# Low-Code Solution for IoT Testing

Hugo Cunha
Faculty of Engineering,
University of Porto
Porto, Portugal
Email: up201404587@fe.up.pt

João Pascoal Faria
INESC TEC and
Faculty of Engineering,
University of Porto
Porto, Portugal
Email: jpf@fe.up.pt

Bruno Lima
INESC TEC and
Faculty of Engineering,
University of Porto
Porto, Portugal
Email: bruno.lima@fe.up.pt

***Abstract*—In recent years, there has been an increase in the use of Internet of Things (IoT), mostly resulting from the increase in the number of ever-smaller devices being commercialized by different vendors with different purposes. These devices and the ecosystem that they are part of typically are highly complex due to their heterogeneous nature and are typically end-user focused. As a result, testing such systems becomes a challenge, especially when the system logic is configured by end users. To address such challenge, a low-code approach was designed that allows users with no programming, or testing knowledge, to test an IoT scenario with a set of sensors and actuators. This approach has a set of test patterns implemented out-of-the-box so the user simply executes the test suggested by the tool and observes the results. The work was validated in a case study involving a group of users with and without technical knowledge. The results showed that both groups managed to finish the tasks selected with ease. The results obtained during the validation phase with end users affirm that the approach eases the process of testing such systems.**

***Keywords*–*Visual Interface; IoT; Integration Testing.***

## I. Introduction

Over the last few years, there has been a growth in the usage of Internet of Things (IoT) devices [1]. These small devices have been “turning heads” in terms of robustness, price and general usability [1]. From sensors with the intent of measuring the temperature or humidity of a room to actuators, capable of turning the TV on or adjusting the air conditioning, these devices are taking a leap forward in both technology and also complexity. With the addition of more features and the increasing number of manufacturers providing low-cost solutions which do not provide integration techniques, there is a rising problem - guarantee the correct communication and functioning when grouped together.

One of the areas that is on the rise, regarding IoT, is eHealth, automotive and home automation, or domotics. eHealth is a somewhat recent area that integrates informatics and health in the same domain [2]. In other words, is the possibility of creation of new services in the healthcare domain using the internet. Automotive is also another domain to get certain attention [3]. Lately, a large number of companies are attempting to create autonomous vehicles. These vehicles are expected to not only detect all kinds of road hazards - pedestrians, road signs, traffic lights - but also be able to communicate with the infrastructure. Home automation, or domotics, is an example of an area which is having an increased use in our lives [4] for simple home solutions where, for example, smart sensors, connected to an air conditioning are able to control the temperature of a room.

From the previously observed domains (eHealth and automotive), it is perceptible that errors derived from these activities may cause serious damage to human lives [5]. It is critical that such infrastructures and systems are tested to guarantee its correct functioning. In the case of domotics, it is not such a critical area since it does not deal with human lives directly but, nevertheless if, for example, a door lock is connected to a device that fails to operate it becomes a crucial safety problem.

Another problem that we can point out is the fact that most of the products for such scenarios are developed on different platforms, in different programming languages and manufactured by distinct companies [6]. Communication between such devices may be very hard to establish because of such aspects.

Lastly, the process of testing such complex scenarios is still a difficult task and not everybody can accomplish. In fact, there are already a large number of tools that allow for testing of such devices mentioned, although most of them are proven to be out of the domain we are presenting and are unable to be used by people with low expertise in the area. Most are focused on large-scale systems and require either programming or a high level of technical knowledge to operate.

The rest of the paper is organized as follows: Section II presents the proposed solution; evaluation is described in Section III; related work is presented in Section IV and conclusions and future work are presented in Section V.

## II. Proposed Solution

In this section, we present our proposed solution to reduce the expertise needed for a user to test IoT scenarios.

### A. *Architecture*

The developed low-code solution for IoT testing comprises two components: a visual interface (Izinto Frontend in Figure 1) for the configuration of IoT test scenarios, selection of applicable test patterns and visualization of test results; and a second one which is a pattern-based integration testing framework for IoT (Izinto Backend), developed in a previous work [7], responsible for test execution.

