Design and implementation of a system to manage the automatic behavioral testing of an IoT environment for the company TaIO Systems

Juan David Velasquez Rosero
Advisor: PhD. Wilson O. Achicanoy M.
Company advisor: PhD. Juan Pablo Ruiz Rosero.
Department of Electronic Engineering
Universidad de Nariño

***Abstract*—Software development techniques are a fundamental part of software engineering theory. This is because correct planning and validation of the characteristics of a computer system contribute positively to the process of maintenance, scalability, and correction of errors in it. A widely used technique nowadays is behavior-based development, which has advantages over other methods such as TDD (Test Driven Development), for making use of natural language to describe the scenarios to be evaluated in the tests and allowing easy communication of failures between the different levels that make up a company. The current strong demand for this technique is evident in software development for a variety of purposes, but its implementation in the Internet of Things (IoT) systems has not been sufficiently documented. The purpose of this thesis project is to design and implement a system capable of managing the automatic behavior tests of an IoT environment, in order to detect errors and verify compliance with the defined requirements, different functionalities of the system, in addition to allowing agile and efficient handling of information on failures, between developers, testers and other participants in the business environment. To achieve this objective, the main characteristics that describe the operation of the IoT system and their respective programming languages will be studied, and an optimal strategy will be determined to implement the test tool, taking into account the current conditions of the hardware and software from which it is made. The validation of the design and implementation of the application will be supervised by the professionals of the company TaIO Systems.**

I. Introduction

A. Problem definition

In recent decades, technological advances have been presented with a great impact on different aspects of our daily lives. The growth of our communication networks around the world and access to the internet have produced significant changes in the way we communicate with each other and even the way we connect or relate to the "things" that surround us. The latter is known as the Internet of Things (IoT). The IoT is an ecosystem that is constantly growing that integrates hardware, software, computing devices, physical objects, and animals or people, through a network that allows them to interact, communicate, collect and exchange data. The IoT has become a technological tool with the potential to solve problems in different areas such as medicine [1], [2], [3], security [4], entertainment [5], vehicle tracking and other means of transport [6], [7], the transport and parcel industry [8], [9], agriculture [10], [11], and even as a reinforcement for other technologies, allowing the collection of large amounts of information for later use in training of machine learning algorithms and big data applications [12], [13], [14].

For the implementation of this type of digital ecosystems in which large amounts of connected electronic devices are present, different development platforms, programming languages and experts are involved in the different phases of the development process, due to their correct structure variety of electronic and computer elements, such as servers, microprocessors,microcontrollers, power supplies, sensor networks, communication protocols, encryption mechanisms and many others, each with different characteristics and whose operations must be described, understood and validated before to be released into production.

It should be noted that the issue of software quality has been systematically discussed since the early 1980s with the concept of software engineering, and since then various methods and techniques have emerged that aim to ensure the quality of computer systems. More recently, Behavior Driven Development (BDD) has gained greater acceptance by offering a mechanism to ensure that the software performs as expected, getting it to be adopted by different agile development methods [15].

Finally, it is important to mention that the time of development and correction of faults is an issue of vital importance for companies that are responsible for developing software and hardware. The absence or poor execution of the tools that guide the development and validation of an electronic and / or computer system generates delays in delivery times, high maintenance costs, long periods of fault correction and maximization of project risks. Moreover, the omission of methods and / or software development and validation techniques can generate communication problems when transferring informa-

Final report of the Degree Work presented as a requirement to opt for the title of Electronic Engineer at the University of Nariño, Pasto, Colombia. Abril XX, 2020.

Juan David Velasquez Rosero (2121601297). Student of the Department of Electronic Engineering. email: juand.vr28@gmail.com

PhD. Wilson O. Achicanoy M., Assistant Professor of the Department of Electronic Engineering. email: wilachic@udenar.edu.co

PhD. Juan Pablo Ruiz Rosero, Director of the company TaIO Systems. email: juan.ruiz@taiosystems.com

tion about failures between the different members of the team that direct and develop the system, including a disarticulation between the requirements and needs of the end user and its functionalities. On the other hand, the presence of the aforementioned shortcomings can lead to a lack of trust on the part of developers and testers, due to a decrease in their ability to predict and understand the structure of their work. This is the reason why the BDD concept acquires great importance in this type of application, being an appropriate way to verify the operation of the system, understood as a set of scenarios that describe the different characteristics that make up its multiplatform structure [15], [16], [17].

B. State of the art

The development of computer systems using the behavior-based development method has been analyzed as a starting point for the approach of this project. The present state of the art is generated from an investigation carried out on the most significant results found.

Between May and June 2019, Myint Myint Moe publishes in the International Journal of Trends in Scientific Research and Development (IJTSRD) a comparative study between 3 software development techniques, TDD, BDD and ATDD in which he describes the main advantages and disadvantages of each of the methods [18].

In Brazil, Hugo Lopes Tavares, Gustavo Guimarães Rezende, Vanderson Mota dos Santos, Rodrigo Soares Manhães and Rogério Atem de Carvalho present a set of tools for the implementation, specification and testing of software following Behavior Driven Development (BDD) practices in the language python [19].

Carlos Solís and Xiaofeng Wang present a study of the characteristics of behavior-based development, where they describe the main characteristics of BDD identified through an analysis of the corresponding literature, in addition to the commonly used sets of tools to implement this method. development, providing a basis for understanding BDD, as well as for expanding existing BDD tool sets or developing new ones [20].

In 2019, Mohammad alhaj, Gilbert Arbez and Liam Peyton published a study on the integration of Behavioral Development with Hardware and Software co-design, applied to a renewable energy project in collaboration with a private company in Canada, to build a system of autonomous load management of self-forming renewable energy nano-grids [21].

Finally, Behave provides an open source repository oriented to the implementation of the agile BDD software development technique in the Python language [22].

C. Work contributions

In this project, an automatic behavioral testing system is designed and implemented, based on BDD methodology, for an IoT environment. To achieve this objective, an analysis of the operation of the IoT environment is carried out and the implementation of a simulation system capable of executing all the components of the IoT environment on a computer with a Linux operating system is completed. This simulation system was implemented in such a way that it simulates all the components and different communication channels between them, in addition to having a synchronization mechanism for multiple processes running in different programming languages.

A mechanism is proposed to perform behavioral testing and about 100 testing scenarios are developed capable of facing the system to different conditions based on the number of sensor nodes, percentage of lost packets, disconnection periods, and others, which are justified based on the conditions that the real system is faced with. The performance of the automatic behavioral testing system is measured by taking into account the speed factor compared to execution in a real environment, its capacity for agile debugging of the system, in addition to the contribution to the process of correction of bugs and development of new functionalities.

II. Context and general vision of the project of the company TaIO Systems

TaIO Systems is a company located in the city of Popayán (Colombia), which has a human team with extensive professional and academic knowledge about integrated systems, firmware development, mobile application development, and information and communications technology (ICT) project management. Throughout the company's operating time, the TaIO Systems staff has successfully supported Intel Corporation on various projects, as an independent consulting and development company. TaIO specializes in providing services such as Firmware, mobile systems, and prototypes and offering development for hardware peripherals and common technologies such as Bluetooth 4.0 / 5.0, RFID, Zigbee, and LoRa.

In terms of hardware design, TaIO Systems has extensive experience designing devices under size and power constraints. In the case of mobile systems, they have developed applications on iOS and Android platforms to build hybrid systems using smartphones, which have different types of connection (WiFi, NFC, Bluetooth, USB, etc.) and different types of sensors. Besides, the company develops prototypes using standard software engineering techniques and rigorous test plans. These prototypes are necessary for the validation of architecture and decision-making in information and communication technology (ICT) projects.

The company is currently developing a project in the context of logistics and asset management, the purpose of which is to be able to track packages sent through various modes of transport, monitoring variables such as tilt, temperature, humidity, location, shock and pressure, such as those shown in Figure 1.

The developed system is defined under the category of Wireless Sensor Network and Internet of Things (WSN-IoT). This IoT environment includes a set of nodes called "Tags" which must be attached to the packets or loads to be monitored, one or more devices called "Unified Gateway Server"; that act as a gateway between the 2.4 GHz Low-Rate Wireless Personal Area Network (LR-WPAN) and the transmission of data to the cloud via Wi-Fi or the cellular network; and finally multiple modules of the cloud, which allow communication

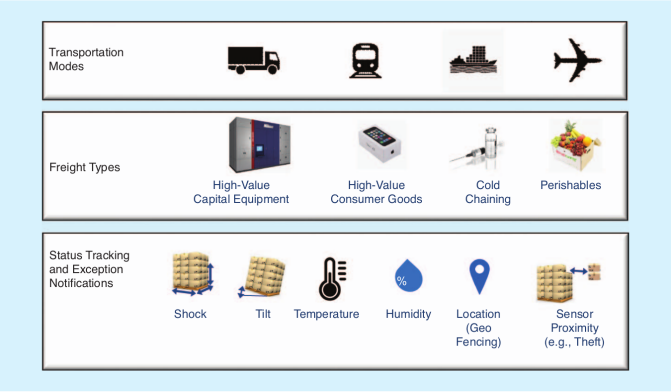

Fig. 1. Logistics and asset management across diverse transportation modes and freight types. Source: Adapted from [23]

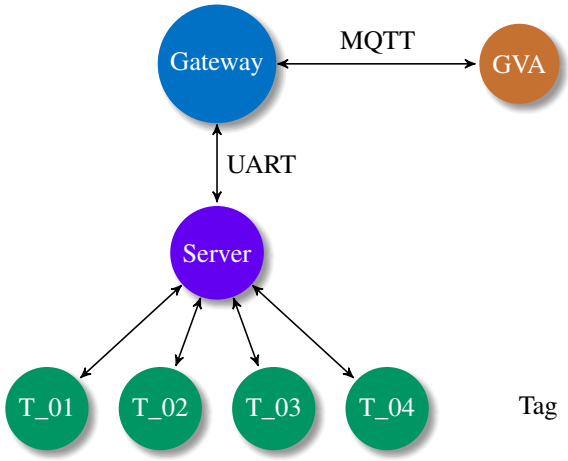

Fig. 2. Structure of the IoT enviroment

with the devices, in addition to monitoring and configuring system characteristics, and visualizing the data, achieving greater operational efficiency [23].

