DOI: 10.1109/PIMRC.2016.7794845

Dynamic Service Switching for the Medical IoT

Philipp Kindt*, Daniel Yunge*, Andreas Tobola†, Georg Fischer‡, Samarjit Chakraborty*
*Technical University of Munich (TUM)
†Fraunhofer Institute for Integrated Circuits (IIS)
‡Institute for Electronics Engineering, University Erlangen-Nurnberg (FAU)

***Abstract*—With the Internet of Things (IoT) becoming a reality, power-efficient techniques are crucial to achieve sufficient battery lifetimes. Whereas current medical IoT devices typically acquire data with a constant quality, we propose an architecture that dynamically adjusts the data quality adaptively based on the current medical condition of the subject being monitored. Since transmission and processing make up a large fraction of the energy consumption, the reduction of the link traffic and processing effort caused by such an adjustment results in a decreased energy consumption of the devices. For example, if anomalies in the monitored data are detected, the monitoring is performed with an increased granularity and more exhaustive processing. Further, not all data generated by the medical sensors needs to be transmitted during all times. Only if certain events are detected, the transmission of the complete data needs to be activated. In this paper, we present a novel approach for body-worn medical IoT devices. In particular, a generic, distributed architecture for the power-management of the whole system, which is based on dynamically switching services, is presented. We show that such an architecture can reduce the energy-consumption of medical sensors by up to 80 % in real-world measurements.**

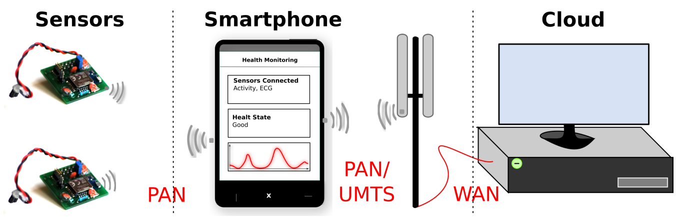

Fig. 1. Typical setup for a medical Internet of Things

I. Introduction

The society is aging in most industrial countries. For example, more than a fourth of the population of Japan is older than 65 years [1]. With the further increasing share of elderlies, the effort a national economy has to spend on care is expected to grow significantly. Because of this, and also because it is perceived as more convenient, many people will spend most of their retirement at home rather than in an old-age home. To support a self-determined and yet safe life of elderly people, smart tele-monitoring systems similar to the one depicted in Figure 1 are being actively studied today. In such a network, multiple sensors are located on a patients body. We assume that this system is flexible, in a way that the actual setup of sensors can be changed as needed. For example, a mobile electrocardiograph (ECG) can be added if needed in combination with an activity sensor, or sensors for oxygen-saturation can be used for other patients. These devices are connected wirelessly to a smartphone which establishes the connection to the Internet using its 3G/4G interface. An application-specific server which is managed by a care service provider evaluates the data from the sensors. If the parameters indicate an accident such as a fall or a medical incident such as an abnormal heart rate, a physician is contacted or the person is given a call to ask for his wellbeing.

Battery life and hence reducing the power-consumption is a major challenge in the usability of such devices, since it is unreasonable to have patients take off their on-body sensors for recharging them or to replace their batteries frequently. In addition to the known power management techniques, for medical IoT systems, there is additional domain-specific potential for substantial power reductions. Current architectures assume that the quality and amount of data recorded and transmitted is constant. However, from a medical point of view, large parts of this data or its quality are not required in many cases. For example, consider an ECG sensor. Rather than sending the whole waveform to the smartphone during all times to detect arrhythmia on the server, in some situations, it is sufficient to classify the QRS complex on the sensor and sending the processed data with a certain period. Thereby, the amount of data transmitted decreases and hence the energy consumption both for the smartphone and the sensor is reduced. However, as soon as arrhythmias or other anomalies are detected, the physician might be interested in the ECG waveform and therefore the transmission of the ECG needs to be triggered dynamically. Moreover, motion artifacts often derogate the validity of the ECG signals. Therefore, as soon as significant motion is detected, the ECG sensor can be switched off to save power.

Dynamically activating and deactivating services imposes the need for an automated evaluation system of the data, which autonomously detects certain events, triggers the appropriate actions and, if required, presents small chunks of pre-processed data to the medical personal. The features to be attentive on depend on the individual patient and his specific medical condition. Therefore, medical knowledge needs to be incorporated into the decision system for each scenario individually. Such an incident detection system can also be extended towards making power-management decisions online. To illustrate the techniques presented, we consider the following scenario as a running example throughout the paper.