The Frontend is a Node.js [8] web application coupled to a diagram design framework, JointJS [9]. The framework allowed for the development of the blocks, links and interaction

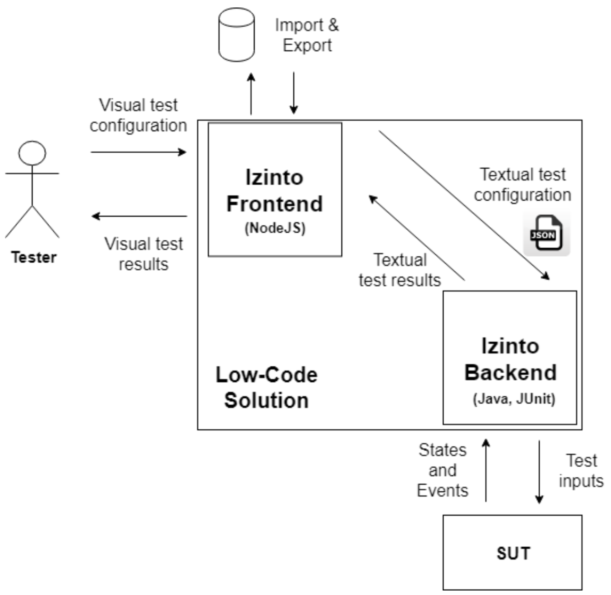

Figure 1. General architecture of the work developed

between them, and the Node.js is responsible to handle server-side events, such as the test execution and test results report.

In the Frontend, the user visually designs the application scenario, its connections and parameters. After that, the tool will suggest test patterns that can be executed and the user chooses the ones desired. Following, the tool will convert the designed scenario into a format the Izinto backend can understand and start its execution. After the tests are executed, the results are reported back on the visual interface so the user can understand what was successful or unsuccessful.

The Izinto backend comprises two main modules: test logic module and IoT components module. The test logic module implements a set of test patterns using JUnit [10]. The IoT module is responsible for the necessary communication with the IoT devices during test execution. As to input data, prior to the starting of the integration test, the Izinto backend interprets a configuration file, in JSON, with the information needed for the application of the test patterns. An excerpt of this file can be observed in Figure 3.

In Figure 1, it is possible to observe the components and flow of our solution.

### *B. Visual Definition of Test Scenario*

In Figure 2 we can observe the user interface of our tool. In this picture, it is possible to distinguish five main parts, or areas, each regarding different functionalities or objectives. In this section, we will focus on the toolbox (1) and workspace (4).

On the left (1), there is the toolbox in which the user can start creating blocks/elements. The blocks are abstractions for physical sensors, actuators, applications and notifications. Each block has a small form for the user to fill in and specify the parameters of the abstraction it represents and also for the test to be executed. In the middle (4), there is the workspace in which the user is able to move, connect and edit the block's properties and test parameters. Figure 2 shows the blocks and links for the running example, as well as a form with the properties of the blocks.

For a better user experience, the tool allows for exporting and importing already designed scenarios as JSON files (5).

### *C. Test Selection*

After describing the application scenario, the user must indicate which tests he/she wishes to execute. In the Test Pattern Arena (2) in Figure 2, there are five test patterns available: action, alert, periodic readings, user triggered readings and actuator.

At this stage, the users select the test patterns to apply. The tool will check which tests the user is able to run, through an algorithm which was developed to suggest test patterns, in regard to the designed scenario. The execution of the algorithm is triggered when there is any change in the workspace and will attempt to find flows that represent one of, or more than one the five available patterns. Once it detects a test pattern able to be executed, it will make it available to run. In the running example, the suggested test pattern is action, since there is a sensor connected to a logic box which is connected to an actuator.

### *D. Test Execution*

When the user presses the test execution button (5), the tool generates a JSON file, similar to the one present in Figure 3. For each test pattern, a different configuration file is generated, although they are all ran at the same time. The Izinto backend will execute the tests by communicating with the devices within the system under testing (sensors, actuators, etc).

