Dynamic Service Synthesis and Switching for Medical IoT and Ambient Assisted Living

Daniel Yunge*, Sangyoung Park*, Philipp Kindt*, Graziano Pravadelli†, and Samarjit Chakraborty*
*Technische Universität München (TUM), Email: {yunge,park,kindt,chakraborty}@rcs.ei.tum.de
†University of Verona, Email: graziano.pravadelli@univr.it

***Abstract*—In the Internet of Things (IoT), our surrounding will include a large variety of devices from different manufacturers. One of its promising branches, the medical IoT, will also be accompanied by heterogeneous smart-home infrastructures. However, the efficacy of a medical IoT application will depend on how well the surrounding smart devices collaborate with it to serve the individual needs of the users. Pre-programmed solutions lack flexibility to adapt to each need and environment, and fail to make full use of the capabilities of a set of smart devices. In this paper, we propose a concept based on the flexible and user-friendly synthesis and switching of services for medical IoT applications. The crux of the concept is to provide a methodology in which non-experts can dynamically define services based on their needs. We describe a potential scenario, discuss the associated challenges, and present preliminary results on the feasibility of this approach. Particularly, we focus on design aspects for realizing the concept and propose the use of interpreters on the smart devices as alternative solution. We show that such an approach is feasible in terms of implementation and energy consumption while still maintaining the full flexibility of the service synthesis.**

***Keywords*—Ambient Assisted Living, Cyber-Physical Systems, Automatic Synthesis, Wireless Sensor Networks**

I. Introduction

The Internet of Things (IoT) is becoming a reality with many of everyday objects becoming "smart". For example, light bulbs can be controlled remotely to change their color, or windows offer wireless interfaces for opening or closing them. In addition, wearable medical devices, such as heart rate sensors [1], sleep monitoring systems [2] or gait posture analysis sensors [3] are widely used and studied in recent literature. The large number of smart devices in various environments such as home or work opens up the possibility for an unforeseeable large variety of scenarios in which a person can interact with the environment. However, most of the contemporary IoT devices feature only a limited number of pre-programmed functionalities as the manufacturer do not consider applications in which its device interacts with others. This prevents multiple smart devices from interacting flexibly and providing customized services to the users. As a result, only a small subset of the range of feasible applications can be realized in existing IoT setups, which significantly deteriorates their usefulness in supporting our everyday life.

In the scenarios we target, multiple networked sensors and actuators are installed in a smart home of a subject as shown in Fig. 1. Wearable sensors monitor the health status of the subject. The data can be transmitted to the cloud to be observed by medical experts in the case of abnormalities, as shown in the top of the figure. Furthermore, the wearables interact with the devices in the environment, such as heaters, doors, windows, or sensors for measuring temperature, brightness or humidity. As mentioned, while such hardware setups are available today, their functionality is mostly predefined [4]

978-1-5090-4270-8/16/$31.00

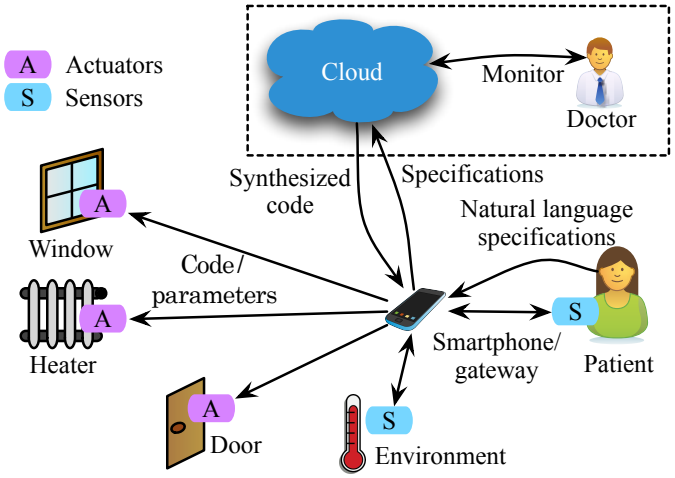

Fig. 1: Dynamic service synthesis and switching framework.

and therefore, possible applications are limited. Moreover, current smart homes typically support only the triggering of simple actions based on basic rules, such as checking for the exceedance of numeric thresholds that can be configured in e.g., a web interface. However, even configuring such simple applications is often too burdensome for a layperson.