An elderly person often feels dizzy and therefore fell unexpectedly multiple times during walking. As a cause, taking into account the medical history of the patient, the doctor assumes a heart disease as a cause of the dizziness. Therefore, to verify this assumption, whenever a fall or an abnormal heart beat is detected, the ECG waveform is recorded and sent to the cloud for the purpose of a later manual observation by the medical doctor. Since frequent premature ventricular contractions can lead to dizziness [2], the system uses the output of a beat classifier (BT) [3]. In addition, an acceleration sensor detects whether the persons rests or is in motion. If there is significant motion detected, the ECG waveform is potentially invalid and the ECG sensor is switched off to save power until the person rests again. Further, in phases with significant motion, the acceleration sensor switches to a mode in which its sends the raw acceleration data to the phone. On the phone, the data can be analyzed more thoroughly, since more computational power is available, e.g., for performing fall detection [4] or gait analysis [5], or a later offline analysis.

It needs to be pointed out that the actual setup of sensors and the incidents to be attentive on is individual for a patient and his diseases. In this paper, we propose a system architecture that allows for such a flexibility. It is capable of detecting suitable situations for updating the monitoring granularity autonomously and taking the appropriate actions. Dynamic rules, which are individual to the setup of sensors and the medical background of a patient, are implemented throughout the system in a distributed fashion. The main purpose of these rules is – besides warning relatives and caregivers in the case of emergency situations – trading monitoring granularity against energy consumption in an optimized manner. The rules can trigger the following actions, which have a significant impact on the energy consumption:

1) Switching on/off virtual channels: The data recorded by the sensors can be evaluated at different physical locations of the system, viz. the sensor, the smartphone or the cloud. Often, it is meaningful to only transmit the data which is really of interest at the remote side and to carry out as much preprocessing as possible on the sensors, as we will show in Section III. To realize this, we define multiple virtual *data channels* for each sensor, which are associated to different amounts of local processing and traffic on the wireless links. For example, an ECG sensor offers a raw data channel and multiple channels for processed data, e.g. heart rate (HR), heart rate variability (HRV) and an abnormal beat classifier (BT). If no raw data is needed in the cloud, only e.g. the HR-channel might be transmitted and therefore the traffic on the link and the energy consumption is reduced.

2) Adjusting the Measurement Quality: Each data channel has a quality parameter $Q \in [0, 100]$ assigned to it. Its interpretation is channel-specific and maps to concrete values of the following low-level parameters: Sampling rate of the raw data, measurement interval T_m (e.g., a value of the HRV is transmitted every 2.5 minutes), transfer latency, computation effort, quantization and others.

Such an approach is expected to scale well for larger number of sensors and subjects, since a) the amount of data and hence computation is reduced, b) processing is done where it is most efficient and c) the technique is generic for a large range of different setups. While such optimizations have been considered for some specific cases (e.g., in [6] the impact of end-to-end-delay of the link on the energy consumption has been studied, or in [7], a MAC protocol for medical data has been presented which only sends packets if the values are abnormal), to the best of our knowledge, no holistic and generic approach to exploit different service quality requirements dynamically has been studied. Existing solutions are restricted to static and simplistic rules (e.g., simple thresholds) for the decision, thereby risking to miss important medical incidents. An approach which scales for multiple setups, almost all conceivable monitoring scenarios and patient individualities requires new architectures and techniques. In this paper, we for the first time propose a solution towards this, and make the following contributions:

1) We propose a system architecture for the medical Internet of Things, which adapts the monitoring granularity dynamically to the current medical requirements for all conceivable scenarios during runtime.

2) Using models and real-wold measurements, we evaluate the potential energy savings of such a system. The results show that significant amounts of energy can be saved by dynamic service switching, e.g. $80\,\%$ of a mobile ECG or $62\,\%$ of an acceleration sensor.

Besides energy savings, our proposed concept offers more flexibility than current solutions. In addition, dynamic service switching increases the privacy of the users, since data is only transmitted to the cloud when it is really necessary.

The rest of this paper is organized as follows. In Section II, we describe our proposed architecture and all of its components in detail. Next, in Section III, we evaluate the potential savings by results that we obtained from real-world measurements, literature data and energy models. Finally, in Section IV, we conclude our work.

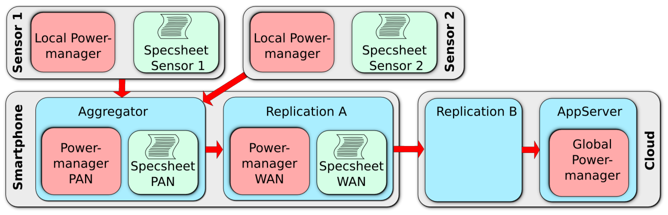

Fig. 2. Overview of our proposed architecture

II. System Architecture

A. Overview

An overview of our proposed system is given in Figure 2. At the top, there are multiple sensors which collect health data and transmit them via Bluetooth Low Energy (BLE) to a smartphone. On the phone, there are two pieces of software. The one is denoted as the *aggregator*. Its main purpose is forwarding the data from the sensors to the next building block and vice-versa. The other piece is the *replication system* which is composed of two parts: One is located at the smart phone (A) and one on an Internet server (B). Whenever a UMTS/LTE connection exists, the replication system transmits the data from side A to side B and vice-versa, while buffering it in the meantime. On the other side, there are one or mutiple *application servers* (*AppServers*) connected for evaluating the data.

