Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 210 (2022) 112–115

Procedia Computer Science

www.elsevier.com/locate/procedia

The 12th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2022)
October 26-28, 2022, Leuven, Belgium

Towards a health software supporting platform for wearable devices

Maxmilian Otta[a,*]

[a]*Department of Computer Science and Engineering, University of West Bohemia, Technicka 8, 306 14 Plzen, Czech Republic*

Abstract

The number of broadly available wearable devices like smart watches or fitness bands keeps growing, as well as their performance and number of provided features related to user's health. This was the reason for our decision to bring the SmartCGMS (Smart Continuous Glucose Monitoring and Controlling System) to a wearable device, in order to foster its way to practical deployment in healthcare. Currently, the SmartCGMS system is able to run on Windows, macOS, Linux, RaspberryPi or Android phones and tablets. What is currently hindering us to run SmartCGMS on a wearable device is the heterogeneity of devices and primarily lack of real-time OS features available to developers. This is natural, because device vendors aim for best user experience and so foreground tasks get the highest priority in order to maximize device responsiveness and suppress background tasks for minimum CPU load and battery drain. But running medical software on a wearable device requires almost the opposite – tasks reading sensor data and especially tasks controlling drug dosage, that are running on the wearable, require high priority and minimum interference with other running tasks. In this article, we present an initial design of a software framework in development, that will provide us the features we are currently missing in available wearable devices operating systems: a common application image that is able to run on various wearable devices and support for high priority tasks.

Keywords: healthcare IoT; wearable devices; SmartCGMS; FreeRTOS;

1. Introduction

The SmartCGMS (Smart Continuous Glucose Monitoring and Controlling System) [1] developed at the Department of Computer Science and Engineering, University of West Bohemia, is a highly modular framework (implemented in the C++ language) which has a variety of use cases: monitoring the blood glucose level when connected to a sensor, generating output control for an insulin pump based either on externally provided data, data measured by a sensor or both. It even can be used in a simulated environment fed with real data collected from a patient's sensor over a certain time-period. Currently, the SmartCGMS runs on small devices like Raspberry Pi or Android based

* Corresponding author. Tel.: +420 377 632 476; fax: +420 377 632 402.
E-mail address: maxmilio@kiv.zcu.cz

smartphones, but these devices have major limitations and disadvantages in order to be seriously used in healthcare. The Raspberry Pi can't be considered a wearable device because of its size and dependency on external power supply. Smartphones are from this point of view more suitable, but they are primarily used for communication and, especially by young people, for media consumption or gaming, resulting in high CPU usage and consequently fast battery drain. And finally, it is very easy to forget the smartphone somewhere. Thus, our idea was to bring SmartCGMS to a wearable device like a smart watch and, if the proposed concept works, probably to other medical IoT devices [2]. In the future, the system will be supported by a secure [4] framework continuous software delivery to Health IoT devices [3] in general and cloud-assisted health monitoring [5]. For the start, we have chosen the LilyGo T-Watch which is a cheap (approx. 40 USD or 37 EUR) smart watch based on the ESP32 system on chip (SoC). The reason for this choice was that ESP32 is a low-cost, low-power SoC dual-core microcontroller with many features we needed like 802.11b/g/n WiFi, Bluetooth v4.2 BR/EDR and Bluetooth Low Energy (important for the communication with the blood sugar sensor and the insulin pump) and power management with an ultra-low power coprocessor supporting several sleep modes. The watch itself has a 1.54 inch LCD capacitive touch screen, an RTC clock module, a 3-axis accelerometer, a vibration motor and a speaker. Another advantage of the ESP32 SoC is, that there's a plethora of available libraries and open-source software, so we can achieve high software leverage and focus only on new challenges of bringing SmartCGMS to a wearable device.