Nevertheless, in many cases, useful applications depend on the individual needs of the user and therefore need to be set up by a layperson. For example, based on measurements of the sleep quality and the room temperature, the user might want to specify that the window should be opened autonomously during the night, if the sleep quality is low. Or, on rainy days, the smart light bulb could be requested to imitate the color and brightness of a natural sunrise if the curtains have not been closed by the user. Such a behavior cannot be realized with ease using pre-programmed IoT devices if possible at all. Moreover, the interaction among multiple software platforms from different vendors is often not possible bringing up the issues of interoperability.

To overcome these major limitations of current IoT setups, a radically different approach in designing the software of such devices is needed. Rather than attempting to foresee and implement all useful applications at the design time, we propose to synthesize services dynamically during runtime, based on the requirements from the users and the set of surrounding devices in a particular environment. Such a system can be especially beneficial when targeted towards medical IoT devices, which interact with smart-home appliances, but we believe that it can also be expanded to IoT applications. In particular, we propose that devices do not come with any pre-implemented functionalities except the generic libraries to access their hardware and to execute scripts that will be dynamically received. In other words, each device serves as a generic platform, which is flexible enough to realize any

feasible scenario requested by its users after it has been installed. Each device would have to advertise its basic functionalities, such as "open window", "sense temperature", etc., over the wireless network. Further, it offers the functionality to wirelessly receive and execute code, to which a standardized Application Program Interface (API) for accessing the device's basic functionalities is offered. A more powerful device, such as a smartphone that is occasionally present in the network, synthesizes and distributes the code for new services whenever the user requests. In contrast to the configuration approaches of current IoT solutions (e.g., asking users to configure their IoT ecosystem using a web interface), advanced, more convenient input methods can be used. For example, our proposed system can integrate into existing natural language interfaces such as Apple's Siri. Using natural language interfaces, the user can e.g., specify that the window needs to be opened once the air quality decreases in an easy and convenient manner. The system may also provide a feedback on whether the synthesized service corresponds to the initial request or ask the user for missing information.

In this position paper, we propose a system to realize dynamic service synthesis in medical IoT systems. We outline the generic techniques towards this, which involves, i) deriving a formal specification from a user-friendly high-level specification (e.g., natural language), ii) synthesizing code accordingly, iii) transmitting the code wirelessly to the IoT device and iv) interpreting the code on the IoT device. What is notable is that we propose to adopt an interpreter for IoT devices for the sake of flexibility of functionality of each IoT device. Since all body-worn devices and also a large set of smart-home appliances are battery-powered, energy consumption is one of the main challenges in such devices. We show that the overhead due to the code interpretation on the embedded IoT platforms is acceptable. Further, we use the concept of *dynamic service switching* to further save energy. Since the intended scenario is known, data acquisition and processing can be tailored towards the use-cases needed. Therefore, unlike in static setups, the data-quality can be limited to the demands the scenario currently requires.

The technique described in this paper is based on the concept of Hybrid Apps, which we have presented in our previous work [5]. In addition, it is built upon Dynamic Service Switching, which we have proposed in [6]. Since Hybrid Apps and Dynamic Service Switching are combined and extended towards Dynamic Service Synthesis, we also reuse some parts of our previous work to illustrate these techniques. The rest of the paper is organized as follows. Section II describes the relevant work on the dynamic reconfiguration of IoT devices. Next, in Section III, we describe the hardware and software infrastructure and technical hurdles where this approach is useful. Section IV elaborates the concept of Dynamic Service Synthesis and Switching. Finally, Section V provides preliminary results and case studies, underlining the feasibility of our approach.

II. Related Work

Many research projects have already evaluated different end-to-end system design for AAL applications, i.e., from the sensing platforms to the data collecting servers. In [7], the authors describe the main concepts behind the *Keep in Touch* (KIT) and the *Closed Loop Healthcare* services. The former is based on RFID and NFC technologies to improve the interaction of patients with the smart environment, whereas the latter implements a back channel to remind patients to take their medicaments and to offer them access to their recorded medical data. The authors in [8] present an IoT-based healthcare system consisting of a personal embedded device to assist diabetes patients with the insulin dosage calculation. The system makes use of RFID cards and gives full connectivity to the personal devices through the 6LoWPAN protocol. Whereas these project are focused on offering a management infrastructure for caregivers and supporting patients with specific tasks, e.g., the insulin dosage calculation, our approach concentrates on methods with which patients become more independent of caregivers by defining dynamically their own custom tasks, which will support them with their activities of daily living.