To be energy-efficient, each node contains a local power-manager. It only has access to the information available on the device it is located on and attempts to reduce its energy consumption, as described below. In addition, there is a global power manager which chooses a policy based on the (medical) monitoring requirements. This policy is then implemented in a distributed fashion. To achieve the necessary communication among the nodes, we define multiple common interfaces which are denoted as *specSheets*. In the following, we describe our proposed system in detail, using the use-case described in the previous section as a running example.

B. Sensors and virtual Channels

One physical node can be the source for different types of data. First, it might contain multiple dedicated sensing devices (e.g., an accelerometer and a gyroscope). Second, instead of sending all the raw data to the cloud, it is in many cases more energy efficient to extract and send certain parameters from the raw data. For example, an ECG sensor can generate a raw ECG waveform, or multiple pre-processed parameters such as the as HR or HRV. To allow for exploiting these potentials for saving energy systematically, we define the abstraction of logical *channels*. Each channel behaves as a virtual, independent sensor that delivers only one particular type of data (e.g., HR). Each channel can be activated and deactivated independently from each other. Clearly, the set of active channels determines the amount of on-node processing and bytes per second to transmit. Each channel defines a unique identifier (UUID) similar to ISO/IEEE 11073, the interpretation of the data sent over the channel (viz., how to translate the bitstream into physical values) and a specSheet as described in detail in Section II-C.

C. SpecSheet Interfaces

To cope with the flexibility of allowing new, unknown sensors to join the system during runtime, we define an interface for querying and modifying parameter values. When a new sensor is added, the only assumption we impose is that the data-channels offered by the sensor are known. Each channel provides a *specSheet* interface. A specSheet is a table in which each row corresponds to one specific parameter. Each parameter has an unique id, some flags (readable/writable), the minimum/maximum value (if applicable) and the actual parameter value. Every node in the network can either read the whole specSheet or poll/write a single row of it. Its actual parameters are as follows.

- A quality parameter Q_k. Its interpretation is channel-specific.
- The maximum allowed latency for communication and processing, since the latency can be traded with energy consumption in BLE links [8].

TABLE I
OPERATION MODES M OF THE ECG

M	T_m	HR	HRV	BT	RAW	$P[\mu W]$	$R[bps]$
M1	ES	✓	✓	✓		130	1.4
M2	cont	✓				402	10.7
M3	cont	✓				585	10.7
M4	cont	✓	✓	✓		1218	10.8
M5	cont				✓	947	8000.0
M6	cont	✓	✓	✓	✓	1218	8010.8

- A read-only-parameter which contains the required throughput for the current quality level. It is used by the power manager of the personal area network (PAN) to determine the link throughput, as described below.
- An on/off- switch for the whole channel.

Whenever a specSheet has been modified, all participants at the right side of the software module the value has been changed at (according to Figure 2) need to be notified. To illustrate the proposed concept, we consider the following sensors as a running example.

1) ECG sensor: In this paper, we consider an custom adjustable ECG sensor with multiple modes, which define different amounts of processing for different output data types [9]. The sensor has been designed for maximum scalability at low power consumption in the range of 100 to 1000 μW, depending on the configuration. In particular, its mode can be switched during runtime to obtain a raw ECG waveform or different sets of preprocessed parameters, viz. HR, HRV, and a beat type classifier (BT) for abnormal beats, which detects premature ventricular contractions (PVC).

The energy-consumptions and data-rates of this sensor in its different modes are summarized in Table I. Mode M1 supports *episodic sampling* (ES) [10], which works as follows. The sensor measures data continuously for 2 minutes at a sampling rate of $1000\,\mathrm{Hz}$. After that, it sends the values of the HR, the HRV and the BT computed during this period and sleeps for 13 minutes. Next, the procedure repeats. This mode causes a high latency of up to 15 minutes, but further reduces the energy-consumption. Mode M2 is a low-power alternative using an analog comparator to detect R-peaks with a reduced HR-accuracy instead of the more complex algorithm used in the other modes. Mode M3 operates at a sampling rate of $200\,\mathrm{Hz}$ and a low bandwidth of 7 to $24\,\mathrm{Hz}$, which is sufficient to compute the HR, but too low for the other parameters. Mode M4 delivers the same parameters at the same quality as mode M1, but sampling is done continuously. Mode M5 is a raw data mode in which no parameters are computed directly on the sensor. In mode M6, the sensor computes and sends all parameters in addition to the raw data. For our ECG sensor, the HR influences the power consumption only in the modes M1, M2 and M6 due to the beat classifier. The results for $80\,\mathrm{bpm}$ are also contained in Table I.