2. Initial implementation

As the base operating system for the wearable device we use FreeRTOS [13] - the real-time operating system for microcontrollers which is nowadays the de facto standard operating system for microcontrollers in IoT devices. There are many flavors of FreeRTOS, but there are two ones worth to mention: the FreeRTOS provided by Amazon Web Services (AWS), which includes lots of libraries for easy integration with Amazon's services and the FreeRTOS provided directly by Espresiff (the ESP32 SoC manufacturer) as a key part of the Espressif IoT Development Framework (IDF). While FreeRTOS offers basic OS features like task priorities, support for memory protection ("sandboxing" by using unprivileged tasks), task communication (stream and message buffers, queues) and synchronization (semaphores and mutexes), the IDF provides functionality we will need like Bluetooth Low Energy API, Wi-Fi API, TCP/IP, MQTT, HTTP(S) and TLS protocols and even an API for firmware updates over HTTPS.

The overall concept of the wearable device framework initial implementation is visible in the Figure 1. The framework is able to install (or remove) and run a small application on the device. Such a small application we call a feature and it runs as an isolated task with access to a given part of the flash memory. The main components of the framework are the communication module, the feature manager and the update manager. Further, the features are deployed as micro-containers and can be remotely installed and configured on the device. We propose a micro-container - a tiny software package consisting of a descriptor, based on a manifest file according to RFC-9019 [14] and compiled code. The compiled code will not be a native binary code of the target microcontroller, but WebAssembly code (WASM) run by an interpreter, so in fact, there will a lightweight virtual machine integrated into the framework. The idea running a WASM virtual machine on a wearable device starts to be reasonable these days, as it was already presented in several papers like [6], [7], [8] or [9]. We choose this approach, because then we are able to achieve smooth software flow (liquid software) [9] without having to cope with the infrastructure heterogeneity and use the SmartCGMS advantages at their full power. Currently, there's an open-source WebAssembly interpreter available at GitHub [15] which can be run on an ESP32 microcontroller.

The aim of the initial implementation is to verify how the described idea will work on a wearable device with limited resources. The ESP32 system has an Ultra Low Power (ULP) processor which supports several sleep modes by turning off parts of the SoC and thus saving power. The ULP processor is also capable to wake up the ESP32 subsystems when a given event occurs. This is implemented via programmable interrupts, that can be triggered by an RTC timer or by a peripheral (or sensor) connected to a GPIO pin. This feature enables us to run the features in the framework's container only when they are needed - at regular time intervals or when a sensor sends data.

