24th IMEKO TC4 International Symposium
22nd International Workshop on ADC and DAC Modelling and Testing
IMEKO TC-4 2020
Palermo, Italy, September 14-16, 2020

Testing Wearable IoT Applications through Sensor Virtualization

Salvatore Gaglio[1,2], Giuseppe Lo Re[1], Daniele Peri[1], Riccardo Rizzo[1], Claudio Sorrenti[1]

[1]*University of Palermo, Department of Engineering, viale delle Scienze, Ed. 6, 90128, Palermo, Italy*
salvatore.gaglio@unipa.it, giuseppe.lore@unipa.it, daniele.peri@unipa.it,
riccardo.rizzo07@community.unipa.it, claudio.sorrenti@community.unipa.it
[2]*ICAR-CNR, 90146 Palermo, Italy*

***Abstract* – The development of distributed IoT applications requires the integration of data provided by different sensors embedded in multiple devices.**

As an example, an application for health monitoring in an assisted living scenario may include several wearable and fixed nodes each carrying different sensors and running specific code. Verifying that the application is properly working according to the specifications requires assessing that the code of each node behaves consistently in all the possible use cases. Tests involving sensor data may be difficult or costly to replicate realistically and this could also slow down the development of the application in its early stages.

In this paper we introduce a tool that allows developers of IoT distributed applications to test the interoperability of code running on heterogeneous IoT devices through sensor virtualization.

We show the feasibility of the approach in a case study of an application involving a wearable device and a single-board computer connected through Bluetooth Low Energy.

I. INTRODUCTION

The increasingly diffusion of Body Area Networks (BAN), Wireless Sensor and Actuators Networks (WSAN), and the Internet of Things (IoT) paves the way to the deployment of sophisticated Health Monitoring or Assisted Living applications [1, 2, 3]. Development and test of distributed wireless applications involves several steps and simulators are often used to reduce the testing efforts. However, simulations usually only verify the executable code running on virtual environments without including the evaluating of actual hardware aspects, let alone of sensors, in the global behavior of the application under test [4, 5, 6]. For example, testing of home assistant systems involves many devices and sensors of different types [7]. This also applies to other relevant IoT applications such as home-energy monitoring and management [8]. An approach based on the exchange of executable code (DC4CD) has proven effective in dealing with many interconnected heterogeneous devices [9], in the development of distributed applications. High-level formalisms such as fuzzy logic expressions can extend the expressivity of the code that is exchanged among the nodes [10] and executed even on resource-constrained ones [11]. This approach also permits fast interactive development and test on deployed nodes without either cross-compilation or flashing of the code [12], and it also makes possible to model and test the very same hardware on which the code runs [13]. With the aim to extend the symbolic approach to also test the sensor-related aspects of distributed applications, in this paper we introduce the virtualization of sensors in heterogeneous networks. Briefly, virtualization consists in using samples from dataset instead of real samples from sensors so to test consistently both local and distributed application code against common test data. This way, besides testing the application correctness, also comparisons with other similar applications in terms of performance can be carried out. In the remainder of the paper we first describe the development and test system, then we show the feasibility of the approach in a case study of an application involving a wearable device and a single-board computer connected through Bluetooth Low Energy (BLE). Finally, we draw our conclusions and describe the related future work.

II. SYSTEM DESCRIPTION

The development and test system mimics the structure of a real deployable distributed application for health monitoring or assisted living. Both fixed and wearable devices are thus considered. For our tests we used one fixed device and one wearable device.

A. *System components*

The fixed device is a RaspberryPi 3 B+ [14], provided with 1 GB of RAM and a 64 bit quad core processor clocked at 1.4 GHz, with a micro-SD card as mass storage medium, a WiFi module, and a Bluetooth Low Energy (BLE) module. The latter will be used to communicate with the wearable device.
The wearable device is a Sensortile (STEVAL-STLCS01V1) [15] by STMicroelectronics, which includes a BLE module (BlueNRG-MS), an ARM

Cortex-M4 32 bit 80 MHz microcontroller with very low power consumption, a 128 KB RAM and a 1 MB of Flash memory, an LSM6DSM (3D accelerometer and 3D gyroscope).Other on-board sensors, such as LSM6DSM (3D accelerometer and 3D gyroscope), LSM303AGR (ultra low power 3D accelerometer and 3D magnetometer) and LPS22HB (MEMS nano pressure sensor), are available and will be considered for virtualization in future work.