The sensor can be integrated into our proposed architecture as follows. A local power-manager activates the appropriate mode of the ECG sensor, based on the set of channels activated.

D. Acceleration Sensor

As already mentioned, our proposed architecture is generic and works for almost all possible setups of sensors. To demonstrate this flexibility, we have implemented a custom 3-axis acceleration sensor. It is based on an ARM-Cortex M0 SOC with a BLE radio and a 3-axis accelerometer. The sensor has a sleep current of a few μA, only. Like the ECG sensor, the acceleration sensor provides options to trade data quality against power consumption. In particular, these capabilities are as follows.

- Adjustable sampling-rate: The quality-parameter Q is interpreted as the percentage of the maximum sampling rate $f_{s,max} = 100Hz$, such that $f_s = Q$.
- Detection algorithm: In addition to the raw acceleration, another data channel contains the result of a light-weight on-sensor activity detection. The detected activity (viz. resting, minor movements or intense activities) is sent whenever the state of motion changes.

The activity detection is realized as follows. Let $\vec{a_i} = (a_{x,i}, a_{y,i}, a_{z,i})$ be the measured acceleration at a given instance i of the sampling period and $\vec{a_{i-1}}$ the acceleration at the previous period $i-1$. We define $\Delta a_i = a_i - a_{i-1}$ for all dimensions x,y,z and σ_i as a measure for the amount of activity, with $\sigma_0 = 0$. σ_i is computed recursively with

$$\sigma_i = (1-\alpha)\sigma_{i-1} + \alpha \cdot (|\Delta a_{x,i}| + |\Delta a_{y,i}| + |\Delta a_{z,i}|). \quad (1)$$

Equation 1 implements an exponential smoothing method [11] with a parameter α to combine previously taken acceleration samples with the currently sampled instance i. For a sampling frequency of $f_s = Q = 100Hz$, we empirically selected α = 0.01. Like for the raw acceleration channel, the activity detection can be performed using different sampling rates to trade energy consumption against signal quality. If fewer samples are taken in the same period, the exponential smoothing introduces higher latencies and previously taken measurements dominate the value of σ_i. Therefore, whenever the sampling rate is modified using the specsheet interface, α needs to be adjusted by $\alpha(f_s) = \alpha(100Hz)^{\frac{100Hz}{f_s}}$. If σ_i is below a threshold σ_{rest}, the algorithm classifies the subject's motion as *resting*. If it exceeds σ_{rest}, but is below σ_{minor}, *minor activity* is detected. If it exceeds σ_{minor}, the motion is classified as *significant activity* (e.g., running).

The sensor can be represented as two data channels, one for the raw acceleration and one for the detected activity. The physical sampling rate the accelerometer is read with is the maximum one selected for the two channels.

The acceleration sensor is connected via BLE with a connection interval of $8.75ms$ and a slave-latency of 114.

E. Aggregator & Replication

The data from the sensors is transmitted wirelessly to the smartphone. The aggregator manages the connection between the smartphone and the sensors and forwards the data to the replication via a generic interface. Typically, IoT connectivity solutions such as 6LoBTLE [12] assume that there is a permanent connection between the gateway and the cloud. In our case where the smartphone acts as a gateway, the internet connection might be unavailable during certain periods. Therefore we propose using a replication system rather than directly relaying the data. The replication system serializes the data received to generate a bitstream. This bitstream is stored in a local database, first. If a WAN connection (LTE/UMTS/GPRS) exists, the smartphone then synchronizes the local database with the remote one. SpecSheet data and rules can be exchanged between both sides.

F. Application Server

The application server (*AppServer*) is the sink for the sensor data. Its main purpose is displaying and analyzing medical parameters. For example, the software can be used for the acquisition and analysis of the data in clinical studies. Also, doctors can use long term monitoring to detect hidden diseases easier and caregivers can use it for detecting emergency situations. Since medical knowledge is available on the AppServer, it also serves as the global power-manager. This functionality is described next.

G. Power-Management

1) Global Power-Management: As already mentioned, knowledge concerning the medical background which is individual to the patient is available at the *AppServer*. For that purpose, it contains a rule generator to generate scripts which switch between multiple operating modes- and qualities of the sensors autonomously during runtime.