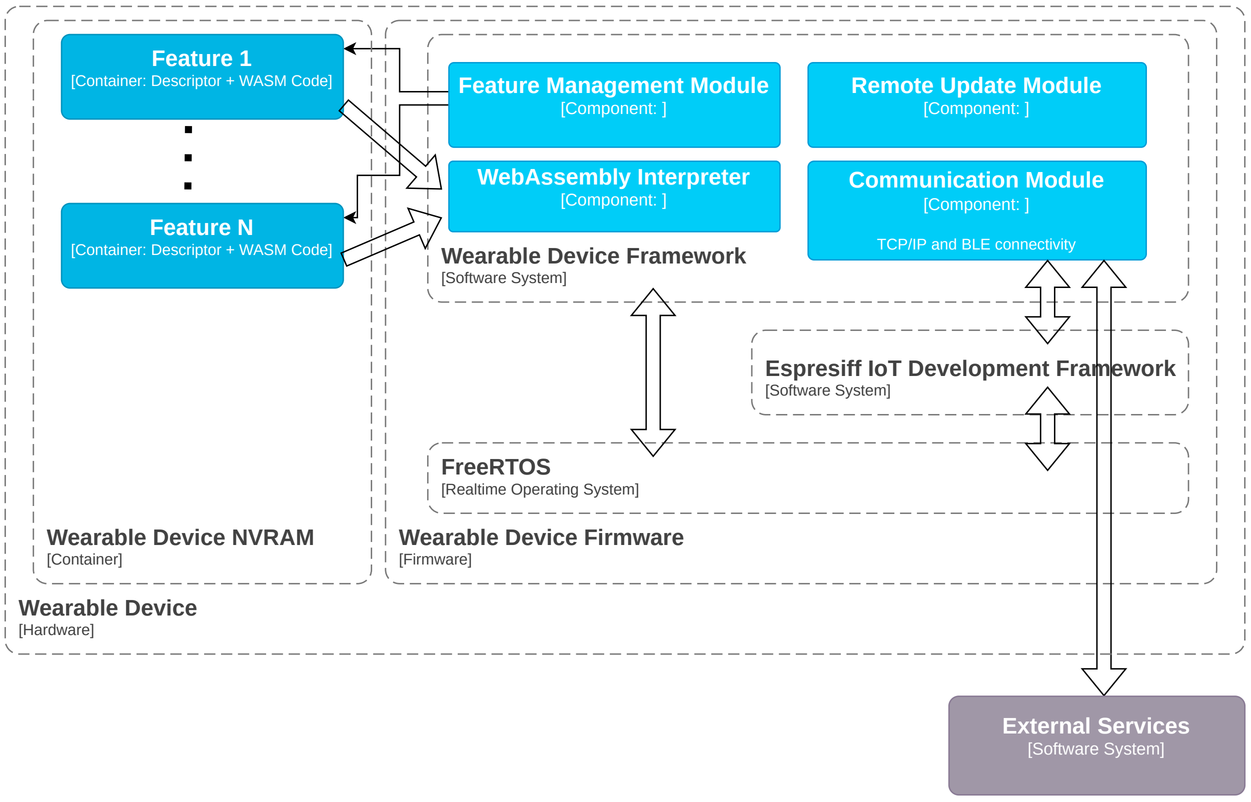

Fig. 1. Wearable device framework architecture

3. Related work

At the time of writing this article, we found the only similar idea presented in [9]. The goal of the authors is to achieve isomorphic IoT system architectures, so all devices can use and run the same software components unaltered, so enabling smooth code delivery and migration. This is just one pillar of our planned framework. We further need secure and reliable software delivery and updates, a micro-container technology for easy feature management and an easy and smooth device onboarding system. Another interesting platform is presented in [10], which doesn't consider wearable devices, but includes definition of services which apply also to platforms with wearable devices.

When we compare our approach with the ones in the related work, our goal is not to provide a proof of concept of a particular technology or a method. Our goal is to provide a platform, that would speed-up the deployment of medical devices and particularly the deployment of software (developed by our team) to real world environment in order to get feedback and, of course, the test results.

4. Conclusion and future work

The next step in our work, as already stated in the section above, is implementation of a device onboarding and management system. Also, if we want to go beyond the ESP32 based wearable platform, we will need a solution for handling various device APIs. Concerning the communication with medical devices, we will of course lean on the CEN ISO/IEEE 11073 Health informatics - Medical/health device communication standards.

Our vision is, that a physician can take an "off-the-shelf" feature with a general model from a repository, personalize it and verify in a simulator and then send it to the patient's wearable device, like it is shown in Figure 2. Moreover, in order to allow a physician (or a domain expert in general) to perform more sophisticated customization of wearable device features, we can use a special designed DSL for this purpose. The domain expert would just describe the