B. Communication

The two devices communicate via BLE. This technology is based on a protocol stack. In particular, one of the top layer protocol is the Generic Attribute Profile (GATT) [16, 17], which is based on the concepts of services and characteristics.
A characteristic may be defined as a container for a value and, optionally, one or more descriptors. A descriptor is an attribute conveying about the value of the characteristic. A collection of characteristics makes up a service. Each entities is associated with a Universally Unique Identifier (UUID). GATT specifications include many standard characteristics to deal with data from common activities such as walking, running, cycling, fitness, personal data such as name, age, date of birth, health monitoring data, such as heart rate, and device details. Characteristics define generally values that are 20-byte wide.

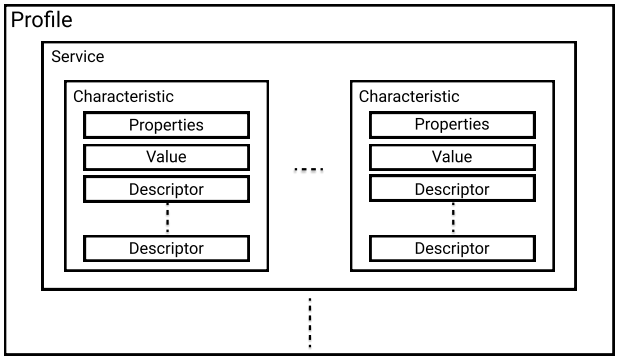

Fig. 1. GATT profile hierarchy

Characteristics can have one or more properties such as READ, WRITE and NOTIFY. While the first and the second are self-explanatory, the third allows to connected device to read the characteristic value whenever it changes. Custom characteristics are also supported by the GATT protocol. In fact, because no standard characteristics suitable for our application were available, we defined two custom characteristics. These were added to a custom service for general purpose use in the program running on the Sensortile. The first characteristic (`VirtualSampleIn`) is written to by the fixed device to send a virtual sample to the wearable device, the second (`VirtualSampleOut`) is written to by the wearable device to send back data to the fixed device. These characteristics have the same properties discussed above. The NOTIFY property of the `VirtualSampleIn` characteristic is used to command the wearable device to activate the sequential reading of data sent from the fixed device while the notify property of the `VirtualSampleOut` characteristic is used to trigger reading of the reply messages sent by the wearable device (Fig. 2).

C. System operation

The fixed device uses the Noble library, a Nodejs BLE Central Module [18]. First, the program connects with the wearable device via BLE. After the connection, it starts the exploration of the services and characteristics of the wearable device, each identified by its UUID.
In this work the sample rate is defined by the fixed device that tries to follow the sample rate of the dataset. The program executes two loops. The internal loop is used to write each sample of the session recording in the `VirtualSampleIn` characteristic. The external one, repeats the process a given number of times. Each sample includes a timestamp with the elapsed time (ms) (Fig. 3). On the receiving side the program takes care of subscribing to the notifications of the other characteristic, used by the wearable device to send its virtual data. When the wearable device needs to send a sample it changes the value of the `VirtualSampleOut` characteristic, then a notification reaches the fixed device where a callback function is executed. This function receives the data along with time information.

Similarly, the wearable device waits for a notification of the arrival of a new sample from the the fixed device. Then it feeds the processing waiting for a sample, for instance a fall detection algorithm, as if it were provided by an on-board sensor. In the system implementation shown in this work the link between the virtual sensor and the processing code is resolved at compile time. Using the aforementioned symbolic distributed processing platform (DC4CD) it can be established at runtime on deployed nodes. The processing code then proceeds normally, in the case of the fall detection algorithm by silently waiting or notifying the fixed device of a fall event through a specific characteristic.