III. Description of the Operation and Components of the IoT Environment

A. Structure of the IoT environment

As mentioned in the previous section, the main components of the system are the tags, Unified Gateway Server device and finally differents cloud modules which together are called Gateway Virtual Appliance (GVA). In this IoT environment, it is used a star network topology in which the nodes are identified as server and client (tag), based on their operations, this topology is showed in the Figure 2. The client node is a physical device/tag used for in situ data collection from environmental and device health sensors. The server (or coordinator) node can have hardware identical to that of a client but acts as the Personal Area Network coordinator (PAN coordinator) and aggregator of sensor data from its client nodes. The clients are equipped with sensors and must be provisioned to associate with an assigned server, which can be connected to an Internet-enabled GW for cloud services. Each device is battery operated with limited resources, and each network of server and clients operates on a shared 15.4 wireless channel using a synchronous time-division multiplexing (TDM) schedule guided by local timers and WSN packet payloads, these mechanisms are explained in more detail in section III-E1.

Each of the components of the system are explained in more detail below:

B. Tag

It is a device based on a 32-bit microcontroller powered by a coin-cell battery, which communicates with different external modules; as PWM controller, NFC RFID, flash memory, SRAM, cryptographic module and temperature, humidity, luminosity, and accelerometer sensors; making use of the SPI or I2C protocols. Furthermore, this device has an integrated RF interface that operates in the 2.4GHz frequency band, which is characterized by having 16 channels ranging from 2.4 to 2.4835GHz, an O-QPSK modulation, and a bit rate of 250 kbps per channel.

The tag has the task of capturing data from the sensors, establishing communication with the server and storing data in periods of disconnection with the wireless sensor network.

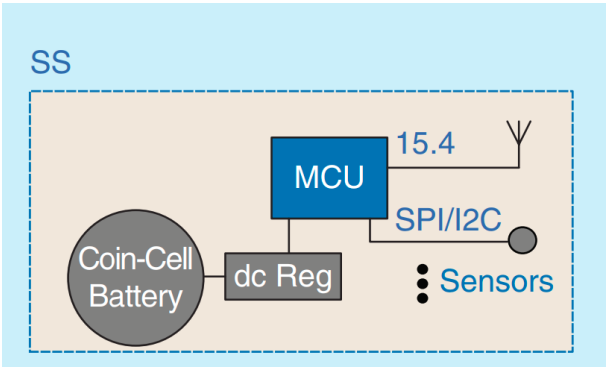

Fig. 3. Block diagrams of Tag module on the IoT environment. Source: Adapted from [23]

The microcontroller is programmed in C language, using multiple libraries, some of them supplied by the microcontroller manufacturers and manufacturers of the different embedded modules, as well as libraries developed by the company's development team.

To manage all the processes that must be executed by the Tag, programming based on state machines and timed state machines was developed within it, in this way, the Tag is able to identify different events that allow it to change the states of operation, programming timed tasks, and defining sleep periods in order to execute minimal instructions quickly, and only as needed, prioritizes sleep to minimize the duty cycle and the current consumption of the circuit. An example of state machine used for an tag is shown in the Figure 4.

Besides, below it is presented an brief description about the main states or phases of the general operation for a tag.

1) **Provisiosing:** It is the initial state for a node. Its function is to wait until basic configurations are provisioned to a node for network connectivity, for example, channel and Gateway Ids, macro frame interval, sensor thresholds, etc.
2) **Association:** After the node receives the configurations in the provision state, the node must join the network. In this phase the client is not yet synchronized, and the clients need to receive their time slot assignment from the server to form the TDM network. For this reason the client node waits until receive from the server an association beacon with the respective time-slot.
3) **Managed:** After the association process is accomplished the client node can work on the operations as a sensor node, that is, the node conduct routine scheduled operations according to micro/macro frame periods, using various beacons to synchronize, change configuration, report sensor data, etc.
4) **Recovery (reconnection):** If a client node loses synchronization due to radio interference, drift, etc., causing a mismatch between the clock reference points of the server and client, it accelerates its wake-up schedule, attempting to increase the probability of receiving a beacon.

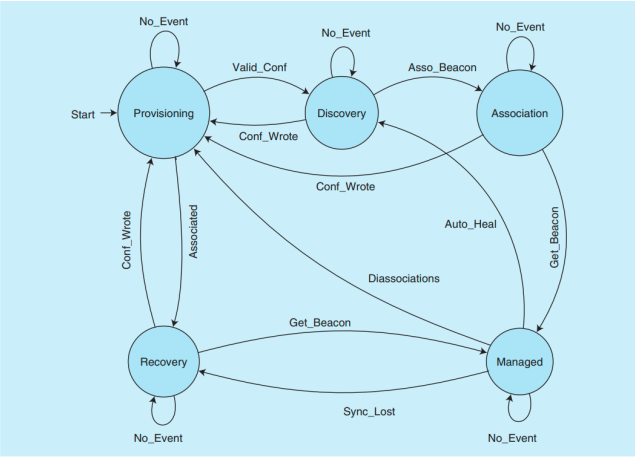

Fig. 4. An example of the management subsystem state machine used for the wireless sensor network. Source: Adapted from [23]

C. Unified Gateway Server

It is a hybrid system made up of an Android device and a microcontroller-based device. These devices are called Gateway and Server respectively and they are explained in more detail below:

1) Gateway: It is a device based on a microprocessor of 4 cores and run a Android operating system. This device is responsible for receiving information related to the tags from the server through a UART module, processing the information, packaging it and finally sending it to the cloud using the WiFi network with the IEEE 802.15.4 technical standard or the cellular network.

Regarding its components, this device is equipped with a GPS module, an LCD screen to display information about the connection process with the sensor network, a power indicator LED, a beeper, and a battery with a capacity of 10 to 40 days. Moreover, the Gateway is responsible for supplying energy to the server.

Different applications work in this device that allows communication between the cloud and the wireless sensor network, each of these fulfills specific tasks, this modular architecture allows distributing the load of the different tasks and managing the search and troubleshooting more efficiently. Among the functions developed by the different applications are:

1) Function as a communication interface for the WSN server, converting complex messages into commands that are understandable to the server and also in the opposite direction.
2) Function as an interface for communication with the cloud for telemetry data.
3) Function as an interface for trusty functions, such as key storage, encryption, etc.
4) Send critical diagnostic information to cloud for troubleshooting purposes.
5) Check for software updates.

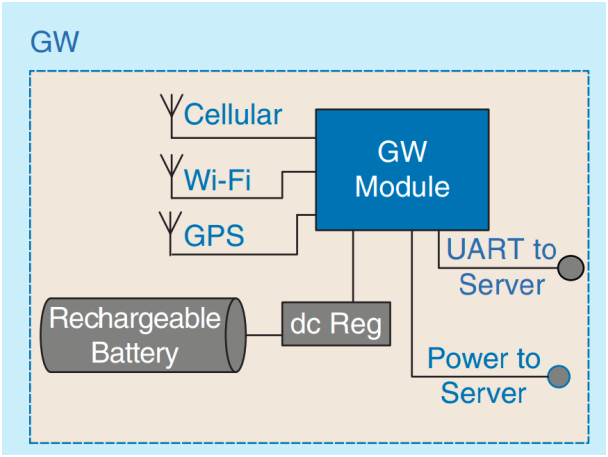

Fig. 5. Block diagrams of Gateway module on the unified GW server device. Source: Adapted from [23]

2) Server: Like the tag, the server is a device based on a 32-bit microcontroller, whose main purpose is to be a communication bridge between Gateway and the tags' network. Therefore server firmware is designed to receive and send messages to both sides. Moreover, it has a connection with different sensors like the tag, for this reason it has to be pending to retrieve data from the sensors module.

To accomplish its functions, the server has an integrated RF interface that operates in the 2.4GHz frequency band, with the same characteristics as the tag, UART, SPI and I2C communication modules, cryptographic module and temperature, humidity, luminosity and accelerometer sensors.

Regarding its operation, it is important to clarify that the server has its programming logic based on state machines. Each state is designed to behave differently using a specific task list routine, and each task within a list runs sequentially as it is shown in the Figure 7.

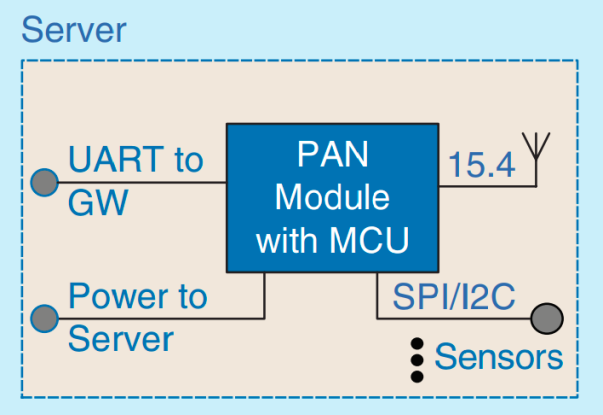

Fig. 6. Block diagrams of Server module on the unified GW server device. Source: Adapted from [23]

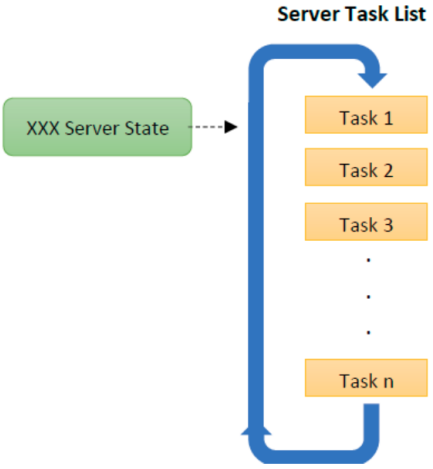

Fig. 7. Server Operation based on task list routine.

D. Gateway Virtual Appliance

The GVA is a group of modules in the cloud, capable of communicating with the sensor network using an protocol for the IoT, capturing and processing messages, in addition to storing data in relational and non-relational databases, managing information related to the configuration, operation, and capture sensor network data, for later viewing on a web platform.