In an important prior work [5], we have proposed a framework for reconfiguring smart home devices in order to run custom applications. Users establish a set of rules and behaviors with their smartphone, which creates code to be interpreted on the smart devices. This work forms part of the Dynamic Service Synthesis concept proposed in this paper in a sense that it also creates a code dynamically, transmits to the smart devices, and interprets the code for execution. In [6], we also presented a telemonitoring system for healthcare in which vital parameters measured by body sensors are transmitted to a server. It was proposed to automatically adjust the quality of the sensor signals according to predefined events in order to reduce energy consumption of the sensors. This idea is incorporated into the proposed framework as Dynamic Service Switching.

To address the problem of programming heterogeneous IoT environments, the authors in [9] presented an application-agnostic embedded infrastructure in which devices communicate their elementary functionalities to a cloud system using REST resources, similar as in our case. The authors in [10] proposed the concept of software-defined IoT units, which are sets of APIs which encapsulate IoT resources and capabilities for the unified development of IoT applications. However, in contrast with our work in which applications run on the IoT devices in a decentralized manner, in the cited works the IoT environment is remotely operated from a single application running on a server.

Regarding code interpretation on memory constrained embedded platforms, virtual machine based approaches have been proposed for wireless sensors as a mean to achieve reconfiguration of an already deployed network. In [11], [12], a subset of the Java language has been used to describe generic sensing applications. The Java source code is compiled offline into bytecode, which is sent to the sensors instead. Finally, another approach which is specially suitable for memory-constrained sensors, though less generic, is the use of application-specific virtual machines presented by [13], [14]. Whereas in all cases the virtual machines consisted of bytecode interpreters, our approach is based on direct C-code interpretation, which is suitable for any kind of applications and dispenses with any compilation process.

Finally, a method to extract formal specifications from a natural language requirement using an ontology was presented in [15], and a formal specification for an electronic system was extracted from a natural language in [16].

III. Medical IoT and Ambient Assisted Living

A. Example Scenario for Dynamic Service Synthesis

In this section, to illustrate our proposed system, we present an example scenario in which a subject interacts with medical and assisted living IoT devices.

We assume that a set of sensors and actuators are installed in the smart environment of a subject. As depicted in Fig. 2, a temperature sensor is installed to measure the room temperature. In addition, a CO_2-sensor can quantify the air quality. For influencing the room temperature and the air quality, the window is equipped with a motor to open and to close it. In addition, a smart heater is able to control the room temperature.

A large variety of different scenarios are possible in this environment, especially if further devices are within range. In our example, we assume the user is a subject with disabilities. For this subject, approaching the window to open it, or to go to the heater or air conditioner to adjust the temperature manually is very inconvenient. Therefore, the subject would like to control the temperature, humidity and air quality of his room is automatically. However, the automatic control must take into account multiple situations that have not been foreseen by the manufacturers of the devices. For example, the manufacturer of the electric window opener has not intended to combine it with a CO_2 sensor.

In a static system, simple threshold-based rules could be predefined by a technician e.g. on a central gateway. However, the system only increases the convenience of the user, if multiple more complex rules are kept. For example, such rules are:

- If the air quality decreases during the day, no automatic control should be executed, since the user prefers to use his remote-control to open the window manually instead. Hence, during the day, only a warning message should be sent to a mobile phone if the air quality decreases.
- During the night, the window should open automatically in the case of an insufficient air quality, if the outdoor temperature is not below a certain threshold.
- Once the air quality becomes high enough again, the window should close. The same should happen if the room temperature falls below a threshold.
- During all times, the heater should maintain a nearly constant air temperature. For a more convenient sleep, the temperature in the night should be by two degrees lower than during the day.
- If the subject is not at home (detected by either his wearables or his smartphone being not within the range of reception), the window should only open for a few minutes every day and the room temperature should be low to save heating energy.
- If the patient wears a oxygen sensor, the window should additionally open for a short period of time whenever the blood oxygen level decreases during the night.