III. VIRTUALIZATION OF AN ACCELEROMETER

As a case study we show how our approach can be used to test a fall detection application in an assisted living scenario. In this case, both the fall detection algorithm on the wearable node and the response from the fixed device must be considered. In this paper we deal with the first. To provide data to the virtual sensor we used the Sisfall dataset [20]. This dataset collects data recorded by a device worn at the waist in tests involving 38 people, 23 in ages 19-30 and 15 in ages 60-75. In particular, it includes 19 different Activities of Daily Living (ADL) and 15 different fall types. Each test collects data from two accelerometers

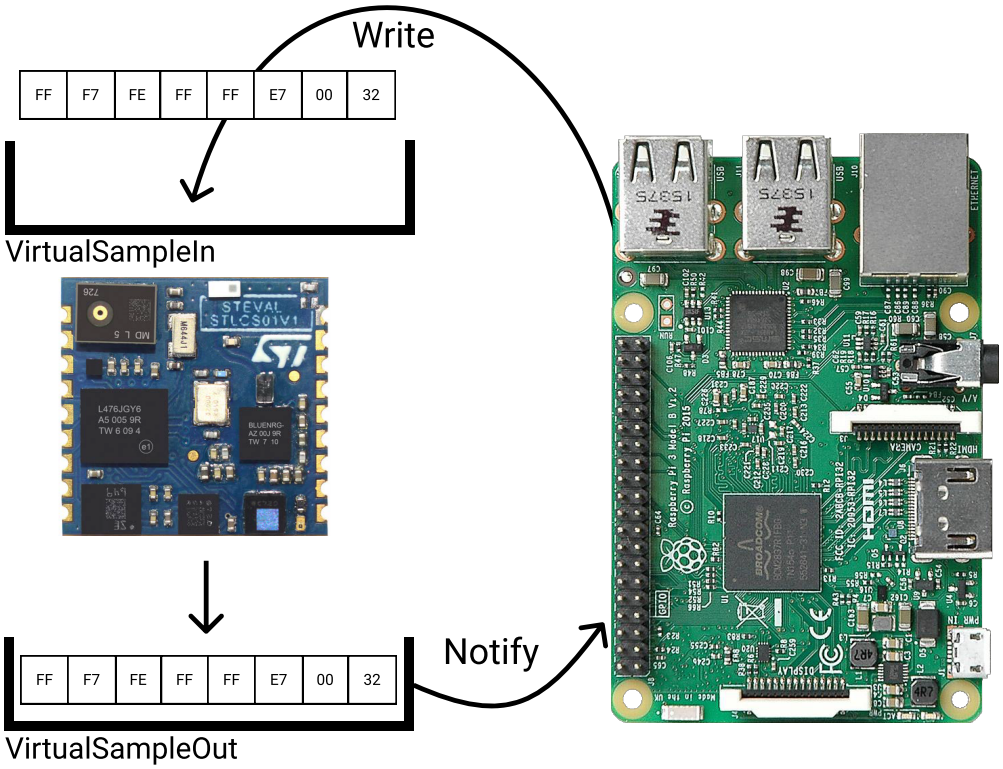

Fig. 2. Communication schema

(ADXL345 and MMA8451Q) and a gyroscope (ITG3200), all sampled at 200 Hz [19].

In our experiments we virtualized the ADXL345 accelerometer using data from one of the fall recordings. From the SisFall specifications, this sensor was set to a resolution of 13 bits and a range of ± 16 g for all the acquisitions. Each sample contains a value for each of the three axes of the accelerometer.

As described in the previous section, in a virtual experimental session the fixed device sends a sensor recording of a real session to the wearable device one sample at a time. For this tests the receiving device was provided with code to send back each sample to the fixed device, as in a continuous tracking application. All the fall recordings include 3000 samples collected at 200 Hz for a duration of about 15 s. Actually, as the first sample is sent at time zero, the duration of a recording is exactly 14.995 s.

To improve the accuracy of the assessment, taking into account all the processing not strictly involved in the virtualization such as connection delay, latency of the interpreter at launch, and general overhead of the underlying OS, we performed a test with increasing number of repetitions of a virtualized session.

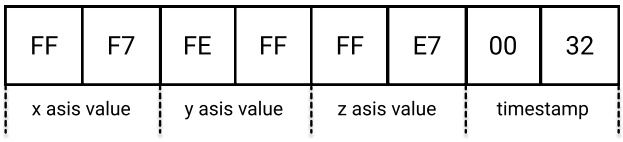

Fig. 3. Data structure example

The plot in Fig. 4 shows the mean duration (y-axis) of a virtualized session of n repetitions (x-axis) as measured on the fixed device. The value starts very close to the ideal duration for less than 10 repetitions then tends asymptotically to a value slightly larger than the real session duration as the underlying OS compensated for the relatively high sustained I/O load of the process. The plot thus provides also realistic estimates of the minimum and maximum variations in the duration of a virtualized session.