E. Communication between system components

The process of sending data between the components of the sensor network is made up of three levels. The following is a brief description of the operation of each of these:

1) Tag/Server communication: The WSN comprises N sensor nodes (tags) that connect to a receiver (server) using the IEEE 802.15.4 standard. These devices communicate by sending packets with defined structures, specific for each type of message (beacon). The structure of a message can contain different fields, with a length in bytes assigned, such as: a header used as an identifier the message-type, configuration flags, synchronization time, system time, information about the TDM system, system parameters and other payload data. An example message is shown in the Figure 8

Each tag and server device uses local timers and counters that assist in accurate measurement of time intervals. Besides, the system has a mechanism for synchronizing the clocks of the nodes based on the messages called "reference beacons".

The reference beacon marks a "micro-frame period" initiated by the server, which allows the member nodes of the WSN to synchronize the clocks. Upon receiving a beacon, all tags calculate their respective "time slots" and configure their wake-up triggers to respond with a heartbeat or report a message in that time slot.

A multiple of that period is the "macro frame period", in which tags can send additional information (such as long-term analysis data) as well as errors or breaches ("anomalies").

Once the network is provisioned, all tags are anchored to the reference beacon for network management or data collection. The Figure 9 shows the operation mode for 2 tags in a network. Note that the response of client 2 is offset by a time from the reception of the beacon, avoiding the crossing of messages between clients. This operation ensures the precise separation between "time slots" since synchronization is carried out with the arrival of each reference beacon.

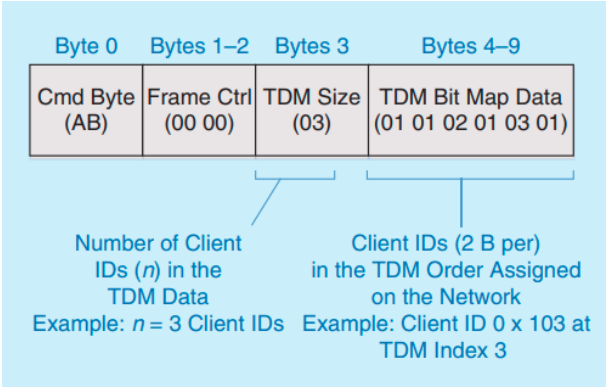

Fig. 8. An message example (beacon with AB header) for a wirless sensor network with 3 tags. Adapted from [23]

2) Server/Gateway communication: In this section the server/gateway communication is described but it is important to clarify that all the incoming data or packets from either wireless client network or serial port will be put into a message entry in related interrupt handlers and be queued in a server message list, so that server can pick up the right time to process and won't interfere with the beacon sending and time synchronization between server and client.

Communication between these components uses the serial port to transmit packets in the form of a byte frame with defined headers to identify the type of message, and different structures depending on the header, the payload of the messages may contain a different number of bytes (length) and be related to system configurations, such as communication channel used, shipment id, operating modes, sensor configuration parameters, as well as provisioning and association processes.

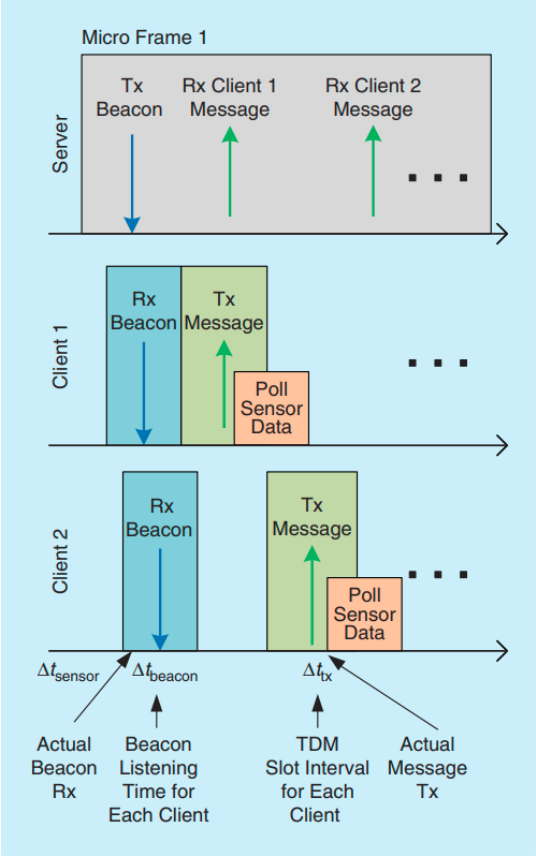

Fig. 9. Operation of an sensor network with two clients. Rx: receive, Tx: transmit. Adapted from [23]

As previously said, the main purpose of the gateway is to serve as a bridge between the Server and the cloud, for this reason one of the applications within the gateway is responsible for converting the bytes received from the server into complex messages to be sent to the cloud. as well as converting the messages received from the cloud into an array of bytes to be sent to the server.

3) Gateway/GVA communication: The GVA uses the MQTT protocol to communicate with the gateway. MQTT is an standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc [24].

The messages sent from the GVA to the gateway, and in the opposite direction, are in the form of JavaScript Object Notation (JSON). JSON is a lightweight data exchange format, reading and writing these messages is simple for humans, for machines it is simple to interpret and generate it. It is based on a subset of the JavaScript programming language but its use is independent of the development language, including low-level and high-level languages. These properties make JSON an ideal language for data exchange [25]. An example JSON message is shown in the Figure 10.


```
{
  "markers": [
    {
      "latitude":40.416875,
      "longitude":-3.703308,
      "city":"Madrid",
      "description":"the capital of
      Spain"
    },
    {
      "latitude":40.417438,
      "longitude":-3.693363,
      "city":"Bogota",
      "description":"the capital of
      Colombia"
    },
    {
      "latitude":40.407015,
      "longitude":-3.691163,
      "city":"Buenos Aires",
      "description":"the capital of
      Argentina"
    }
  ]
}
```


Fig. 10. Example json message.

IV. Description of the Operation of the Simulated System

To run functional tests on the code of the different components, the company has developed scripts that make it easy to compile and run on Linux-based systems. This section will describe the main characteristics of the simulation system, this includes the simulation mechanism of each component and the way they communicate with each other. In the Figure 32 is shown a diagram of the simulated system structure.

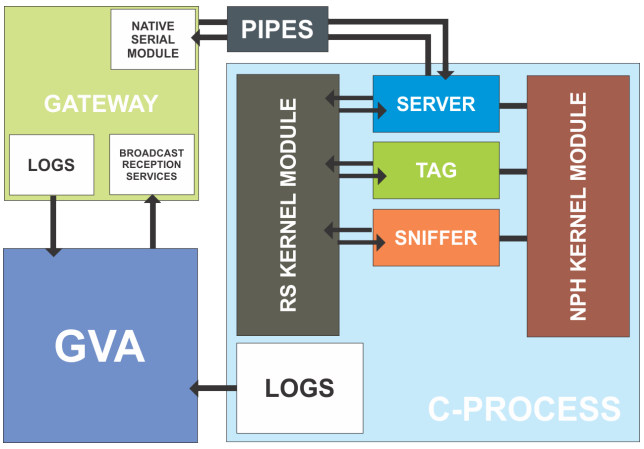

Fig. 11. Simulated system structure

A. Simulated tag and server

As mentioned above, the tag runs firmware written in the C language, which can be compiled on a Linux system without complications using the gcc compiler, generating a binary file

that can be run by the computer. For the binary to be executed correctly, it was necessary to write several scripts in order to replace the operation of different subcomponents of them, for example, flash memory, sensor response, the cryptographic module, random number generation, and the communication phase with the server.

Something of vital importance in this implementation is the ability to switch between the scripts of the simulated system and the real system using compilation flags, allowing to share most of the source code between these two modes of operation and thus the main operation of the tag is not altered, this includes the handling of its state machines, timed state machines, task listing routine, and other execution routines.

B. Sniffer

This C script allows you to capture the information transmitted by the communication channel used by the server and the tags, to be viewed in a readable form by the user. In the same way as the server and the tags, this script is compiled using the gcc compilation tool.

C. Synchronization of multiple threads based on kernel

For certain tests, it is enough to simulate each of the system components separately, but to verify functions that define more complex behaviors, it is required to simulate more than one component at the same time, for example a server and a tag, or a server and multiple tags.

It is in these cases where a mechanism is required that allows multiple scripts (simulated firmwares) to be executed and their clocks synchronized independently of the priority given by the computer's operating system to each of the executed processes.

To meet this objective, we worked together with the company's team in the implementation of a system that allows the execution of multiple binary files resulting from the compilation of the sniffer, server and tags. Additionally, a kernel module was implemented capable of handling global variables shared by all processes such as simulation time and connection/disconnection flags. This module is called the Native Process Handler Kernel Module (NPHKM).

The operation of the synchronization process consists of the execution of sections of code in each of the threads (tags/server), each of these sections is known as "macro-task". Each of the threads executes a macro-task and sends a signal called "Yield" to the kernel module indicating that it is ready for a next clock step. In the case of the server and the tags, a macro-task can be defined between two sleep intervals or in a data reception or transmission cycle. In the case of the sniffer, a macro task corresponds to an operating cycle, in which it checks for messages on the communication channel. A diagram of the synchronization process of kernel module is shown in Figure 12.

All messages sent to the kernel module are transmitted in the form of an array of bytes, processed by a command handler, which chooses a function to execute based on the first byte that plays the role of header and identifier of the message.

The different types of messages received by the module are:

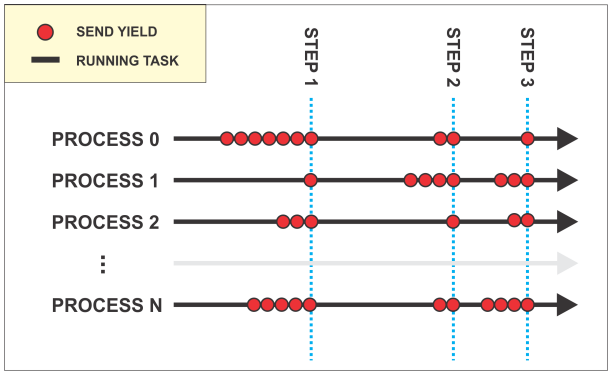

Fig. 12. Synchronization process NPH Kernel Module

- ADD NEW PROCESS: Used to add a new subprocess identification to the list the threads of the kernel module.
- PROCESS YIELD: Used to set the yield flag of a specific thread.
- SET NUMBER ACTIVE PROCESS: Used to set the number of active threads in order to know when all processes are ready for a new clock step.
- SET CONNECTION STATUS: Used to set the state of connection of a specific thread.
- CLEAN: Used to clean the system clock, number of process and flags in the kernel module.