Existing approaches lack of the flexibility to set up such systems by a layperson. With our proposed approach, the subject could specify the rules during e.g. a natural language interface. The smartphone would then synthesize code that realizes these rules. The code for each device is transmitted wirelessly to them. All devices jointly realize the scenario requested by the users. This demonstrates that such a system architecture would open a whole new world of applications, which make the everyday life more convenient.

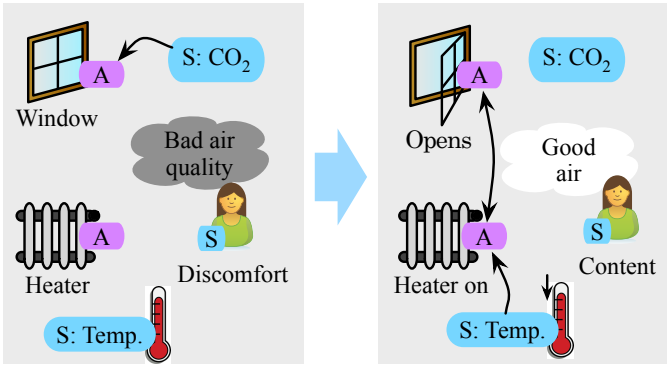

Fig. 2: How multiple IoT devices could collaborate to achieve the goal given by the patients.

B. Technical Hurdles

There exist a number of technical difficulties in order to make the described concept possible. Besides the difficulties in natural language recognition and synthesis of formal specifications from it, we focus on challenges from the perspective of the design of the IoT devices.

Power and performance constraints of the IoT devices: After code is generated and sent to the sensors, the next challenge consists of the computational limitations of the embedded devices, which hinders the execution of complex interpretable algorithms. Moreover, multiple tasks using same sensors simultaneously demand a tasks scheduling, which demands additional resources of the sensor. On the bright side, if resources like energy or memory are too limited in a certain moment, the scheduler can limit the execution of a certain task according to predefined priorities.

Coordinated control of the environment: After the instruction's syntax and the referred objects are verified, a controller can be generated. However, if multiple controllers share a physical space and objects, a coordinated design is necessary to avoid race conditions. In addition, there might coexist simultaneous system associated to the same sensors and actuators. For example, closing the window reduces temperature exchange with the exterior and also reduces the acoustic noise coming from the outside. Since a physical model of these phenomena is not available in advanced, synthesized controllers can be only generic and based on feedback. This imposes the need of associating a sensor to every actuator. However, generic controllers can be tuned with an on-line learning process to optimize control metrics like settling time or stability.

Heterogeneity of IoT devices: IoT devices will be manufactured from various vendors, and are likely to have heterogeneous hardware and software platforms. Synthesizing valid code for heterogeneous types of IoT devices on the cloud would be a demanding task. Among the potential solutions for resolving this problem, we believe the use of an interpreter would be a viable one. The interpreter will provide an abstraction layer, which relieves the burden of the cloud to cross-compile code for all sort of IoT devices.

Security and safety: Also security issues are present when a smart ecosystem has such reprogramming capabilities. Access control and encryption mechanisms are necessary to avoid malicious use of the sensors by attackers, which demand again additional memory resources. On the other hand, doctors,

physicians and caregivers will require access to the system to establish rules as well, which demands a management system where profiles and privileges are defined and accessible from all sensors in the system.

IV. IoT System Design for Medical and Ambient Assisted Living

A. Interpreters on IoT Devices

As mention before, we propose the use of code interpreters as mechanism to achieve hardware abstraction on the IoT devices. Some of the advantages of this approach are listed next:

Strengths of using Interpreters: One important advantage of interpreters is the *flexibility* to modify the sensor's functionality quickly. This is also a method to optimize memory use, since an unlimited amount of functionalities can be implemented on the device by replacing old code with new one. An another advantage is *specificity*. IoT devices can process sensing data more precisely according to its destiny, thus being more efficient by communicating this data. For example, a temperature sensor can be programmed to transmit daily maximal, minimal and average values if the data is for statistical purposes, or if the data is for controlling purposes, it can be programmed to communicate only when a significant difference in the temperature is detected. A third advantage of using code interpreters on an IoT device is *decentralization*, in which a task can be executed by multiple IoT devices without a central decision maker. This has the potential of creating a more robust system. A fourth benefit is the *disposable code* concept, in which temporary code can be generated and executed on the IoT device, e.g. for testing purposes. Once the test results have been generated and communicated by the code, this can be removed and the original functionality of the IoT device can be reestablished. Finally, another benefit is *hardware abstraction*, which permits generic code to be executed on heterogeneous platforms using a common API. The API also restricts the code of taking total control of the hardware.