### *E. Visualization of Test Results*

Upon test completion, the Izinto backend will return a report in the format of text. Such report will be interpreted by the logic module of the web application and will demonstrate to the user the errors that may have occurred. There are three ways the results are displayed to the user. Firstly, the user will have ”drawn” beside each pattern a red cross (in case of failure) or a green checkmark (in case of success), in the Test Pattern Area (2). Secondly, these same figures will be displayed inside each sub-test (each test is divided into smaller and more specific tests). Lastly, the elements of the workspace will be painted green, red or grey in case of success, failure or not tested, accordingly. In Figure 4 we can observe the results of a test which had some failures, but also some successes. In this case, it was executed an action test, which involves a sensor, a logic box and an actuator. As observable, in case of the sensor that performed readings correctly and within delay and deviation set by the user and failures. In case of the actuator, it did not change its internal state upon having received an order to do so by the application, so it is painted as red since it failed.

## III. Evaluation

With the objective of validating the work developed, it was conducted a usability evaluation experiment involving users with the objective of assessing the following research questions:

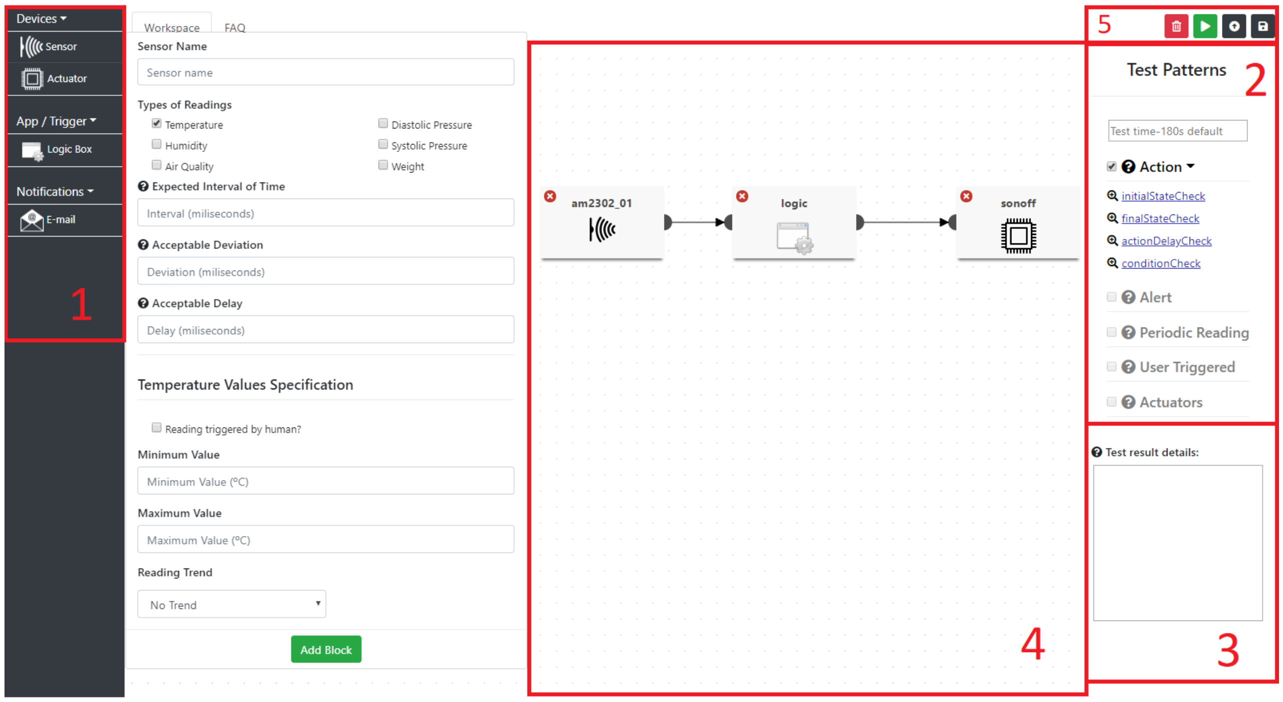

Figure 2. Workspace for scenario definition and pattern test suggestion to be applied to each scenario. The image is already split into the most important parts.

```
{"type": "Sensor",
 "specs": {"id": "AM-2302", "readingSettings":
          [{"type": "Temperature",
            "expectedInterval": 60000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": -40,
                                   "maximumValue": 80}},
           {"type": "Humidity",
            "expectedInterval": 120000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": 0,
                                   "maximumValue": 100}}]
}}
```

Figure 3. Excerpt of a configuration file as input to Izinto Backend.