The whole assessment results, including mean, variance, and round-trip time (RTT) as measured by both the fixed and the wearable devices are collected in Table 1.

The low value of variance confirms that each test does not differ much from mean value. The difference from the ideal case is of 0.65 s. This difference could be explained by the processing load between sample creation and sending. In each sample sending cycle, a buffer is allocated and filled with dataset data and time information. All of this is repeated for each sample.

On the Sensortile side, which benefits of bare metal code exploiting a hardware timer and interrupts with no

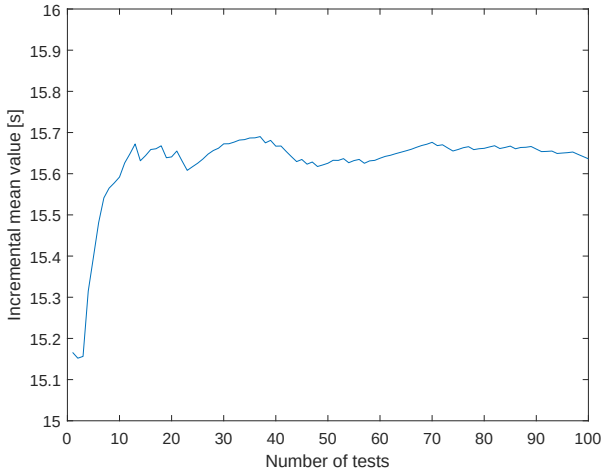

Fig. 4. Raspberry incremental mean value.

overhead from a non-realtime general-purpose operating system as in the case of the fixed system, the mean value of the reception time was calculated measuring the total reception time elapsed after 100 repetitions, that is 1509.908 s. This measurement was made starting a timer as soon as the first data packet of the first virtualized session arrives and stopping it on the arrival of the last data packet of the last session. The mean value is approximately 15.1 s, so just 0.105 s more than the ideal value. Variance is also rather low.

Table 1. Experimental results

	RaspberryPi	SensorTile
μ [s] (Mean value)	15.64	15.1
σ^2 (Variance)	0.11	$6.17 \cdot 10^{-5}$
Mean RTT [ms]	24.64	

Another assessment is about the time it takes a sample to make a round trip from the fixed device to the wearable, and then back to the fixed device. The mean RTT considering all the samples was 24.64 ms.

IV. CONCLUSIONS

Results show that a virtualized sensing session can closely match a real one with respect to time constraints. Even though the sampling rate of the virtual sensor was dependent on the ability of the fixed device to keep up with the sample rate of the dataset, the overall differences with the timings of the recordings were marginal on a wide range of workloads in a non-realtime environment. This bodes well for the extension of the approach with wearable-based synchronization. The real-time capabilities of the latter are, in fact, expected to improve the adherence of virtualized sessions to the dataset. Moreover, the low RTT that was measured suggests that more intensive processing on both the fixed and the wearable device could be implemented without impacting the effectiveness of virtualization. The approach can be also extended to other types of sensor and also to hybrid scenarios including both real and virtual sensors. Future work will extend the range and number of virtualized sensors as well as their use modalities, and will focus on the inclusion of sensor virtualization in a distributed application development environment based on the exchange of executable code for resource-constrained devices (DC4CD).

REFERENCES

[1] D.Peri, “Body Area Networks and Healthcare”, In: S.Gaglio, G.Lo Re (eds) Advances onto the Internet of Things. Advances in Intelligent Systems and Computing, Springer, vol.260, pp.301-310, doi:10.1007/978-3-319-03992-3_21.

[2] J.Dieffenderfer, H.Goodell, S.Mills, M.McKnight, S.Yao, F.Lin, E.Beppler, B.Bent, B.Lee, V.Misra, Y.Zhu, O.Oralkan, J.Strohmaier, J.Muth, and D.Peden, A. Bozkurt, “Low-Power Wearable Systems for Continuous Monitoring of Environment and Health for Chronic Respiratory Disease”, IEEE Journal of Biomedical and Health Informatics, vol.20, n.5, 2016, pp.1251-1264.

