Systems, Special issue on Wireless Health Systems*, To appear.

[3] V. M. Jones, A. Rensink, T. C. Ruys, H. Brinksma, and A. T. van Halteren, "A formal MDA approach for mobile health systems," in *2nd European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations*, D. H. Akehurst, Ed., no. TR 17-04. Canterbury: University of Kent, 2004, pp. 28–35.

[4] V. Prasad, T. Yang, P. Jayachandran, Z. Li, S. H. Son, J. A. Stankovic, J. Hansson, and T. Abdelzaher, "ANDES: An ANalysis-Based DEsign tool for wireless Sensor networks," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, Dec. 2007, pp. 203–213.

[5] S. Nabar, A. Banerjee, S. K. S. Gupta, and R. Poovendran, "Evaluation of body sensor network platforms: a design space and benchmarking analysis," in *Wireless Health 2010.* New York, NY, USA: ACM, 2010, pp. 118–127.

[6] X. Chen, A. Waluyo, I. Pek, and W.-S. Yeoh, "Mobile middleware for wireless body area network," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, 2010, pp. 5504–5507.

[7] G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio, "SPINE2: developing BSN applications on heterogeneous sensor nodes," in *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, july 2009, pp. 128–131.

[8] M. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri, "A framework for modeling, simulation and automatic code generation of sensor network application," in *IEEE Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08.*, june 2008, pp. 515–522.

[9] J. B. Lim, B. Jang, S. Yoon, M. L. Sichitiu, and A. G. Dean, "RaPTEX: Rapid prototyping tool for embedded communication systems," *ACM Trans. Sen. Netw.*, vol. 7, August 2010.

[10] E. Cheong, E. A. Lee, and Y. Zhao, "Viptos: a graphical development and simulation environment for TinyOS-based wireless sensor networks," in *3rd int'l conf. Embedded networked sensor systems, SenSys '05.* New York, NY, USA: ACM, 2005, pp. 302–302.

[11] W. P. McCartney and N. Sridhar, "Tosdev: a rapid development environment for TinyOS," in *4th int'l conf. on Embedded networked sensor systems, SenSys '06.* New York, NY, USA: ACM, 2006, pp. 387–388.

[12] http://math.nist.gov/mcsd/savg/papers/15-08-0780-09-0006-tg6-channel-model.pdf.

[13] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong, "On the Levy-Walk nature of human mobility," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, april 2008, pp. 924–932.