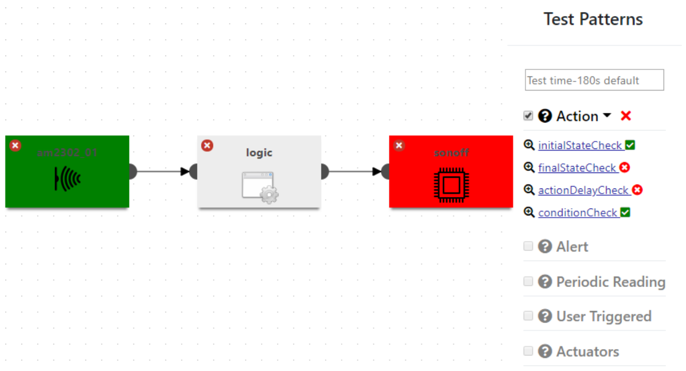

Figure 4. Example of a test result with some successes, but also some failures.

- RQ1: Do users find it easy and pleasant to create and execute automated tests for IoT systems using the developed solution?
- RQ2: Regarding RQ1, are there differences between users with a low and high technical background?
- RQ3: Are users able to quickly create and execute automated tests for IoT systems using the developed solution?
- RQ4: Regarding RQ3, are there differences between users with a low and high technical background?

The metrics used to evaluate were the time per task and the results obtained on a questionnaire made at the end of each task and also at the end of the test. The choosing of participants was split into two parts. In the first, it was selected participants with lower programming and testing knowledge and, secondly, it was gathered users with higher expertise on both areas.

The test was composed of the developed solution and the system under test. The test was executed in a lab which simulates a smart house presented in Figure 5. In this scenario, there is a temperature and humidity sensor, connected to a Raspberry Pi, and a smart socket connected to an air conditioning. Both the socket and the Raspberry Pi are connected to the Wi-Fi of the building. The Raspberry Pi is able to control the temperature of the room and toggle the air conditioning on, or off, as the temperature reaches unwanted values.

The case study involved the execution of five tasks. In the first one, the users only had to import an already set up scenario from the computer and run the available test. Secondly, the user had to test the correct functioning of an actuator. Thirdly, the user would test the correct functioning of the temperature and humidity sensor by running a test of periodic readings. Fourthly, the user would test the triggering of an action when the values read by the sensor would reach certain values, set by

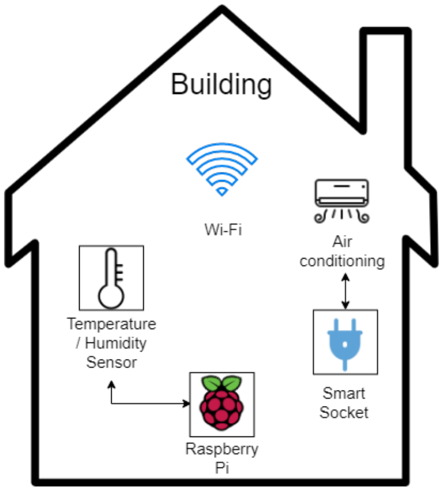

Figure 5. Application example of an IoT system.

the user. The last task had the purpose of testing the triggering of an alert (by sending an e-mail) when the values read by the sensor would reach certain values.

Aggregated in Table I is the data gathered from both the questionnaires at the end of each task and the time per task. The table is split into tasks, one to five, and inside each one, it is possible to observe the scores of both easiness and pleasantness to execute it. Such scores range from one to five, being one very difficult / very unpleasant, and five very easy / very pleasant. On the rightmost column, are present the T-Test values for the difference of the two means [11], regarding each parameter. The differences are not statistically significant (T-Test values greater than 0.05) except one (time to perform the last task).

We can conclude that, based on the results obtained, the visual interface was considered very easy and pleasant to use by both groups of users, allowing to prove RQ1 and RQ2. Also, it is important to point out that, although there is a time difference between the two, that gap is not that big - RQ4. Although, there is a certain time difference between both groups, in general, the times are somewhat small for a task of this nature - RQ3.