[3] F.Wu, T.Wu, M.R.Yuce, “An Internet-of-Things (IoT) Network System for Connected Safety and Health Monitoring Applications”, Sensors 2019, 19, 21

[4] S.Saginbekov, C.Shakenov, “Hybrid simulators for wireless sensor networks”, 2016 IEEE Conference on Wireless Sensors, ICWiSE 2016.

[5] N.Padmalaya Nayak, S.Pallavi, “Comparison of Routing Protocols in WSN Using NetSim Simulator: LEACH vs LEACH-C”, IEEE proceeding IJCA, vol. 106, no. 11, November 2014, pp.0975-8887.

[6] J.Albesa, R.Casas, M.T.Penella, M.Gasulla, “REALnet: An Environmental WSN Testbed”, Proceedings of the International Conference on Sensor Technologies and Applications, 2007, pp.502-507.

[7] A.De Paola, P.Ferraro, S.Gaglio, G.Lo Re, M.Morana, M.Ortolani, D.Peri, “An ambient intelligence system for assisted living”, 2017 AEIT International Annual Conference, Cagliari, 2017, pp.1-6, doi:10.23919/AEIT.2017.8240559.

[8] A.De Paola, P.Ferraro, G.Lo Re, M.Morana, M.Ortolani, “A fog-based hybrid intelligent system for energy saving in smart buildings”, Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 7, 2020, pp.2793-2807, doi:10.1007/s12652-019-01375-2

[9] S.Gaglio, G.Lo Re, G.Martorella, D.Peri. “A Lightweight Middleware Platform for Distributed

Computing on Wireless Sensor Networks", The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014), Procedia Computer Science, vol.32, 2014, pp.908-913, doi:http://dx.doi.org/10.1016/j.procs.2014.05.510.

[10] S.Gaglio, G.Lo Re, G.Martorella, D.Peri, "High-level Programming and Symbolic Reasoning on IoT Resource Constrained Devices", Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, vol.150, 2015, pp.50-63, doi:10.1007/978-3-319-19656-5_9

[11] S.Gaglio, G.Lo Re, G.Martorella, D.Peri, "DC4CD: A Platform for Distributed Computing on Constrained Devices", ACM Trans. Embed. Comput. Syst. 17, 1, Article 27 (January 2018), 25 pages, doi:https://doi.org/10.1145/3105923.

[12] S.Gaglio, G.Lo Re, G.Martorella, D.Peri, "A fast and interactive approach to application development on Wireless Sensor and Actuator Networks", Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, 2014, pp.1-8, doi:10.1109/ETFA.2014.7005179.

[13] S.Gaglio, G.Lo Re, G.Martorella, D.Peri, "WSN Design and Verification Using On-Board Executable Specifications", IEEE Transactions on Industrial Informatics, vol. 15, no. 2, Feb. 2019, pp.710-718, doi: 10.1109/TII.2018.2840534.

[14] Raspberry Pi 3 Model B+ Specifications `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus`

[15] STEVAL-STLKT01V1 - Sensortile development kit. Available online: `https://www.st.com/resource/en/data_brief/steval-stlkt01v1.pdf`

[16] Bluetooth low energy Characteristics, a beginner's tutorial (2017-03-18). Available online: `https://devzone.nordicsemi.com/nordic/short-range-guides/b/bluetooth-low-energy/posts/ble-characteristics-a-beginners-tutorial`

[17] L.Leonardi, G.Patti, L.Lo Bello, "Multi-hop Real-time Communications over Bluetooth Low Energy Industrial Wireless Mesh Networks", IEEE Access, 2018, doi:10.1109/ACCESS.2018.2834479.

[18] A Nodejs BLE (Bluetooth Low Energy) central module `https://github.com/noble/noble`

[19] SISFALL Dataset. Available online: `http://sistemic.udea.edu.co/en/investigacion/proyectos/english-falls/`

[20] A.Sucerquia, J.D.Lopez, J.F.Vargas-Bonilla, (2017). SisFall: A Fall and Movement Dataset. Sensors. 17. 198. 10.3390/s17010198.