.

Besides, the threads can request the kernel module for the value of the following variables:

- SYSTEM CLOCK: This variable stored by kernel module in order to shared the current simulation time to the threads.
- CONECTION PROCESS FLAG: This variable stored the state of connection of a thread.

D. Simulated gateway

To incorporate the operation of the gateway to the simulated system, the Android SDK is used, which includes an Android device emulator, a virtual device that runs on a computer and allows Android app development and testing without using a physical device. This emulator runs a version of the gateway firmware that covers the function 1 descrita en III-C1. It was not necessary to emulate the other components and applications of the gateway to meet the company's requirements. This Android application is compiled with the development IDE called Android Studio, which makes use of the compilation tool called gradle.

E. Simulated GVA

During the internship it was necessary to fully develop this simulated system module, this is because the gateway messages are not sent to the cloud now, this will be further explored in the section IV-H. The development of this component was carried out in Python language, making use of different free license libraries previously authorized by company policies. This system module is capable of receiving

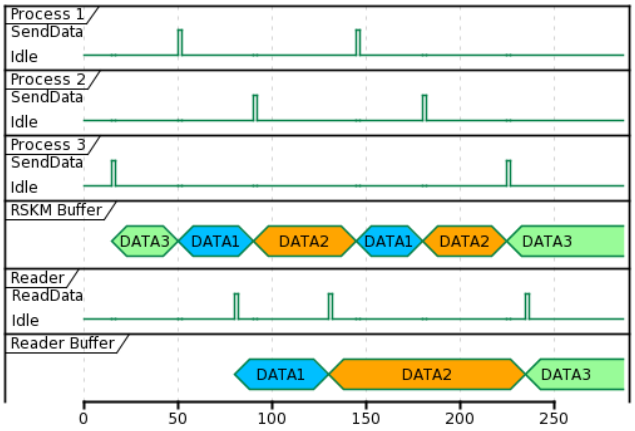

Fig. 13. Time diagram of radio simulator communication example

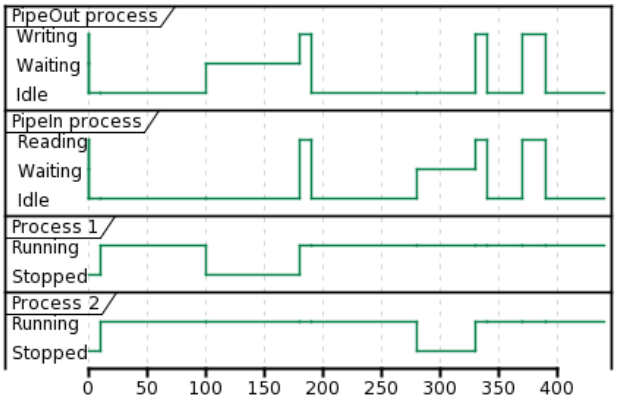

Fig. 14. Time diagram of the pipe communication example

messages from the gateway, processing, validating, executing cryptographic algorithms, and finally providing a response to the sensor network.

F. Simulation of the tag/server communication

As mentioned in a previous section, the server and tag communicate using the IEEE 802.15.4 standard, but in the case of their simulated counterparts, these devices communicate using a kernel module called the "Radio Simulator Kernel Module" (RSKM), which functions as a shared communication channel between all the threads of the simulated system in a very similar way to the behavior in a real scenario. This module allows you to store the messages received from any thread and later send it in response to a read request.

When a new message is received by the kernel, it overwrites the previous message, this could mean a problem for a system that does not use TDM, in which messages from multiple tags would be overwritten before being read by the server.

The operation of this module kernel is showed in the Figure 13

G. Simulation of the server/gateway communication

Unlike tag-to-server communication, server-to-gateway communication has only 2 nodes and there is no need to use a kernel module to share variables with multiple processes. This communication channel must be bidirectional and allow the transfer of information through two separate channels such as UART, therefore a FIFO (first-in-first-out) communication mechanism was used with 2 channels, also called pipe, which allow information to be transmitted between processes in a computer based on a file system [26]. The communication process begins with a process that opens a temporary file on the system in write mode, then sends a data packet, and finally waits for the other part of the pipeline to open the same file in read mode. The processes do not continue their execution until the write / read operation has been completed, for this reason it is necessary to create threads capable of handling the communication task in parallel with the other tasks executed by the gateway and the server, in the same way that UART modules would work in a real environment. An example of this operation is showed in the Figure 14

Since the gateway is run in the android emulator, it is possible to simulate a serial port, passing as emulation parameter the files established for communication with pipes. Moreover, for the communication process between gateway and server to be possible, it was necessary to implement a module that simulates the device that manages the serial communication within the gateway, this module is called Native Serial Module and is composed of multiple scripts in C ++ doing use of the NDK toolset [27].

H. Simulation of the gateway/GVA communication

In the simulation system, the GVA is programmed in python and must be able to receive and send JSON messages to the gateway, so a solution was established to communicate these modules composed of two sharing mechanisms:

1) Send messages to Gateway: Intents are used to transmit messages to the gateway. An intent is a messaging object that you can use to request an action from another component of the app or from an external app [28]. The emulator device is capable of receiving external intents through the commands of the ADB (Android Debug Bridge) tool [29]. These intents are received by a broadcast reception service which is in charge of processing the information from the GVA so that the corresponding command is subsequently sent to the server. An example of a adb command to send an intent is showed in the Figure 15

```
adb shell "am broadcast -a com.example.
application.jsonReciver --es JSON "{ "name": "
Robert", "age": 52, "address": "Cra. 35 #15-22",
 "pets": ["Tasha","Molly","Blast"] }" "
```

Fig. 15. Command to sent a json intent example.

2) Receive messages from Gateway: To receive messages from the gateway, the simulated GVA uses the Logcat tool, which dumps a log of messages from the emulation system of the android device, including stack traces, system error cases and debugging messages that are written from the applications [30].

In the application executed by the GVA, debugging messages are enabled, among which are written the messages that would normally be sent to the cloud in JSON format. In this way the simulated GVA can access them, process them and send a response back if necessary.

I. C-Process

Some of the components that are part of the simulation system are written in C, and compiled with the same gcc compilation tool, for this reason the simulation system has text files called makefiles, which are capable of compiling multiple C programs at the same time using variables, flags, patterns and functions. The tool used to execute these files is called Make [31].

Besides, a script was implemented in order load the binaries of the server, tags, sniffer and kernel modules, and after that, it uses the native c library pthread for running each of the compiled process in a different thread.

V. Description of the BDD methodology

Behaviour Driven Development(BDD) is a synthesis, refinement and evolution of agile development practices derived from Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD).

These techniques allow the developer's efforts to be focused on the implementation of a specific task and are essential in the Quality Assurance (QA) process of software. The concept of quality lends itself to multiple interpretations but always implies that the software satisfies the customer's needs. It is at this point where BDD becomes important and stands out compared to TDD and ATDD because it allows the development of more complex test scenarios that cover behaviors that under other methodologies could be very difficult to define.

The idea of BDD was born from the need for developers to show the results of unit tests, function tests or sets of functions that describe certain behavior of the system, in addition to understanding if the task to be developed had been completed with success, or if there is an error, understand the reason for the failure. Agiledox is a tool that was developed by Chirs Stevenson which allows to extract the names of the unit tests from testing tools like JUnit and print them as sentences.

An example obtained with this tool can be shown in the Figures 16 and 17.

```
public class CustomerLookupTest extends TestCase
  {
     testFindsCustomerById() {
         ...
     }

     testFailsForDuplicateCustomers() {
         ...
     }
     ...
}
```

Fig. 16. Example of the JAVA JUnit test.

```
CustomerLookup
- finds customer by id
- fails for duplicate customers
- ...
```

Fig. 17. Agiledox result.

The word "test" is stripped from both the class name and the method names, and the camel-case writing is transform in the regular text.

This tool is also used by programmers to obtain software documentation files, naming their unit tests with statements that describe the purpose, and even using commercial language to share test results with other members of the company and customers.

In [17], Dan North raises a new strategy that emphasizes in solving the aforementioned, modifying the vocabulary used to define the tests, clarifying the purpose of behavior-based development, basing its structure on a ubiquitous language to describe the requirements of the test. software and finally define them in the form of test scenarios.

The structure or template that defines a BDD scenario has the following form:

1) **Given** some initial context (the givens),
2) **When** an event occurs,
3) **Then** ensure some outcomes.

Besides, the developer can use the extra word "And" in order to extend the "given", "when" or "then" sentences.

Here is a classic example where a customer wants to withdraw cash from an ATM and the software developer wants to evaluate the ATM's performance in different scenarios. For this example, some possible scenarios to evaluate are shown in the figures [18, 19].

Both scenarios are based on the same event and even have some givens and outcomes in common. With this strucure we can re-use givens, events, and outcomes.

```
Scenario 1: Account is in credit

    Given the account is in credit
    And the card is valid
    And the dispenser contains cash
    When the customer requests cash
    Then ensure the account is debited
    And ensure cash is dispensed
    And ensure the card is returned
```

Fig. 18. ATM BDD Scenario 1

```
Scenario 2: Account is overdrawn past the
overdraft limit

    Given the account is overdrawn
    And the card is valid
    When the customer requests cash
    Then ensure a rejection message is displayed
    And ensure cash is not dispensed
    And ensure the card is returned
```

Fig. 19. ATM BDD Scenario 2

When multiple scenarios are focused on the verification of a particular characteristic of a system, they are organized in what is known in BDD as a feature. For the example mentioned above, an example of defining a feature could be the one shown in the Figure 20

```
Title: Customer withdraws cash

 As a customer,
 I want to withdraw cash from an ATM,
 so that I don't have to wait in line at the bank
 .

  Scenario 1: ...
  ...

  Scenario 2: ...
  ...
```

Fig. 20. ATM feature example

This way of writing the scenarios in natural language allows a clearer description of the situation to be tested, the expected behavior, and the meaning of a failure during the process.

Each of these statements is linked to a function that is executed and verified. The language used to code each function is independent of the development methodology, some multiple libraries and tools allow the incorporation of BDD in different languages, developed by companies or communities of programmers.

VI. Description of the implementation of scenarios for the simulated system

The IoT environment developed by the company TaIO System is currently in the support and bug correction phase, for this reason an automatic behavioral testing system was implemented that aims at quality assurance of the different firmware present in the devices.