The users were also asked to give some feedback regarding the tasks, or the general application. This data was gathered and analysed and most of the suggestions pointed out were implemented. There were also a couple suggestions that were considered as future work.

## IV. Related Work

In this section, it will be made reference to existing solutions for IoT regarding both development and testing. It will be made reference to its IoT layer, its test level, test environment, supported platforms, its scope and the presence of a visual interface.

**PlatformIO [12].** It is an open-source IDE for IoT development. It supports multiple platforms and a unit testing system. It works on the edge layer and its tests are run within the physical devices. It is a commercial tool and does not possess a visual interface for test configuration. Has support for multiple platforms.

**IoTIFY [13].** It is a cloud-based IoT performance testing platform for very large-scale scenarios. IoTIFY works on the Edge, Fog and Cloud layers of IoT and has support for unit, integration and system testing. The test environment in IoTIFY is only for simulated devices. The interface is called ”Virtual IoT lab” and enables the user to simulate a virtual hardware lab.

**FIT IoT-LAB [14].** It is a very large-scale infrastructure with the purpose of testing a large number of small wireless sensors and other heterogeneous communication devices. It supports layers of IoT Edge, Fog and Cloud and its main purpose is the testing of scenarios and not for the development of IoT solutions. It is both for academic and commercial use and allows unit, integration and system testing. It does feature a visual interface.

**MAMMotH [15].** It is a large-scale IoT emulator for mobile connected devices through GPRS. MAMMotH works on all IoT layers and allows for integration and system testing. The connection to the devices is emulated and both the platforms supported and its license are two aspects that remain unknown, although MAMMotH was developed in an academic environment. There is no information regarding the existence of a visual interface.

**TOSSIM [16].** It is a wireless sensor network simulator. It was built with the specific goal to simulate TinyOS devices. It is a tool focused on testing, supporting integration one and only support the Edge IoT layer. It was developed in an academic environment and its license is open to be reused. It uses simulated radio connected devices and its graphical user interface (GUI) is optimized for such goal.

**SimpleIoTSimulator [17].** It is an IoT Sensor/device simulator that creates test environments with a large number of sensors and gateways. It is a framework with a focus on the integration testing of devices. It is limited to the Edge and Fog IoT layers and there is no information regarding the existence of a visual interface or its domain. It does not use physical devices and there is no information regarding the supported platforms. It is a commercial tool and its license is closed.

**MBTAAS [18].** It is an approach that combines Model-Based Testing techniques and service-oriented solutions in a platform that allows IoT testing. Allows for testing across the four levels - Unit, Integration, System and Acceptance and it also features support for all IoT Layers. There is no information regarding the number of supported platforms and it is considered an academic tool. Unfortunately, there is no information regarding its license. It features a visual interface for the selection of tests and results visualization, unfortunately, there is still some expertise required to operate it.

**SWE Simulator [19].** It is a tool developed with the intent of representing multiple types and different number of sensors and integrate it with a standard sensor database. It is very much focused on testing of wireless sensor networks and only supports Edge IoT layer. In terms of testing, it only supports system testing and does not use physical devices but simulated ones. It features a GUI but with the objective of monitoring the small wireless sensors' activity. SWE Simulator is a tool developed in an academic environment.

**MobIoTSim [20].** It is a mobile IoT simulator to help investigators handle multiple devices and demonstrate IoT

TABLE I. GROUPED TASKS' QUESTIONNAIRES DATA

| Tasks | Question | Global Average | Higher Skill AVG | Lower Skill AVG | T-Test |
|---|---|---|---|---|---|
| **#1** | Easiness | 4,91 | 4,88 | 5,00 | **0,351** |
| | Pleasantness | 4,73 | 4,63 | 5,00 | **0,197** |
| | Time | 0:33 | 0:29 | 0:46 | **0,063** |
| **#2** | Easiness | 5,00 | 5,00 | 5,00 | - |
| | Pleasantness | 4,82 | 4,75 | 5,00 | **0,17** |
| | Time | 1:46 | 1:43 | 1:53 | **0,587** |
| **#3** | Easiness | 5,00 | 5,00 | 5,00 | - |
| | Pleasantness | 4,82 | 4,88 | 4,67 | **0,605** |
| | Time | 1:43 | 1:38 | 1:55 | **0,091** |
| **#4** | Easiness | 4,91 | 5,00 | 4,67 | **0,423** |
| | Pleasantness | 4,73 | 4,75 | 4,67 | **0,837** |
| | Time | 3:58 | 3:51 | 4:18 | **0,540** |
| **#5** | Easiness | 5,00 | 5,00 | 5,00 | - |
| | Pleasantness | 4,91 | 4,88 | 5,00 | **0,351** |
| | Time | 1:38 | 1:10 | 2:52 | **0,2e-3** |