These rules are created using an intuitive graphical user interface which can be operated e.g. by a medical doctor. Its purpose is to generate executable script files which implement these rules. To generate the code, multiple approaches are feasible. For example, predefined templates for different diseases can be combined and customized using a graphical wizard. Technical details of the system (e.g., available data channels, etc.) do not need to be known by the person generating the rule. For example, a valid rule could be "*record a raw ECG whenever the heart-rate is higher than* $210\,\mathrm{bpm}$ *and no motion is present*", which triggers the activation and deactivation of the necessary virtual channels without the need of being aware of them. The script-files generated are then sent to the nodes they shall be executed on, as described below. These scripts can trigger multiple actions. For example, if a certain threshold of vital parameters is exceeded for longer than a predefined time period, notification messages can be sent by e-mail and short message service to a mobile phone. Similarly, the monitoring granularity can be modified by switching on and off data channels and modifying their quality parameters adaptively.

A rule which realizes be the scenario described in Section I can be defined as follows: First, only the beat type (BT) channel is active and no raw data is transmitted to the application server. As soon as the BT becomes unusual, the raw ECG channel is switched on with a quality of $Q = 100$. The code for such a rule is exemplified in Algorithm 1.

In another example, we consider a setup containing an ECG sensor and an accelerometer. The AppServer is interested in the validity of the ECG waveforms by activating both the ECG waveform channel with a high quality level, and the activity channel with a low quality parameter simultaneously. The activity is monitored in addition to the ECG because any movement of the patient might disturb the ECG signal. The resulting rule scripts need to be executed to implement the power-management policy, as described next.

2) Distributed rule implementation: It would be possible to run the rule scripts locally on the AppServer, only. The AppServer would then monitor all relevant data channels and adaptively set the values of all specSheets. However, there might be undesirable cases where raw data needs to be sent to the AppServer in order to check the rules, even though this data is not of any medical interest at the current point in time. Therefore, energy would be wasted. Furthermore, the connection between the smartphone and the AppServer might be unavailable and the service switching might fail. Therefore, we propose using light-weight code interpreters both on the smartphone and the sensors. A rule script from the AppServer can be replicated to and executed on the remote side to implement it. The interpreted code is executed locally, monitors the parameters and triggers appropriate actions as needed. If channels from multiple sensors need to be examined, the code is executed on the smartphone instead of the sensors. We consider the PicoC interpreter [13] since it is small, easy to use and integrates seamlessly into our proposed framework. The code-interpreter redresses the need of sending raw data in the cases described above, and ensures the execution of rules even when the WAN connection is unavailable.

To allow for a meaningful control, the local power-managers on the sensors and the smartphone provide an API to the interpreted code. It provides functions for reading and writing the values of all specSheets on all devices, with the exception that interpreters on the sensors cannot access any values on the other sensors. In addition, the API supports reading the payload-values from all data-channels which are currently available on the device the interpreter is located on. Using this simple interface, the power-management policy can be implemented at the nodes where it is most energy-efficient. A pseudo-code which could be run on the ECG-Sensor to exemplify the scenario described in Section I is given in Algorithm 1. Here, an abnormal heartbeat triggers the activation of the raw ECG transmission. In the first 3 lines, constants for UUIDs for the data channels are defined. The code reads the data channel for the beat type (ID_BT) by calling the API-function *readDataChannel()*, checks the resulting data for abnormal beats and modifies the specSheet of the raw ECG channel to activate it, next. Whereas this rule is a simplistic example, arbitrary complex scenarios can be implemented using this technique.

3) Local Power-Management: Multiple local power managers fulfill two purposes: 1) They execute the rule scripts which the application server has deposited in them and 2) they optimize the energy consumption locally given the values in the specSheets by incorporating hardware-specific knowledge. In addition to a local power-manager in each physical sensor, the following ones are needed.

Algorithm 1 Sample code for heart rate surveillance

```
$ID\_BT \leftarrow 1$
$ID\_RAWECG \leftarrow 2$
$ID\_RAWECG\_ON \leftarrow 3$
while true do
    $bt \leftarrow readDatChannel(ID\_BT)$
    if $bt == 1$ then
        $writeSpecSheet(ID\_RAWECG, ID\_RAWECG\_ON, 1)$
    end if
end while
```

PAN: The power-manager for the PAN is located in the aggregator and manages and energy-optimizes the wireless link. In BLE, the main parameter that determines the available link throughput is referred to as the *connection interval* T_c. The device wakes up every T_c instances of time. At each wakeup, a *connection event* takes place, in which N_{seq} pairs of packets are exchanged. It has been shown in the literature, e.g. [14], that the connection interval greatly influences the energy consumption of the link. In addition, there is a parameter N_{sl}, referred to as the *slave latency*. It defines a certain number of events the slave might skip before waking up. To transmit with a given data rate, a certain connection interval is needed. Therefore, the power-manager for the personal area network (PAN) reads the traffic demand of each channel from the specSheets of all active channels and chooses the right connection interval. The optimum connection interval for a given data rate R is computed using $T_c = \frac{20\,\text{bytes}\cdot N_{seq}}{R\cdot(1+N_{sl})} \cdot \eta$, with η being the fraction of successfully transmitted packets in relation to the total amount of packets sent.