To carry out this implementation, the following phases were fulfilled:

A. Selection of the BDD library

An important part when developing software for a company is the selection and use of tools, libraries and any third-party code. That is why a review of different libraries that allow the implementation of the BDD methodology was carried out and one of them was chosen based on different technical parameters. Table I shows the compartment of the libraries.

TABLE I
Libreries to BDD implementation

Library	Developers	Language	License
Cucumber-JVM	SmartBear	Java	BSD
Cucumber-CPP	SmartBear	C++	MIT
Cucumber-Android	SmartBear	Java/Kotlin	MIT
Behave	SmartBear/Thirdy-Party	Python	BSD
Jbehave	Jbehave	Java	BSD
Specflow	SmartBear	NET/C chart	BSD
pytest-bdd	Thirdy-Party	Python	MIT
Radish	Thirdy-Party	Python	MIT

In a business environment in which software is developed in order to sell and distribute it to third parties, it is important to ensure that the terms of the license allow this action to be carried out without problems. In the case of the libraries listed, the licenses are of the BSD and MIT type, both grant permission, without charge, to anyone who obtains a copy of this software and the associated documentation files, to operate with it without restrictions, copy, modify and / or distribute this software for any purpose with or without a fee, therefore it is not a differentiating parameter for this library.

The choice of the language in which the behavioral testing system will be implemented is of utmost importance, the difficulty and time of its incorporation into the simulation system may be affected. In the IoT environment there are multiple processes running in different programming languages, communicating through multiple channels, but the processes of authorization, linking, configuration and verification of sensor data are carried out in the highest level component, the GVA, for this The reason sharing the same language with this component ensures easy integration with the simulation system. Based on the above criteria, 3 options are available: Behave, pytest-bdd, and Radish.

In the last stage of selection, the stability of the software to be integrated is taken into account, thinking of adding a library that contains bugs in addition to a small support team could translate into an increase in development time. Thus, a development company with a large team capable of supporting and updating the tool contributes to the stability of the parent software where the library is used.

Based on these 3 selection criteria, the most convenient option is the Behave library which is supported by the SmartBear company with contributions from the community of programmers who use it [22].

B. Automation of the simulated system execution process

One of the requirements to complete the behavioral testing system is that the compilation and execution of the simulated system must be done in an automated way before starting to execute the BDD scenarios. Then a python script was implemented with functions aimed at compiling and executing each component sequentially. A list of the functions and the order of execution is described below:

1) Compile the Gateway application.
2) Compile the C-Process with makefiles.
3) Clone the binary file of simulated tag (if it is necesary).
4) Make fifo to server/gateway communication.
5) Run android emulator.
6) Install the Gateway application.
7) Run the C-Process.
8) Run the Gateway application.
9) Bind logcat output to a document.

Step 3 is only necessary when you want to run multiple tags in the simulation system.

C. Integration of the library to the simulated system

After having a library to implement BDD scenarios and a mechanism to run the simulation system in an automated way,

the Behave library was integrated into the simulation system. This section shows details about the structure of files and folders necessary for compiling the scenarios with this library.

1) Behave project structure: Behave requires a specific file structure for the organization of the features, scenarios, steps and auxiliary files that are used for behavioral testing.

First of all we find the main folder of our project, which must contain a folder called features. This folder should contain a list of the different features that you want to test, each one in a different python file, in which the feature is described and the corresponding scenarios are defined. Moreover, the feature folder must contain a folder called steps in which there are different scripts with the definition of all the functions that execute the scenarios, that is, the translation of the Give, When and Then sentences into the programming language.

An example of this structure is showed in Figure 21.

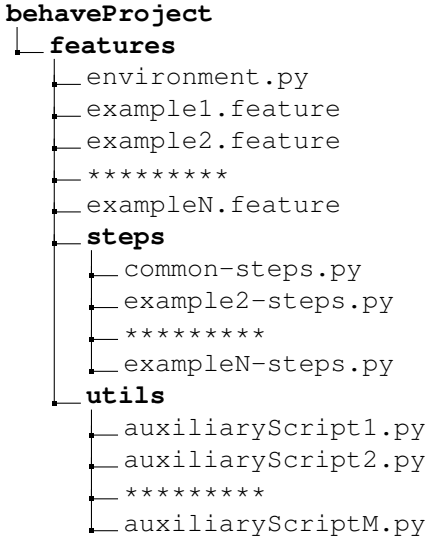

Fig. 21. Behave file structure example.

The enviroment.py file is used to configure functions that run before or after a group of features or scenarios.

Finally, the utils folder store the auxiliary scripts that are necessary for the functionality of the tests.

2) Running a sample scenario for the simulated system: A feature was implemented in order to verify that the simulated system successfully executes each of its components automatically from a BDD scenario, as shown in the Figure 22.

```
Feature: Run the simulated system

  - As a developer,
  - I want to run all the componnets of the
    simulated system,
  - so that I don't have to compile and run each
    process manually.


  Scenario: All the scripts are already to compile
    and run without errors

    Given all the scripts are already to compile
    When I compile and run the Gateway, server and
    tag
    Then I should have a simulated system running
    successfully
```

Fig. 22. Test feature for the simulated system.

The implemented scenario describes the procedure to be tested at a high level. In this case, it is assumed that we have all the tag, server, sniffer, kernels and gateway application scripts, and we proceed to compile and run all the processes, the latter means executing all the automation steps of the simulation described in the previous section. Finally, it is expected to get the whole system working correctly.

In this last step it is verified that each of the components are running and the communication between them is working correctly.

Figures 23 and 24 show the output obtained after running the scenario with behave.

The results allow to verify if the scenario was passed successfully or failed, and if there is a failure, it shows the step that has a problem, and generates a general report of the execution of the scenario including the time it took.

```
Feature: Run the simulated system
    # features/example.feature:1
  - As a developer,
  - I want to run all the componnets of the
    simulated system,
  - so that I dont have to compile and run each
    process manually.

  Scenario: All the scripts are already to compile
    and run without errors
  # features/example.feature:8
    Given all the scripts are already to compile
    # features/steps/example_steps.py:29 0.000s
    When I compile and run the Gateway, server and
    tag
    # features/steps/example_steps.py:33 65.580s
    Then I should have a simulated system running
    successfully
    # features/steps/example_steps.py:43 5.034s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 1m15.348s
```

Fig. 23. Behave output for the scenario without errors.

```
Feature: Run the simulated system
    # features/example.feature:1
    - As a developer,
    - I want to run all the componnets of the
    simulated system,
    - so that I dont have to compile and run each
    process manually.

    Scenario: All the scripts are already to compile
     and run without errors
    # features/example.feature:8
    Given all the scripts are already to compile
    # features/steps/example_steps.py:29 0.000s
    When I compile and run the Gateway, server and
    tag
    # features/steps/example_steps.py:33 65.580s
    Then I should have a simulated system running
    successfully
    # features/steps/example_steps.py:43 5.034s
    Assertion Failed: Simulated Tag is not running

Failing scenarios:
    features/example.feature:8  All the scripts are
    already to compile and run without errors

0 features passed, 1 failed, 0 skipped
0 scenarios passed, 1 failed, 0 skipped
2 steps passed, 1 failed, 0 skipped, 0 undefined
Took 1m10.614s
```

Fig. 24. Behave output for the scenario with errors.

VII. Definition and Development of the Scenarios for the Simulated System

The previous section showed the result of the implementation of an example scenario that ensures that the simulation system is ready for compilation and execution. But this is only the first step to complete the objective of this work. Together with the development team of the company, several characteristics of relevant importance were defined that should be verified, in order to ensure that the system behaves correctly under different conditions. The features are shown below:

- IoT environment working with a single tag
- IoT environment working with multiple tags
- IoT environment working in the presence of packet loss
- IoT environment facing periods of disconnection
- IoT environment working in the presence of unauthorized tags

The first and second of them represent the behavior of the system with a single tag and multiple tags respectively. In this case, the tool has to be able to verify that the system can provision, associate and configure the tags satisfactorily, in addition to receiving data packets in the GVA with the information from the sensors.

In the early stages of the project, the development team had to deal with the loss of packets in the communication channel between server and tags, for which they implemented a packet retransmission routine in order to reduce the probability of information loss and desynchronization. of the tags. For this reason, it was decided to implement the third feature with a series of scenarios that allow corroborating that in the current version of the firmware the system works satisfactorily under these conditions. This behavior was evidenced in different circumstances of a real environment, for example, high levels of noise or interference in the communication channel and distance between tags and server. The distance between tag and server is highly variable in real environments, because companies that use this package tracking system cannot ensure that these (tags) are at a specific distance from the server. Some sections later, an analysis of the relationship between the distance of these components and the percentage of packet loss is presented.

Another very interesting feature of the system is the ability of the tags to store the captures of the sensors in long periods of disconnection. During a disconnection period, the tag stores the data from the sensors in flash memory every macro-interval, when the tag manages to connect to the server again, the tag informs that it has stored data that has not yet been sent, then the server sends a special message to retrieve data by data. This situation occurs frequently in a real scenario in freight transport companies because in sections of the route the packages must be transported by plane or by boat. The regulations applied to the project allow you to keep the radio communication devices turned on in these conditions but without sending packets, for this reason, the tags must store all the data captured from the sensors until after landing, then establish connection in the presence of a gateway-server and finally send all the stored information.

The last characteristic refers to scenarios in which multiple tags try to connect to the system, sending association messages to the server, but not all have authorization to establish connection. This situation occurs when only some of the tags have been registered in the GVA as part of a shipment.

A. *Packet loss and tag-server distance*

The IEEE 802.15.4 PHY layer is responsible for the transmission and reception of information through the radio channel used by the server and the tag. This standard can work in different frequency ranges but the hardware of the IoT environment under which it works uses the 2450 Mhz band because it can be operated all over the world, unlike the 868 Mhz and 915 Mhz bands that are used and regulated in Europe and North America respectively. This layer offers a maximum data rate of 250Kbps and is based on direct sequence spread spectrum (DSSS) technology employing offset quadrature phase-shift keying (O-QPSK) modulation. There are 16 communication channels available in the 2450 MHz range and each channel is 5 MHz wide.