applications using them. It is a testing framework that focuses on the Fog and Cloud IoT layers and supports integration testing. It was developed in an academic environment and its license is open to reuse. There is no information regarding the number of platforms it supports and the only available visual interface is an Android application so that the tester has access to the values being read by the sensors.

**DPWSim [21].** It is a framework that helps developers to implement and test IoT applications by simulating physical devices. Although the team involves one investigator from the commercial scope, it is considered an academic tool. DPWSim works on the Fog and Cloud IoT layers and supports integration testing. It only supports DPWS platforms. Features a visual interface but, unfortunately, only provides managing and simulation support for DPWS devices.

**Atomiton IoT Simulator [22].** It is a testing framework that simulates virtual sensors, actuators and devices with unique behaviours, which communicate in unique patterns. It supports all types of test levels and works on every IoT layer. Its license is closed and it features a visual interface but for virtualization of devices. It does not support any forum or community for developers to settle their doubts.

**Node-RED** [23]. It is a browser-based visual editor that allows a user to connect and wire together online services and API's. It makes use of flow-based editing that makes it visually easy for an average user to create simple, or more advanced, connections between the referred entities. It is not focused on neither testing or developing of IoT solutions but instead a visual interface to connect multiple and diverse services, sensors, actuators, etc.

**Easyedge** [24]. It is an IoT solution using a low-code approach for the connection of multiple devices without the need for programming. It also possesses a flow-based programming visual interface that enables the user to design their scenario. This platform allows for devices to communicate through the most popular cloud services.

In fact, there are already a large number of tools that provide support for all IoT layers but a few do not allow for integration testing, which is a downside. One of the most important factors we can point out is a large number of closed license tools and the test environment being mostly simulated. Most tools also lack the extensibility needed to work over multiple platforms and most of them are platform-centred. The most crucial point to be evaluated was the existence of a visual interface for easier interaction. We could conclude that most tools did not provide the necessary UI or it was not the most suited for the domain required. Although some are focused on very large-scale systems, there are solutions for smaller and simpler environments. Also, another common issue with such tools is the complexity associated with its use and being mainly focused for experts with very high technical knowledge in the area. There is currently a necessity for tools that allow users from all levels of technical knowledge to test smaller scenarios.

## V. Conclusions and Future Work

With the development of this work, it was attempted to reduce the expertise needed for a user to test IoT scenarios. Thereby, a person with no programming, or testing knowledge, can easily test their systems in the most common patterns. Also, we tried to reduce the time a person with higher knowledge would take to test an IoT environment.

In this article, it is made reference to a visual interface with the objective of filling in the existing gap in IoT solutions for people with lower technical skill. Such interface took advantage of an already developed pattern-based testing framework developed on previous work. This interface allows the user to simulate a real scenario, with a set of devices and applications, and perform integration tests with the help of Izinto as an integration testing framework. The visual interface also allows for the visualisation of test results. With such interface, it is aimed to reduce both the time needed for every user to parameterize its test and allow for a larger number of users to test IoT scenarios.

The work was validated with a case study, including users with a distinct level of technical skill. The case study proved that the solution developed was very good. By the data gathered and present in Table I, it is perceptible that even users with no knowledge could complete the tasks. Overall the feedback collected from the users was very good regarding both the easiness of testing an IoT scenario and not needing a high knowledge to use it.

As future work, there are certain factors it is possible to point out, mainly regarding the addition of functionalities to the current work. There are two paths to follow, one with more focus on the addition of functionalities in Izinto and another one by adding more functionalities to the visual interface and better user experience.