Middleware layer: Low-level processes like those related to communication or hardware access are delegated to a hardcoded middleware layer in the IoT. This is responsible of interrupting the interpreter's execution to attend the arrival of new communication packets or read a new sample of the ADC, for example. By using a multitask-capable OS as middleware like FreeRTOS [17], several instances of the interpreter can be executed dynamically and operated as concurrent tasks, as shown in Fig. 3. In addition to benefits like concurrent and supervised execution of interpreted code, this also enables multiple users to allocate their custom functionalities on a single smart device. In this case, shared resources accessed through the API should be safely administered by the OS in order to avoid race conditions. Nevertheless, safe operation of an interpreted program can be achieved even without a sophisticated OS and can be supervised by a dedicated interrupt service routine (ISR) executed periodically, for example. In case of the detection of a prolonged execution of the program, or if any other operation rule is broken, the interpreter's state machine can be reset.

B. Code Generation and Transmission

Code generation: The process of synthesizing a service which will run on one or many IoT devices is described next. First, a user carries a internet-capable device such as a smartphone or a smartwatch, which is used to transmit his/her verbal instructions to a cloud service. This device is also used to recognize the surrounding devices using a shared communication protocol. Both, the instruction and the list of available devices are sent to the cloud as input for the synthesis process.

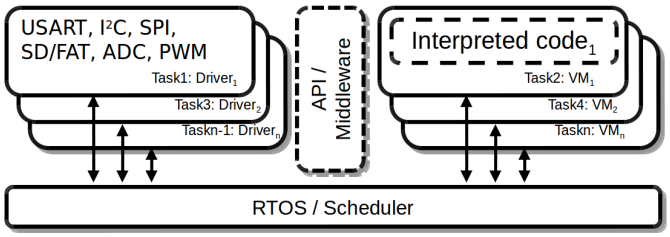

Fig. 3: Virtual platform inside of an IoT device.

Because of the heterogeneity of embedded platforms found in IoT devices, each device needs to carry a description table of its own capabilities or SpecSheet. This SpecSheet is shared with the smartphone in the moment it executes the scan, which is transmitted to the cloud synthesizer afterwards. The synthesizer uses this information to mark in its internal list of API commands which commands are available.

After verbal instructions are processed on the cloud, key parameters are extracted for the code generation process such as the associated devices and environmental parameters to be controlled. In this point is verified whether the described service can be synthesized using the detected surrounding IoT devices or not, for example. A pre-defined high-level model associated to the required service is used as template and filled with the provided information to create a state-machine which is formally verified and translated into code. Once the code is created, it is sent to the smartphone in order to be distributed among the involved IoT devices. The code is also buffered on the smartphone in case the same service is required in the future.

Code transmission: Once the custom functionality is synthesized as code, it is sent to the smart devices using a code transfer service over a wireless communication protocol supported by the smartphone, like Bluetooth (BT), Bluetooth Low Energy (BLE), WiFi or Near Field Communication (NFC). In addition, the smart device also provides a service for controlling a basic *hypervisor*. By using it, users can install code in a standardized fashion, remove installed services running on their devices or adjust access privileges and configuration options. This interface also allow both devices to negotiate a set of rules regarding aspects such as i) the period a program may remain installed on the device, ii) the events that trigger its execution (persistent execution, event-triggered execution, periodic execution, etc.), and iii) which and how many hardware resources may use the program.

C. IoT Device Power Optimization Techniques

In this section, we elaborate the concept of Service Switching, which was proposed in [6]. While Dynamic Service Synthesis refers to a holistic sequence of processes to implement a service, Service Switching is focused on individual sensors and can be applied orthogonally.