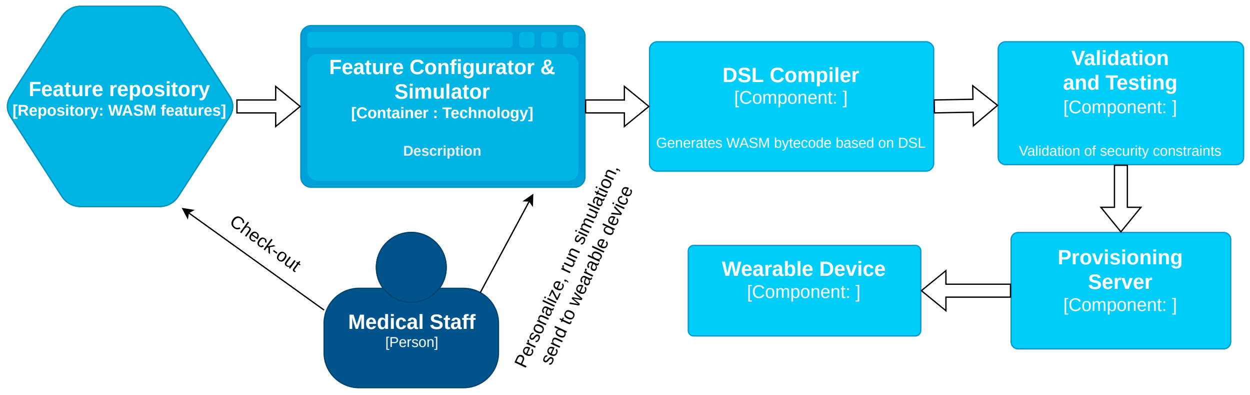

Fig. 2. Wearable device feature flow

customization using the DSL and a generator module would generate the WebAssembly code based on the DSL. The idea of involving domain experts into the software development cycle was already successfully employed at Siemens in the continuous software delivery pipeline (CICD) of computer tomography software [12].

References

[1] Koutny, T. & Ubl, M. Parallel software architecture for the next generation of glucose monitoring. *Procedia Computer Science*. **141** pp. 279-286 (2018)

[2] Verma, D., Singh, K., Yadav, A., Nayak, V., Singh, J., Solanki, P. & Singh, R. Internet of things (IoT) in nano-integrated wearable biosensor devices for healthcare applications. *Biosensors And Bioelectronics: X*. **11** pp. 100-153 (2022)

[3] Haghi Kashani, M., Madanipour, M., Nikravan, M., Asghari, P. & Mahdipour, E. A systematic review of IoT in healthcare: Applications, techniques, and trends. *Journal Of Network And Computer Applications*. **192** pp. 103-164 (2021)

[4] Sengupta, J., Ruj, S. & Das Bit, S. A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT. *Journal Of Network And Computer Applications*. **149** pp. 102-481 (2020)

[5] Hossain, M. & Muhammad, G. Cloud-assisted Industrial Internet of Things (IIoT) – Enabled framework for health monitoring. *Computer Networks*. **101** pp. 192-202 (2016)

[6] Jacobsson, M. & Wåhslén, J. Virtual machine execution for wearables based on WebAssembly. *BODYNETS 2018: 13th EAI International Conference on Body Area Networks*. pp. 381-389 (2018)

[7] Wen, E. & Weber, G. Wasmachine: Bring IoT up to Speed with A WebAssembly OS. *2020 IEEE International Conference On Pervasive Computing And Communications Workshops (PerCom Workshops)*. pp. 1-4 (2020)

[8] Jacobsson, M. & Willén, J. Virtual Machine Execution for Wearables Based on WebAssembly. *13th EAI International Conference On Body Area Networks* . pp. 381-389 (2020)

[9] Mäkitalo, N., Mikkonen, T., Pautasso, C., Bankowski, V., Daubaris, P., Mikkola, R. & Beletski, O. WebAssembly Modules as Lightweight Containers for Liquid IoT Applications. *ICWE*. (2021)

[10] Morabito, R. A performance evaluation of container technologies on Internet of Things devices. *2016 IEEE Conference On Computer Communications Workshops (INFOCOM WKSHPS)*. pp. 999-1000 (2016)

[11] Laukkarinen, T., Kuusinen, K. & Mikkonen, T. DevOps in Regulated Software Development: Case Medical Devices. *2017 IEEE/ACM 39th International Conference On Software Engineering: New Ideas And Emerging Technologies Results Track (ICSE-NIER)*. pp. 15-18 (2017)

[12] Nehls, H. & Ratiu, D. Towards Continuous Delivery for Domain Experts: Using MDE to Integrate Non-Programmers into a Software Delivery Pipeline. *ACM/IEEE 22nd International Conference On Model Driven Engineering Languages And Systems Companion*. pp. 598-604 (2019)

[13] Ibrahim, D. Using the FreeRTOS Multitasking Kernel *Arm-Based Microcontroller Multitasking Projects*. pp. 109-116 (2021)

[14] Moran, B., Tschofenig, H., Brown, D. & Meriac, M. RFC-9019: A Firmware Update Architecture for Internet of Things. (IETF, 2021)

[15] WASM3 - A Fast WebAssembly Interpreter and Universal Runtime. *GitHub*., ⟨https://github.com/wasm3/wasm3⟩, 2022 (accessed 10.07.2022)