WAN: The power-manager for the WAN controls the buffering and the replication between the cloud and the smartphone. Thereby it optimizes the energy consumption of the WAN. For example, UMTS/3G remains in an active state for 15 seconds after each transmission before switching the radio off [15]. Hence, continuously transmitting small chunks of data is wasteful and should be mitigated by adaptive buffering. The task of the power-manager is reading a maximum allowed communication latency parameter in its specSheet and controlling the buffering duration considering the amount of data which is currently delivered by the sensors.

III. Evaluation

We have implemented multiple parts of our proposed architecture. Based on this, in the following, we quantify the energy savings of dynamic service switching.

A. Service Switching for the ECG sensor

For evaluating the achievable energy savings of dynamic service switching in the ECG sensor, we consider changing from sending only the beat type BT and HR/HRV to sending raw ECG waveforms dynamically, thereby realizing the scenario described in Section I.

We have extended the energy-analysis from [9], mainly by computing the bitrates required for the data transmission in all modes, as summarized in Table I. The table shows 6 different modes of operation and their measurement strategies, which are either continuous sampling (M2-M6) or episodic sampling (M1). Further, it shows the data channels offered in each mode, the corresponding energy-consumption for processing and the required link bitrate R. The bitstream with rate R needs to be split up into multiple BLE GATT packets in a power-optimized fashion. A feasible strategy would be using the largest possible connection interval T_c for each situation in order to maximize the duty-cycle. This packet partitioning works as follows. First, we compute the ideal connection interval, $T_{c,i}$, which is the connection-interval required for a given bitrate R without taking into account additional constraints of BLE. Since 20 bytes of payload fit into one packet, $T_{c,i}$ is defined as $T_{c,i} = \lfloor \frac{20\,\mathrm{by}/\mathrm{R}}{1.25\,\mathrm{ms}} \rfloor \cdot 1.25\,\mathrm{ms}$. If $T_{c,i}$ is smaller or equal than the maximum connection interval supported by BLE (viz. $10.24\,\mathrm{s}$), then we set the connection interval to $T_{c,i}$ and set $N_{sl} = 0$. If $T_{c,i}$ exceeds $10.24\,\mathrm{s}$, the connection-interval chosen is $10.24s$ and the slave will regularly skip some of the connection intervals by making use of the slave latency feature with $N_{sl} = \lfloor T_c/10.24 \rfloor - 1$, such that the maximum effective connection interval of $32\,\mathrm{s}$ defined by the BLE standard [16] is never exceeded. The mean number of bytes in each packet is defined as $N_b = R \cdot T_{c,i} \cdot (N_{sl} + 1)$ for all the three cases described. Since N_b can be a non-integer number, we account for that by simulating 1000 packets with the right fraction between $\lfloor N_b \rfloor$ and $\lceil N_b \rceil$ bytes and taking the mean. Using known energy models from the literature (e.g., [17] or [14]), we can compute the energy consumption of the BLE slave. We assume that the master always sends an empty polling packet containing 10 bytes of protocol overhead. The protocol overhead of a non-empty packet is 17 bytes.

The results of these computations are depicted in Figure 3. It shows the energy-consumption of the ECG both for the computation and transmission. As can be seen, in M5 and M6, in which raw data is sent, significantly more energy is needed than for modes in which the HR, HRV and BT are computed and sent continuously without the raw waveform. In the scenario considered, as described in Section I, the ECG would operate in M4 until abnormal beats have been classified. In the case of abnormalities, it would switch to M5 autonomously. As a result, during most times, the device would consume 80% less energy ($1.24\,\mathrm{mW}$ in M4 vs $6.24\,\mathrm{mW}$ in M5) due to dynamic service switching.

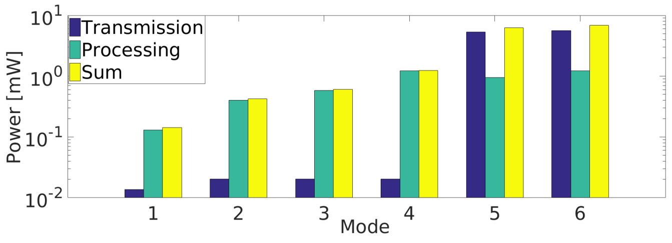

Fig. 3. Power consumption of the ECG sensor

B. Service-Switching of the Activity Sensor

In this section, we evaluate the potential energy-savings of the acceleration sensor by real-world current measurements.