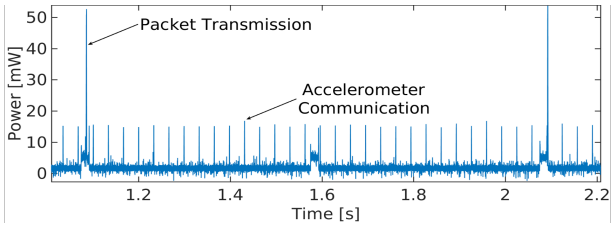

Fig. 4: Power consumption waveform of the acceleration sensor [6].

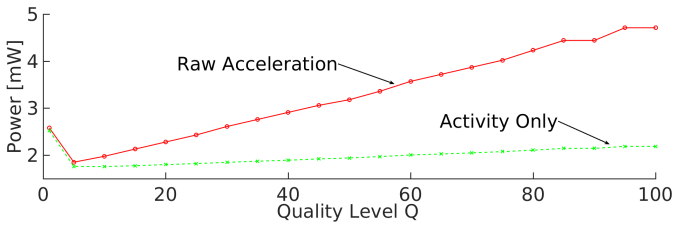

Fig. 5: Communication power consumption at different sampling rates and types of services [6].

As we show in Section V, the computational cost of interpreting some services can be too high for a given task, which makes the use of multiple variants of the same service beneficial. For example, one variant of an activity service on an acceleration sensor can be sending raw samples of an acceleration signal, whereas another more energy-efficient variant can be sending the associated effective value (RMS) of the same signal at a more slower rate.

In order to reduce the sensor's configuration space given the flexibility offered by the code interpreter, an additional abstraction layer called *channel* is introduced to simplify the interface with an external entity. Accordingly, each channel is presented to the outside as a virtual, independent sensor that offers a particular type of data, such as the RMS value of a signal or raw signal data. Additionally, 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.

Once created, each channel is assigned with a unique identifier (UUID) similar to ISO/IEEE 11073, necessary to interpret the data sent over the channel (viz., how to translate the bitstream into physical values). Based on the above assumptions, a Dynamic Service Switching performs the following two actions to adjust the service quality and its power consumption:

Switching on/off virtual channels: The data recorded by the sensors can be evaluated at different physical locations of the system, viz. other sensors, the smartphone or the cloud. Since the data can have multiple simultaneous destinations and have different purposes, only the necessary channels are switched on and off.

Adjusting the Measurement Quality: In addition, each data channel has a quality parameter $Q \in [0, 100]$ assigned to it. Its significance is specific to the channel and maps to concrete parameters according to the associated API description, such as sampling rate or resolution of a signal, sensitivity of a trigger, computation effort and others.

V. Case Studies: Sensor Quality Control and Power Consumption Analysis

As we have mentioned throughout the paper, code interpretation plays an important role for realizing the concept of Service Synthesis and Switching. In this section, we provide some preliminary experimental results on the feasibility of code interpretation on resource constrained sensors.

We analyze an exemplary scenario of the concept proposed in Section IV. An IoT device equipped with an activity sensor switches among four different services related to patient monitoring; raw data transmission (*Raw*), direct activity estimation (*Activity*), step counting (*StepCnt*), and fall detection (*FallDetect*). The algorithms are described in detail as follows: The *Raw* service transmits the raw samples collected from the signal. It requires no additional computation than reading the data from the sensors. The *Activity* service puts samples into a buffer of size $3 \cdot f_s$, where f_s is the sampling frequency. When the buffer is full, the RMS value of the samples is calculated. Then, an activity level from 0 (no activity) to 4 (the most activity) is determined and transmitted to a gateway. If the level is the same as the previous one, no data is transmitted. In the *StepCnt* service, a basic step counting function is implemented. An infinite impulse response (IIR) high-pass filter is implemented, and its output is compared to a threshold which triggers a step event. A value is transmitted accordingly and the analysis is resumed after a certain period. *FallDetect* implements a basic fall detection algorithm. It makes use of *Activity* and checks whether there is a high activity level followed by a state of no activity. We show that different services have different effects on the power consumption of the IoT devices and the quality of the patient's activity estimation.