The packets in 2450 MHz PHY operation contain a 5-byte long sync header and a 1-byte long PHY header. These fields are followed by a PHY payload with variable length (up to 127 bytes) [32]. A byte is made up of 2 symbols of 4 bits, each symbol is translated into a 32-bit long quasi-orthogonal pseudo-random noise (PN) sequences shown in Table II.

The PN-bits resulting from the concatenation process of multiple consecutive symbols is modulated onto the carrier using O-QPSK with odd-indexed chips modulated onto the quadrature-phase carrier and even-indexed chips being modulated onto the inphase carrier.

TABLE II
32-CHIP PN SEQUENCES FOR 4-BIT SYMBOLS. SOURCE [33]

Chip sequence number	Data symbol b0 b1 b2 b3	Chip sequence c0 c1 ... c30 c31
1	0000	11011001110000110101001000101110
2	1000	11101101100111000011010100100010
3	0100	00101110110130011100001101010010
4	1100	00100010111011011001110000110101
5	0010	01010010001011101101100111000011
6	1010	00110101001000101110110110011100
7	0110	11000011010100100010111011011001
8	1110	10011100001101010010001011101101
9	0001	10001100100101100000011101111011
10	1001	10111000110010010110000001110111
11	0101	01111011100011001001011000000111
12	1101	01110111011100011001001011000000
13	0011	00000111011110111000110010010110
14	1011	01100000011101111011100011001001
15	0111	10010110000001110111101110001100
16	1111	11001001011000000111011110111000

On the receiver side the process is the opposite. The received signal is demodulated to obtain the stream of bits belonging to one of the 32-bit sequences. Each sequence is compared with the possible values of the PN table and the one with the smallest bit difference is chosen to later be translated into a symbol. This allows to correctly recognize the symbols that have been transmitted as long as the difference between the transmitted sequence and the received sequence is less than the difference with another sequence in the list. Besides, any error in identifying the transmitted symbols is likely to be identified when the packet checksum is calculated and compared with the checksum carried in the packet's header.

A valid chip sequence differs from other chip sequences in at least 12 positions and up to 20 positions, for this reason any errors that contain 5 bits or less between the transmitted and received transmissions can always be corrected and conversely errors that contain 26 or more bit errors can never be corrected [33].

In [33] a calculation of the probability of obtaining a symbol error is performed since an error of N bits is obtained in the sequence sent with n varying from 1 to 32. The results are shown in the Table III.

The probability of bit error for an O-QPSK modulated signal under additive white gaussian noise (AWGN) is given by [34].

$$B = \frac{1}{2} erfc(\sqrt{\gamma}) \tag{1}$$

where $erfc$ is the complementary error function and γ is the signal to noise ratio (SNR).

If B is the probability of receiving a bit with error, y $P_{symerr(n)}$ is the probability of symbol error when n chips are received in error shown in the Table III, the probability of interpreting a symbol incorrect is given by:

$$S = \sum_{n=1}^{32} \binom{32}{n} B^n (1-B)^{32-n} \times P_{symerror}(n) \tag{2}$$

TABLE III
THE PROBABILITY OF SYMBOL ERROR IN IEEE 802.15.4 PHY. SOURCE [33]

The number of chip errors	Probability of symbol error
5 and less	0
6	0.0020
7	0.0134
8	0.0523
9	0.1498
10	0.3479
11	0.6496
12	0.9156
13	0.9968
14 and more	1

A packet that contain any symbol in error is considered a packet received in error. Therefore, the probability P of receiving an error packet, in a transmission of packets of m bytes in length (or 2m symbols), is given by:

$$P = 1 - (1 - S)^{2m}. \tag{3}$$

Although a correspondence between bit error rate (BER) and SNR is easily understood within an engineering context, when explaining and describing the test scenarios it is important to find a relationship with a variable that is used in a business language.

For this reason, the relationship between SNR and the separation distance between two transmission points is described below.

The Friis transmission formula is used in telecommunications engineering, equating the power at the terminals of a receive antenna as the product of power density of the incident wave and the effective aperture of the receiving antenna under idealized conditions given another antenna some distance away transmitting a known amount of power [35]. The Friss transmition equation is given by:

$$\frac{P_r}{P_t} = D_t D_r (\frac{\lambda}{4\pi d})^2 \tag{4}$$

where P_r is the radio wave power received, P_t is the radio wave power transmitted, D_t is the directivity of the transmmitting antenna, D_r is the directivity of the receiving antenna, λ is the signal wavelength and d is the distance between the antennas. Then the received power and SNR can be defined as:

$$P_r = P_t D_t D_r (\frac{\lambda}{4\pi d})^2 \tag{5}$$

$$\gamma = \frac{P_r}{P_{noise}} \tag{6}$$

Based on the previous analysis, the dependence between the probability of error in the transmission of m bytes and the distance between a radio link (tag and server) can be deducted.

To illustrate the dependence between these variables, Figures 25 and 26 show the resulting graphs for transmition of

packet using spread spectrum (PN sequences) and another without this, assuming the use of isotropic antennas, constant noise power and constant transmit power used by the radio link in a real scenario. The values of these constants are shown in table IV.

TABLE IV
CONSTANT PARAMETERS USED FOR THE BER CALCULATION

parameters	Value
D_t	1
D_r	1
P_t	7 dB
P_{noise}	-53 dB

These values were selected in order to represent a scenario with noisy conditions, the real power of transmition used for the microcontroller, and directivity of the isotropic antennas to simplify calculations.

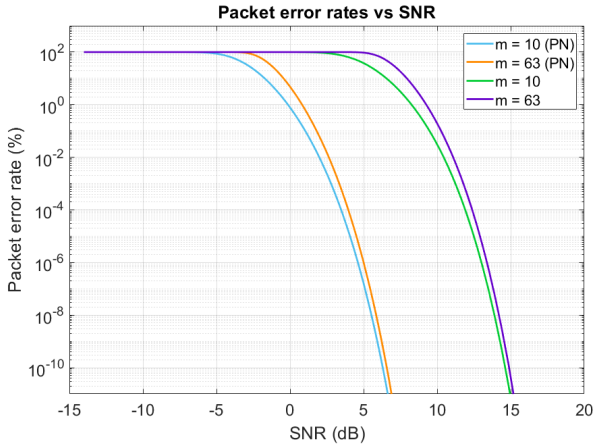

Fig. 25. Plot of PER vs SNR.

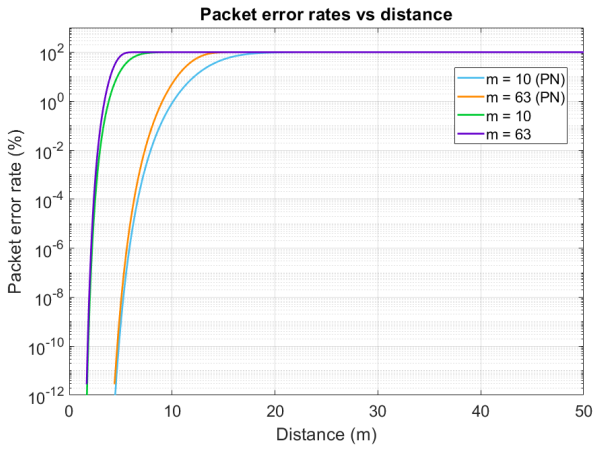

Fig. 26. Plot of PER vs Distance.

Furthermore, the effect of m on the packet error rate can be observed. With larger transmit packets, one of your symbols is more likely to be received in error, so the percentage of erroneous packets increases earlier and with a steeper slope. For the previous graphs, real packet sizes used by the system to transmit messages in each microinterval were chosen.

It can also be observed that in a transmission that does not use Spread spectrum with PN sequence, the packet error increases faster and steeper than the IEEE 802.15.4 PHY transmission. This is one of the benefits of the protocol used by the IoT system implemented by the TaIO systems company.

B. Implementation of the scenarios

Each of the features described in the previous section is composed of different scenarios, which required the implementation of functions, sub-modules and routines. One of these sub-modules is in charge of the cryptographic process, which allows the testing of the system under different security modes. These modes of operation use key establishment protocols that allow two points of the system (GVA and Tag) to establish a shared secret in an insecure communication channel, to later use several layers with message authentication codes based on hashes, schemes block encryption and binary-to-text encoding. All this message authentication process was implemented in both the simulated Tag and the GVA, allowing to authenticate association request messages and sensor data messages. These authentication levels define the different security modes of the system as follows:

- MODE 0: No message authentication
- MODE 1: Authentication of association request messages.
- MODE 2: Authentication of association request messages and sensor data messages.

Another aspect to take into account when defining the scenarios is the limit of the number of tags that the system can handle in the different security modes. In the case of mode 0, the system is capable of handling up to 50 tags, for mode 1 its limit is 10 tags and finally for mode 2 the system can manage a maximum of 5 tags.

On the other hand, the feature that contains scenarios where the tags have disconnection periods requires the implementation of a process of blocking messages from the tag to the server. For this reason, a communication mechanism was developed between the BBD testing script and the NPHKM, which allows the configuration of a flag that represents the connection status of the tags. This flag is requested by each tag when finishing its tasks (before sending yield) and enables or disables the sending of messages to simulate the disconnection. In the section about the synchronization process this flag is defined as "SET CONNECTION STATUS".

To define the scenarios present in each characteristic, two or more of the following simulation characteristics were combined:

- Security mode
- Number of tags
- Number of authenticated tags
- Simulation time
- Disconnection time
- Packet loss percentage

TABLE V
IMPLEMENTED SCENARIOS FOR EACH FEATURE

Iot system working...	Qty	C1	C2	C3	C4	C5	C6
with a single tag	3	X					
with multiple tags	19	X	X				
with packet loss	54	X	X		X		X
with disconnections	16	X	X		X	X	
with unauthorized tags	6	X	X	X			

* Qty: Quantity of scenarios
* C1: Security mode, C2: N°of tags, C3: N° of authenticated tags,
* C4: Simulation time, C5: Disconnection time,
* C6: Packet loss percentage

```
@smode.mt.noise.66
Scenario: {3} tags are already to establish
    communication and send sensor data with packet
    loss percentage of [5]%

    Given the simulated system is running in
    security mode "2"
    When the association process is executed for all
     the tags
    Then I should get sensor data from multiple tags
     for 1 hours
    And ensure there is no lost data
```

Fig. 27. BDD Scenario example with packet loss

In total, 98 scenarios were defined using different combinations of simulation configurations in order to face the system in multiple situations and ensure the quality of the firmwares present in the components of the IoT environment. Table V shows the number of scenarios defined for each feature and the characteristics that were combined for its implementation.