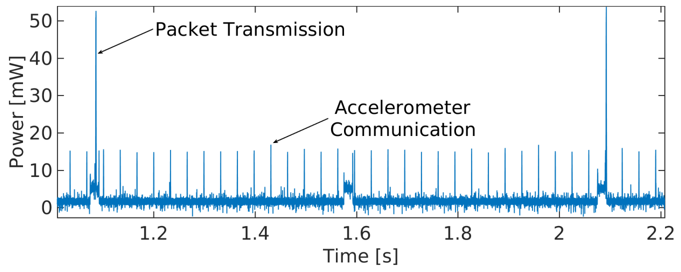

Fig. 4. Power consumption waveform of the acceleration sensor

Figure 4 shows the measured power-consumption in a situation in which no raw data is sent, the activity detection is active and the detected activity remains constant. The sampling-rate in this example is $30\,\mathrm{Hz}$. As can be seen, there are 30 small peaks per second for acquiring the acceleration samples from the MEMS sensor. In addition, there is one larger peak per second, which is related to the exchange of empty BLE polling and response packets. Further, there are certain short periods of CPU activity for processing.

In general, if the sampling-rate of the accelerometer is increased, more of the sampling-peaks will occur. If only the detected activity is sent, data packets are transmitted whenever the detected activity changes. Such changes typically occur only rarely. Hence, for the activity channel, we assume that the activity remains constant for all measurements, and one empty packet per second is sent to maintain the connection.

The mean power consumption of the sensor for different quality levels Q is shown in Figure 5. Recall that for the acceleration sensor, the quality level Q is identical to the sampling rate f_s. As can be seen, the power consumption increases nearly linear with the sampling rate. Sending the raw data consumes significantly more energy than performing the low-complexity activity detection on the sensor, since more packets are sent.

In the scenario described in Section I, as long as no activity is present, the sensor would perform activity detection with a low sampling rate. We empirically found that $f_s = 10\,\mathrm{Hz}$ is sufficient for detecting whether the subject is in motion or not. If any activity is detected, the sensor would switch to sending the raw acceleration with a high sampling rate, e.g. $100\,\mathrm{Hz}$ for a thorough analysis on the smartphone. In this manner, 62% of the energy-consumption of the sensor can be saved in phases in which the subject rests. However, there is an additional overhead for dynamic service switching caused by the code interpreter, which is quantified next.

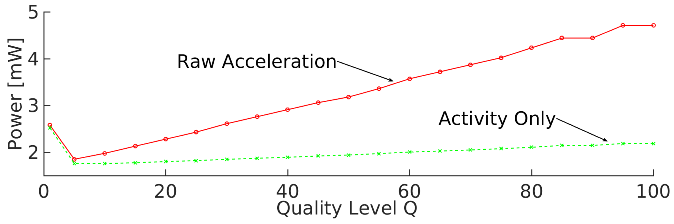

Fig. 5. Mean power consumption of the acceleration sensor

C. Overhead of Code Interpretation

For executing the rule scripts on the smartphone and on the sensors, a light-weight but yet full-featured code interpreter is used. From previous studies [18], it is known that code-interpreters raise a large overhead to the execution times of the code. The overhead results in longer active phases of the CPU, thereby consuming additional amounts of energy.

To measure this overhead, we have installed the open-source interpreter PicoC [13] on a processor with the same micro-architecture and clock-rate as on the ECG sensor and measured the execution times of the interpreted rule scripts. These times have been multiplied by the current consumption of the processor and its voltage. Our interpreted evaluation code carries out the following task: It checks an integer value once every period T_i to execute some action, similar to the code presented in Algorithm 1. The if-clause it contains is assumed to always evaluate to *false*. As an extreme example, we further considered a script which contains a loop to execute this if-clause 100 times in a row, which can be seen as a very complex rule. In Table II, we present the energy E_i consumed for one code execution and the computed mean power overhead for different execution intervals T_i. As can be seen, the overhead is low and shrinks with increasing intervals T_i.

TABLE II
OVERHEAD OF THE RULE SCRIPT EXECUTION

	E_i	$T_i = 0.2s$	$T_i = 2.0s$	$T_i = 20.0s$
1	$18.7\,\mu$J	$93.6\,\mu$W	$9.36\,\mu$W	$0.94\,\mu$W
100	$2500\,\mu$J	12.5 mW	1.25 mW	$125\,\mu$W

IV. Concluding Remarks

In this paper, we have proposed an architecture for the medical IoT, which systematically exploits dynamic service switching. Our evaluation shows that this technique can save up to 80 % of the energy of an ECG and 62 % of an acceleration sensor in a realistic scenario. From these results, we conclude that such a technique has a great potential for extending the battery life of small sensors significantly. In addition, dynamic service switching can, especially combined with other approaches (e.g., cryptographic strategies [19]), increase the privacy of the users.