The IoT device we have used consists of three main elements: an acceleration/inertial MEMS module, an ARM-Cortex M4 Microcontroller Unit (MCU), and a C-code interpreter running on the MCU. The MCU is connected to a PC using a USB serial link, which is used to send the script code to the interpreter.

A. Energy consumption due to communication

First, we investigate the energy consumption due to communication in the IoT device executing each service. We refer to one of our recent works in addressing the communication energy consumption [6]. A typical waveform of power consumption of a BLE-based IoT device equipped with an ARM Cortex M0 MCU and a 3-axis acceleration sensor is shown in Fig. 4. The figure shows two types of events. The first is the sampling of the acceleration sensor, and the second is the radio communication with the gateway. As observed in the figure, the IoT device periodically polls the accelerometer module at f_s, and periodically sends a packet to the gateway (sometimes an empty packet) to keep the BLE connection alive.

The energy consumption of the IoT device at different sensor sampling frequencies -and hence signal qualities- was measured and shown in Fig. 5. The red line shows the power consumption of the *raw* service, which increases as the sampling rate increases (Quality Level Q). This is because the sampling increase leads to an increase of the communication packets and the communication energy. The green line shows the power consumption of an (compiled) *activity* service, instead of the raw service. Because the amount of bytes transmitted to the gateway is significantly reduced, the

TABLE I: Power consumption of testing algorithms at different sampling frequencies measured on ARM Cortex M4.

	$f_{exec} = 10Hz$	$f_{exec} = 50Hz$	$f_{exec} = 100Hz$
Activity	6.3 mW	6.9 mW	7.8 mW
StepCnt	6.3 mW	7.8 mW	9.6 mW
FallDetect	6.3 mW	7.0 mW	7.8 mW
SleepPwr : 5.9 mW			

total communication power is accordingly reduced. The energy consumption by polling the acceleration sensor is the same between *raw* and *activity*. It can be observed that, in the case of sampling at 100 Hz, the energy consumption of the sensor can be halved by processing the signal on the sensor instead of sending it as raw samples. However, we note that this addresses the communication power consumption only, and the additional power consumption overhead for processing the raw data with a code interpreter should be considered as well, which will be presented in the following section.

B. Overhead of code interpretation

In this section, we investigate the performance and energy overhead due to code interpretation. For executing the synthesized service on the IoT device, a light-weight but yet full-featured code interpreter called PicoC [18] was run on a platform based on an ARM Cortex M4 MCU. The device periodically wakes up from sleep mode to transmit activity data to a gateway. From previous studies, it is known that code interpreters raise a large overhead to the execution times of the code [5]. The overhead results in longer active phases of the CPU, thereby consuming additional amounts of energy. Table I shows the mean power overhead for the algorithms described at the beginning of this section at different execution frequencies f_{exec}. *StepCnt* consumes considerably more power than other services as it involves an IIR filter, which executes floating point operations. The results show that the energy consumption is dominated by the sleep power consumption in case of low f_{exec}, in which case the energy overhead due to interpretation is only 6.3%. Even for the case of f_{exec} =100 Hz, which is already quite high for detecting human activities, the power consumption overhead of interpretation is in average 29%, which is acceptable.

In addition, we investigate the hardware resource requirements of interpreting the described algorithms, which are shown in Table II. The ROM and initial RAM requirements of the interpreter are shown in the section "Initialization" in the table. The wake up time associated to leaving the sleep mode is shown under "Wakeup", and can be considered execution overhead. In the section "Programs" of the table, the size of each script, the minimal amount of RAM ($\mathrm{memVM_{min}}$) required by PicoC to execute the script, the maximal stack size ($\mathrm{Stack_{max}}$) observed on the MCU during the script execution, and the execution time (Exec.) of each script is shown. We can see that the RAM requirements are maximal 13.2kB, and the execution time of the programs allow these to be executed as frequently as 1.6kHz in the worst case.