The number of scenarios varies between each feature depending on the number of characteristics that are combined and the impact it has on the operation of the system. For example, the percentage of packet loss has great relevance

```
@special.loggedData.4
Scenario: {10} tags are already to establish
    communication and store data in a disconnetion
    period%

    Given the simulated system is running in
    security mode "0"
    When the association process is executed for all
     the tags
    And the tags disconnet for 2 hours
    And the tags establish connection again
    Then I should get sensor data logged from all
    the tags
```

Fig. 28. BDD Scenario example with disconnection period

and impact on the process of associating the tags to the sensor network, so a large number of scenarios were defined to corroborate that no failure is triggered. On the contrary, the tests with unauthorized tags did not present a significant impact on the operation of the system, therefore the number of defined scenarios is enough to corroborate the robustness of the system to these types of conditions.

VIII. GENERATION OF CURRENT GRAPHS IN THE SIMULATED TAG

An additional functionality developed for the simulated system is the ability to obtain a graph of the current consumption based on the measurement of the time that the tag remains in different operating modes and its relationship with the consumption of the tag.

To capture the consumption measurements in a real tag, the Otii Arc equipment of the QOITECH company was used, which is a power analyzer and power supply that allows feeding a load, recording and displaying currents, voltages and/or UART registers in real time. In the case of current measurements, the equipment has a resolution of nA with a sampling frequency of up to 4ksps. It has a desktop application that allows you to store measurement information and export the data in popular formats such as csv.

Table VI shows the main operating modes of the tag with their corresponding current consumption.

TABLE VI
CURRENT CONSUMPTION OF THE MAIN OPERATION MODES OF THE TAG

Modes	Current (mA)
Sleep	0.034
Rx	31.2
Tx	36.1
Sensor Callback	23.8

To generate a current graph of the simulated tag, markers were placed at the beginning and end of each of the operating modes. These markers are key messages included in the tag debugging records, which allow determining the duration of each process to later relate them to the corresponding current consumption captured in the real scenario. The structure of the markers is simple, they have a message that indicates the

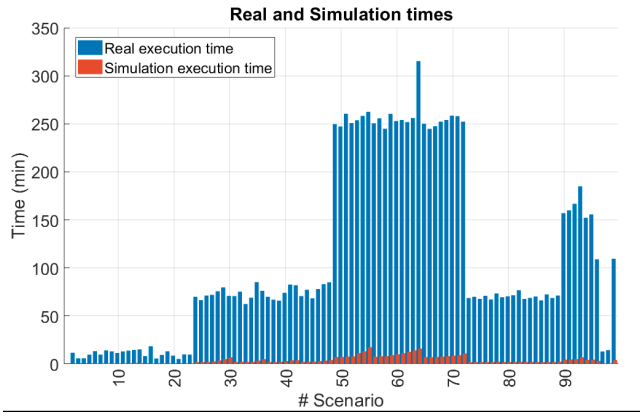

Fig. 30. Real and simulation times.

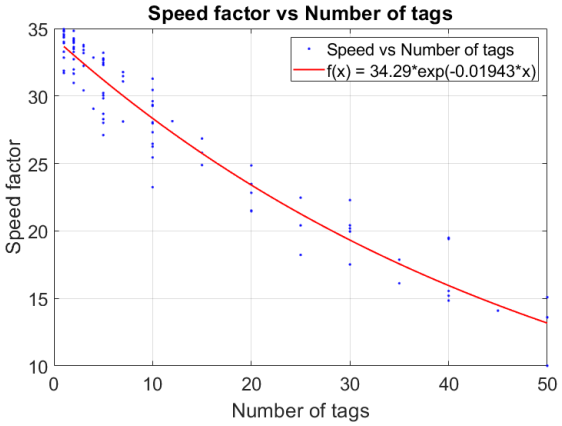

Fig. 31. Speed factor vs N°.Tags.

operating mode and the running time of the system, as shown in Figure 29.

```
    Current consumption ==> MODE: TX init at 9628843
    Current consumption ==> MODE: TX end at 9628849
```

Fig. 29. Current consuption markers example

After capturing all the markers in the tag debugging messages, the data was stored generating the time intervals given in each operating mode, to finally generate equidistant time vectors that allow a graph with data for each step of the system clock. . Each unit of time was assigned its corresponding current value.

IX. Resultados

The results obtained in this work are shown below, organized by sections that describe the main advantages of the behavioral testing system.

A. Speed factor

As mentioned in section VII-B, about 100 scenarios were implemented, which were executed, obtaining measurements of the simulation execution time and the real execution time. The simulation execution time is the time it takes to run the scenario and the real execution time refers to how much time has elapsed in the simulated system clock. Measurements of these times are shown in Figure 30.

The difference in the magnitude between the execution times and the simulation times are significant, but it is observed that the excution durations vary between the different scenarios defined in the behavioral testing tool. This is mainly due to one of the characteristics defined in the implementation of the scenarios, the number of tags. It is intuitive to think that the number of simulated processes has an impact on the computational load of the machine where they are executed, the impact of the number of tags on the speed factor of the simulated system is shown in Figure 31.

Table VII shows the average values of the speed factor of the simulated system versus the real system, for different number of tags.

TABLE VII
SPEED FACTOR FOR DIFFERENT NUMBER OF TAGS

N°.Tags	Speed factor
1	33.92
2	33.23
5	30.76
10	27.89
20	24.28
30	20.71
40	16.43
50	12.90

In the case of a single tag, the behavioral testing system is capable of executing a 4-hour test in just 7.3 minutes, which allows the company's development team to streamline the testing and quality control process in the different system firwares.

The acceleration factor of the simulated system exhibits a behavior that adjusts to a decreasing exponential curve and not to a straight line. This may be due to the multiprocessing capacity of the computer, running multiple processes on different cores and threads. In the present work, a study was not carried out on the use of system resources compared to the simulation system implemented.

B. Debugging of the whole system

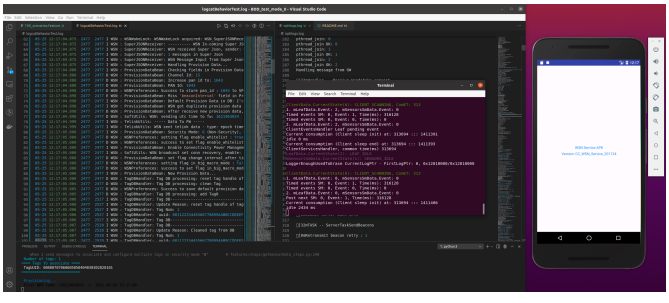

Fig. 32. Simulated system running

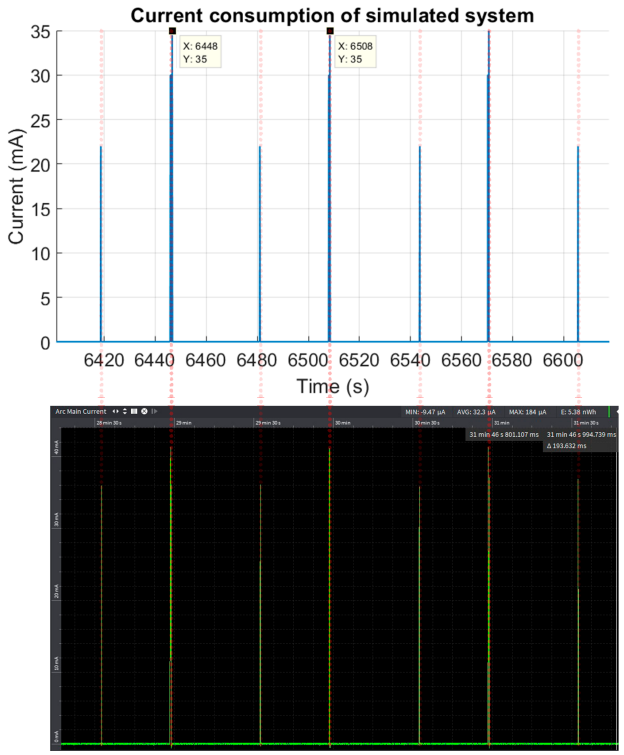

Fig. 33. Micro intervals consumption graph.

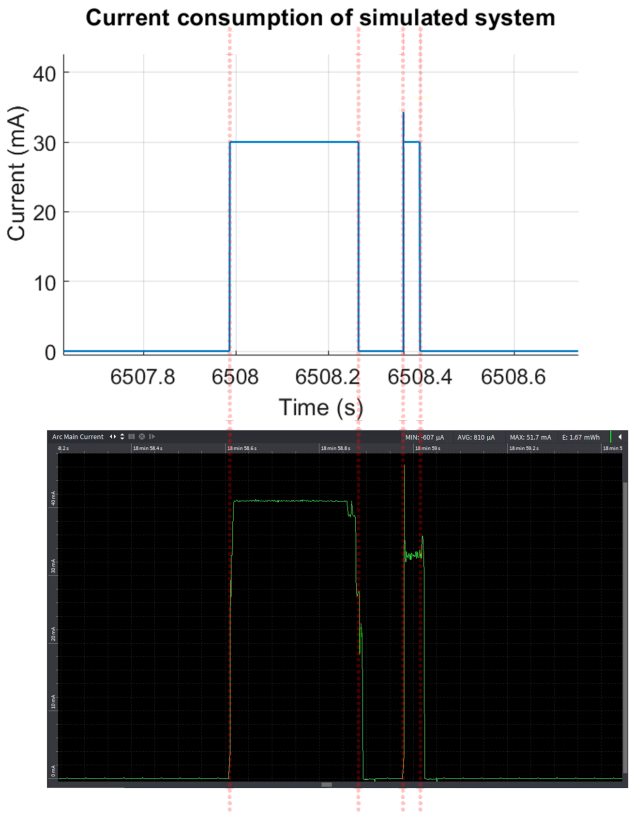

Fig. 34. Graph of current consumption of sending messages

As shown in figure 32, both the C-Process and the Gateway are capable of storing debugging messages (logs) of the entire simulation process. The simulated system stores these messages in separate text files, which are updated in real time allowing a follow-up of each step and task performed by the different components. In addition, the GVA also has debugging messages to check the reception and sending of messages to the gateway, authentication failures and any failure of the GVA itself. This mechanism allows to corroborate in detail the correct operation of each routine, subroutine or function executed in any of the components, and therefore facilitates the detection and correction of faults that are not detected in macro behaviors such as those tested in the BDD scenarios. Figure XX shows a screenshot of the simulation system running and showing the debugging messages of the system.