The architecture proposes a distributed rule evaluation by using code interpreters. Whereas the overhead of the code-interpretation in terms of computation is low, interpreters require significant amounts of RAM [18]. In our evaluations, a cortex-M4 CPU with 96 kBytes of RAM has been used, whereas the CPUs of our sensors have significantly smaller RAM sizes. In further research, code-interpreters which are currently designed for larger MCUs need to be tailored to run on smaller controllers with 16 kBytes of RAM and less.

V. Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. This work was partially supported by *HE2mT - High-Level Development Methods for Energy-Saving, Mobile Telemonitoring Systems*, a project funded by the federal ministry of education and research of Germany (BMBF). While the authors mentioned on the first page have carried out the main work for this paper, the following persons also contributed to this work: Christian Hofmann, Christian Weigand, Bruno Ristok, Thomas Herbig, Helga Mühlbauer, Thomas Huber, Robert Diemer, Nils Heitmann, Onur Desde.

References

[1] Statistics Bureau in the Ministry of International Affairs and Communications of Japan, "Population estimates in May 2015." [Online]. Available: www.e-stat.go.jp/SG1/estat/ListE.do?lid=000001134232

[2] J. O. Coffey, J. F. Viles-Gonzalez, J. Willner, and D. Mehta, "Premature ventricular contraction-induced symptomatic slow pathway conduction successfully treated by catheter ablation," *Circulation: Arrhythmia and Electrophysiology*, vol. 6, no. 4, pp. e56–e57, 2013.

[3] P. Hamilton, "Open source ECG analysis," in *Computers in Cardiology (CINC)*, Sept 2002.

[4] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, and T. Jms, "Comparison of low-complexity fall detection algorithms for body attached accelerometers," *Gait & Posture*, vol. 28, no. 2, pp. 285 – 291, 2008.

[5] J. Klucken, J. Barth, P. Kugler, J. Schlachetzki, T. Henze, F. Marxreither, Z. Kohl, S. R., J. Hornegger, B. Eskofier, and J. Winkler, "Unbiased and mobile gait analysis detects motor impairment in parkinson's disease," *PLoS One*, vol. e56956, no. 8, 2013.

[6] K. Wac, M. Bargh, B.-j. Van Beijnum, R. Bults, P. Pawar, and A. Peddemors, "Power- and delay-awareness of health telemonitoring services: the mobihealth system case study," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 4, pp. 525–536, May 2009.

[7] A. Ahmad, N. Javaid, Z. Khan, M. Imran, and M. Alnuem, "iA-MAC: Improved adaptive medium access control protocol for wireless body area networks," in *International Symposium on Communications and Information Technologies (ISCIT)*, Sept 2014, pp. 156–160.

[8] P. Kindt, D. Yunge, M. Gopp, and S. Chakraborty, "Adaptive online power-management for bluetooth low energy," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2015.

[9] A. Tobola, C. Espig, F. J. Streit, O. Korpok, H. Leutheuser, B. Schmitz, C. Hofmann, M. Struck, C. Weigand, B. Eskofier, and G. Fischer, "Scalable ECG hardware and algorithms for extended runtime of wearable sensors," in *IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, 2015.

[10] L. K. Au, M. A. Batalin, T. Stathopoulos, A. A. Bui, and W. J. Kaiser, "Episodic sampling: Towards energy-efficient patient monitoring with wearable sensors," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2009.

[11] A. Watts, "On exponential smoothing of discrete time series (corresp.)," *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 630–630, 1970.

[12] J. Decuir, "Bluetooth smart support for 6LoBTLE: Applications and connection questions." *IEEE Consumer Electronics Magazine*, vol. 4, no. 2, pp. 67–70, April 2015.

[13] Z. Saleeba, "Picoc project homepage," available via code.google.com/p/picoc.

[14] P. Kindt, D. Yunge, R. Diemer, and S. Chakraborty, "Precise energy modeling for the bluetooth low energy protocol," *arxiv.org*, 2013. [Online]. Available: http://arxiv.org/abs/1403.2919

[15] B. Zhao, Q. Zheng, G. Cao, and S. Addepalli, "Energy-aware web browsing in 3G based smartphones," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, July 2013, pp. 165–175.

[16] Bluetooth SIG, "Specification of the bluetooth system 4.0," June 2010, volume 0, available via bluetooth.org.

[17] M. Siekkinen, M. Hiienkari, J. Nurminen, and J. Nieminen, "How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4," in *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2012.

[18] D. Yunge, P. Kindt, M. Balszun, and S. Chakraborty, "Hybrid apps: Apps for the internet of things," *IEEE International Conference on Embedded Software and Systems (ICESS)*, 2015.

[19] E. Klaoudatou, E. Konstantinou, G. Kambourakis, and S. Gritzalis, "A survey on cluster-based group key agreement protocols for wsns," *IEEE Communications Surveys Tutorials*, vol. 13, no. 3, pp. 429–442, 2011.