TABLE II: Computational requirements of code interpretation

	Code [Byte]	RAM [Byte] .bss / $\mathrm{Stack_{max}}$	Wakeup [ms]
Initialization	.data: 3K .text: 48K	2K+memVM / 920	0.19
Programs	**Code [Byte]**	**RAM [Byte] $\mathrm{memVM_{min}}$/$\mathrm{Stack_{max}}$**	**Exec. [ms]**
Activity	439	8.1K / 1.5K	0.31
StepCnt	250	7.5K / 1.3K	0.62
FallDetect	755	9.2K / 2.0K	0.31

VI. Concluding Remarks

In this positioning paper, we proposed a Dynamic Service Synthesis and Switching framework for medical IoT and ambient assisted living. The framework allows a patient or a person in need of assistance to provide a natural language specification of a desired service, and automatically synthesizes the service to be implemented on various IoT devices in the environment. We provided an example application of the framework and addressed the technical hurdles in realizing it. In addition, we proposed code interpreters as reconfigurability mechanism for the IoT devices, which provided a hardware abstraction layer to the service synthesis process. The preliminary measurement results show the feasibility of the proposed framework. Finally, we anticipate this work will provide a viable outlook on medical IoT devices and expedite its wide adoption.

References

[1] A. Tobola *et al.*, "Scalable ECG hardware and algorithms for extended runtime of wearable sensors," in *IEEE Int. Symp. on Medical Measurements and Applications (MeMeA)*, 2015.

[2] Y. Ren, C. Wang, J. Yang, and Y. Chen, "Fine-grained sleep monitoring: Hearing your breathing with smartphones," in *IEEE Conf. on Computer Comm. (INFOCOM)*, 2015.

[3] J. Klucken *et al.*, "Unbiased and mobile gait analysis detects motor impairment in parkinson's disease," *PLoS One*, 2013.

[4] R. H. Weber, "Internet of things new security and privacy challenges," *Computer Law & Security Review*, 2010.

[5] D. Yunge, P. Kindt, M. Balszun, and S. Chakraborty, "Hybrid apps: Apps for the internet of things," in *2015 IEEE 12th Int. Conf. on Embedded Software and Systems (ICESS)*, 2015.

[6] P. Kindt, D. Yunge, A. Tobola, G. Fischer, and S. Chakraborty, "Dynamic service switching for the medical iot," in *Personal, Indoor, and Mobile Radio Comm. (PIMRC), IEEE 26th Annual Int. Symp. on*, 2016.

[7] A. Dohr, R. Modre-Osprian, M. Drobics, D. Hayn, and G. Schreier, "The internet of things for ambient assisted living." *ITNG*, 2010.

[8] A. J. Jara, M. A. Zamora, and A. F. Skarmeta, "An internet of things–based personal device for diabetes therapy management in ambient assisted living (aal)," *Personal and Ubiquitous Computing*, 2011.

[9] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 6th Int. Conf. on*, 2012.

[10] S. Nastic, S. Sehic, D. H. Le, H. L. Truong, and S. Dustdar, "Provisioning software-defined iot cloud systems," in *Future Internet of Things and Cloud (FiCloud), 2014 Int. Conf. on*, 2014.

[11] N. Brouwers, K. Langendoen, and P. Corke, "Darjeeling, a feature-rich vm for the resource poor," in *Proc. of the 7th ACM Conf. on Embedded Networked Sensor Systems*. New York, USA: ACM, 2009.

[12] D. Simon *et al.*, "Java™on the bare metal of wireless sensor devices: The squawk java virtual machine," in *Proc. of the 2nd Int. Conf. on Virtual Execution Environments*, 2006.

[13] P. Levis, D. Gay, and D. Culler, "Active sensor networks," in *Proc. of the 2nd Conf. on Symposium on Networked Systems Design & Implementation*, 2005.

[14] R. Mueller *et al.*, "A dynamic and flexible sensor network platform," in *Proc. of the 2007 ACM SIGMOD Int. Conf. on Management of Data*. New York, NY, USA: ACM, 2007.

[15] D. Sadoun, C. Dubois, Y. Ghamri-Doudane, and B. Grau, "From natural language requirements to formal specification using an ontology," in *IEEE 25th Int. Conf. on Tools with Artificial Intelligence*, 2013.

[16] R. Drechsler, M. Soeken, and R. Wille, "Formal specification level: Towards verification-driven design based on natural language processing," in *Specification and Design Languages (FDL), Forum on*, 2012.

[17] Freertos homepage. [Online]. Available: www.freertos.org

[18] Z. Saleeba. Picoc project homepage. [Online]. Available: code.google.com/p/picoc