In terms of addition of features to Izinto, it is possible to identify a set of new patterns to be added. As of now, Izinto covers the test of features. There are more patterns that can, for example, cover the connectivity, performance, scalability of IoT systems. By covering a greater set of patterns, it is possible to ensure better functioning of such distinct and heterogeneous systems and ensure their integration. There is also the possibility of creating a new set of test patterns for the scope of IoT.

In terms of the visual interface for testing, there is the possibility of creating a module for displaying the sensor readings, or the actuator's state in real time. By doing this, the tester would feel in a more controlled scenario of test and feel in greater contact with the actual values being used for test purpose.

## REFERENCES

[1] F. Fernandez and G. C. Pallis, "Opportunities and challenges of the internet of things for healthcare: Systems engineering perspective," in Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on. IEEE, 2014, pp. 263–266.

[2] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya, "Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare," Future Generation Computer Systems, vol. 78, 2018, pp. 659–676.

[3] X. Krasniqi and E. Hajrizi, "Use of iot technology to drive the automotive industry from connected to full autonomous vehicles," IFAC-PapersOnLine, vol. 49, no. 29, 2016, pp. 269–274.

[4] A. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home automation in the wild: challenges and opportunities," in proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2011, pp. 2115–2124.

[5] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in 2008 1st international conference on software testing, verification, and validation. IEEE, 2008, pp. 485–493.

[6] B. Lima and J. P. Faria, "A survey on testing distributed and heterogeneous systems: The state of the practice," in International Conference on Software Technologies. Springer, 2016, pp. 88–107.

[7] P. M. Pontes, B. Lima, and J. P. Faria, "Izinto: a pattern-based iot testing framework," in Companion Proceedings for the ISSTA/ECOOP 2018 Workshops. ACM, 2018, pp. 125–131.

[8] Node.js, "Node.js," https://nodejs.org/en/, accessed: 2019-05-19.

[9] JointJS, "Jointjs," https://www.jointjs.com/, accessed: 2019-05-19.

[10] T. J. Team, "Junit," https://junit.org/junit5/, accessed: 2019-01-23.

[11] B. L. Welch, "The significance of the difference between two means when the population variances are unequal," Biometrika, vol. 29, no. 3/4, 1938, pp. 350–362.

[12] P. Plus, "Platformio," http://platformio.org/, accessed: 2019-01-20.

[13] T. GmbH, "Iotify," https://iotify.io/, accessed: 2019-01-20.

[14] F. F. I. T. Facility, "Iot-lab," https://www.iot-lab.info/, accessed: 2019-01-21.

[15] V. Looga, Z. Ou, Y. Deng, and A. Yla-Jaaski, "Mammoth: A massive-scale emulation platform for internet of things," in Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on, vol. 3. IEEE, 2012, pp. 1235–1239.

[16] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in Proceedings of the 1st international conference on Embedded networked sensor systems. ACM, 2003, pp. 126–137.

[17] SimpleSoft, "Simpleiotsimulator," https://www.smplsft.com/, accessed: 2019-01-23.

[18] A. Ahmad, F. Bouquet, E. Fourneret, F. Le Gall, and B. Legeard, "Model-based testing as a service for iot platforms," in International Symposium on Leveraging Applications of Formal Methods. Springer, 2016, pp. 727–742.

[19] P. Giménez, B. Molína, C. E. Palau, and M. Esteve, "Swe simulation and testing for the iot," in Systems, man, and cybernetics (SMC), 2013 IEEE international conference on. IEEE, 2013, pp. 356–361.

[20] T. Pflanzner, A. Kertész, B. Spinnewyn, and S. Latré, "Mobiotsim: towards a mobile iot device simulator," in 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW). IEEE, 2016, pp. 21–27.

[21] S. N. Han, G. M. Lee, N. Crespi, K. Heo, N. Van Luong, M. Brut, and P. Gatellier, "Dpwsim: A simulation toolkit for iot applications using devices profile for web services," in 2014 IEEE World Forum on Internet of Things (WF-IoT). IEEE, 2014, pp. 544–547.

[22] Atomiton, "Atomiton," http://www.atomiton.com/, accessed: 2019-02-02.

[23] J. Foundation, "Node-red," https://nodered.org/, accessed: 2019-01-23.

[24] Domatica, "easyedge," https://www.easyedge.io/, accessed: 2019-01-20.