C. Current consumption graph

As mentioned in section VIII, a mechanism was developed to generate current graphs of the tag. The graphs obtained with this mechanism were compared with those obtained by the power analyzer in a real scenario. Figure 33 shows a comparison between the graphs for a fragment of the system simulation process, specifically a group of microIntervals. In addition, Figure 34 shows a comparison of the current graph for the data sending process in each micro interval.

At each of the intervals, the tag requests information from the sensors and then awaits the arrival of a new reference beacon to respond with information from the sensors or a live message. The duration between each micro interval is constant, for this reason it is observed that the process is repetitive with a period of 60 seconds (micro interval).

In the process of sending data in each micro interval, the tag presents two current pulses, the pulses of lower amplitude represent the process of obtaining data from the sensors, and the higher amplitude represents the transmission of the corresponding message for each interval.

The results obtained by the current graph generation mechanism allow us to observe the impact that different processes executed on the tag have. In addition, it could be used to analyze the impact on current consumption when implementing new functionalities within the tag, where the server search, association, and data sending processes are modified.

D. Bug fixes

During the development process of the automatic behavioral testing tool, the company TaiO system provided the support service to the IoT system project, receiving different reports of bugs in some functionalities. One of these bugs was related to the verification of threshold values for the configuration of the sensors, these values are used to activate interruptions or alarms in the tag sensors and thus report anomalous values during their operation. These values are configured from a web platform that verifies that the limit values are within a certain range, to finally send that information to the GVA, which is in charge of configuring the sensor network with these parameters. The error occurred due to a failure in the

```
Scenario: {1} tags are already to establish
    communication and be configured

    Given the simulated system is running in
    security mode "0"
    When the association process is executed
    And the configuration is sent by the GVA
    Then the tag should receive the configuration
    correctly
    And sent the sensor data in the next macro
    interval
```

Fig. 35. BDD Scenario implemented to correct the threshold bug

verification of these parameters, which caused a threshold value higher than the established range to be sent and finally an erratic behavior in the system. For this reason, the development team had to debug the system in order to replicate the error, and later correct it.

A single tag scenario was used, and the simulated GVA was configured to send a configuration packet with the same scenario conditions. After executing this scenario, we proceeded to review the debugging messages obtained from the different components of the simulated system, showing that there was an inconsistency between the message received by the gateway and the message sent to the server.

As in all agile development methodology based on tests, the first step is to define the test scenario, so after detecting the component that caused the error, a new scenario was designed to verify that the threshold configuration is transmitted correctly to all system components. In Figure 35 the definition of the scenario is shown.

Then the scenario was executed to verify that it failed and finally a solution was implemented that managed to pass the test.

E. Optimization of the development time

During the last phase of the internship, new development tasks were assigned to the company team in order to strengthen knowledge and familiarization with the IoT system firmware, as part of constant training. These tasks are related to the implementation of new functionalities and modification of some existing ones.

One of these development tasks consisted of modifying the sensor message forwarding process. When a tag sends a package of sensor information and the package is lost during the process, the server detects this loss and sends a reconnection beacon, then the tag stores the lost data in its flash memory to later forward it in the next macro frame. . When the IoT system operates under conditions of high noise levels, this process can be repeated frequently having to store and read information from flash memory.

The new implementation of the forwarding process was to avoid storing large amounts of data, and instead try to forward the message during the same macro frame. This process gets the tag to use non-volatile storage only when new information is retrieved from sensors.

To carry out this implementation, the development team used the automatic behavioral testing tool, mainly the scenarios that included packet loss to simulate the conditions in which this new data transfer process is executed. Carrying out the changes in the simulated system, with the possibility of debugging information in all the components and the rapid execution of the scenarios allowed this task to be completed in a shorter time than expected by the company. Moreover, the possibility of testing multiple scenarios with different percentages of packet loss and execution durations, allowed to correct errors during the implementation before carrying out tests in a real scenario. It is noteworthy that after ensuring that the new implementation passed all the BDD tests, it worked correctly in the real scenario, without the need for further changes.

The execution time of the simulation of the scenarios executed during development was 1 hour and 29 minutes, which translates into 50 hours of real execution. The executed sceanarios are shown in Table VIII.

TABLE VIII
SCENARIOS USED FOR THE NEW IMPLEMENTATION

Scenario	Repetitions	Total simulation time	Total real time
1 hour of DC	20	0.5898	20 hours
2 hour of DC	5	0.2949	10 hours
4 hour of DC	5	0.5896	20 hours

* DC: data collection

In the middle of the execution of these scenarios, changes were made in the code to correct errors present in the implemented solution to ensure its correct operation, which would have been an increasing factor in development time since the debugging, compilation, and flashing processes in the real scenario it is more complex and late.

Finally, Table IX shows a comparative table between the old and the new implementation assuming 2 minutes for macro interval, packet loss of 20 percent and the flash memory total size of 389120 bytes. It can observe that the new implementation presented a great advantage over the old one, specifically in increasing the capacity to store information in flash memory for more days of execution.

TABLE IX
ADVANTAGE OF THE NEW IMPLEMENTATION

Implementation	Max days to store
Old	57
New	287

The new implementation reduces the number of logged messages 5 times. Therefore it is capable of store data in flash during more days. Moreover, if the macro interval is greater, the probability of logging data is reduced and the number of days with available memory increases.

X. CONCLUSIONS

In this work, the design and implementation of an automatic behavioral testing system for an IoT environment for the company TaIO systems was carried out, in order to detect and

correct errors, verify compliance with system requirements, and optimize the development process. .

The simulation system implemented allowed to execute all the components of the system, synchronize their operation and communicate them between them, thanks to this it was possible to implement different test scenarios based on the BDD development strategy, which allowed the system to face different simulation conditions present in the normal operation of the real system.

The behavioral testing system was able to run simulations of real scenarios with a speed factor greater than the real system, and with the ability to retrieve debugging messages from the entire system, significantly contributing to the decrease in the time required for detection and correction. of errors as well as in the implementation of new functionalities. The system presents a decrease in the speed factor when the number of simulated tags is increased.

In addition, the implementation of the process of obtaining current graphs of the tag was able to generate results very close to the real behavior, which can contribute to the analysis of the impact of the implementation of changes in the firmware and new functionalities to the current consumption of the system.

A new multi-process simulation strategy should be implemented to improve the performance of the simulation system in the presence of large numbers of tags. This would require a restructuring of the simulation system and the use of new tools and computational resources such as GPUs.

To carry out simulated current measurements that allow obtaining statistical data such as current average in a long-duration process, it would require a modification of the simulation and synchronization system of multiple processes as well as the collection of data on the current consumption of the tag in a large quantity of tasks and subroutines.

ACKNOWLEDGMENT

I would like to express my gratitude to my supervisors, Juan Pablo Ruiz and Wilson Achicanoy, who guided me throughout this project. I would also like to thank the company TaIO systems for the opportunity to do my internship within one of its projects.

REFERENCES

[1] 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, pp. 659–676, 2018.

[2] F. Aktas, C. Ceken, and Y. E. Erdemli, "Iot-based healthcare framework for biomedical applications," *Journal of Medical and Biological Engineering*, vol. 38, no. 6, pp. 966–979, 2018.

[3] X. Chen, W. Rhee, and Z. Wang, "Low power sensor design for iot and mobile healthcare applications," *China Communications*, vol. 12, no. 5, pp. 42–54, 2015.

[4] J. Mao, Q. Lin, and J. Bian, "Application of learning algorithms in smart home iot system security," *Mathematical foundations of computing*, vol. 1, no. 1, p. 63, 2018.

[5] V. A. Kumar, G. Saranya, D. Elangovan, V. R. Chiranjeevi, and V. A. Kumar, "Iot-based smart museum using wearable device," in *International Conference on Innovative Computing and Communications*. Springer, 2019, pp. 33–42.

[6] S. Lee, G. Tewolde, and J. Kwon, "Design and implementation of vehicle tracking system using gps/gsm/gprs technology and smartphone application," in *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE, 2014, pp. 353–358.

[7] R. Jisha, A. Jyothindranath, and L. S. Kumary, "Iot based school bus tracking and arrival time prediction," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017, pp. 509–514.

[8] C. Caballero-Gil, J. Molina-Gil, P. Caballero-Gil, and A. Quesada-Arencibia, "Iot application in the supply chain logistics," in *International Conference on Computer Aided Systems Theory*. Springer, 2013, pp. 55–62.

[9] Z. Li, G. Liu, L. Liu, X. Lai, and G. Xu, "Iot-based tracking and tracing platform for prepackaged food supply chain," *Industrial Management & Data Systems*, 2017.

[10] D. Davcev, K. Mitreski, S. Trajkovic, V. Nikolovski, and N. Koteli, "Iot agriculture system based on lorawan," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–4.

[11] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, "An overview of internet of things (iot) and data analytics in agriculture: Benefits and challenges," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3758–3773, 2018.

[12] M. Syafrudin, G. Alfian, N. L. Fitriyani, and J. Rhee, "Performance analysis of iot-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing," *Sensors*, vol. 18, no. 9, p. 2946, 2018.

[13] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "Iot-based big data storage systems in cloud computing: perspectives and challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, 2016.

[14] X. Luo, L. O. Oyedele, A. O. Ajayi, C. G. Monyei, O. O. Akinade, and L. A. Akanbi, "Development of an iot-based big data platform for day-ahead prediction of building heating and cooling demands," *Advanced Engineering Informatics*, vol. 41, p. 100926, 2019.

[15] N. Nascimento, A. R. Santos, A. Sales, and R. Chanin, "Behavior-driven development: A case study on its impacts on agile development teams," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 109–116.

[16] M. Soeken, R. Wille, and R. Drechsler, "Assisted behavior driven development using natural language processing," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 2012, pp. 269–287.

[17] D. North, "Introducing behaviour-driven development. 2006," 2010.

[18] M. M. Moe, "Comparative study of test-driven development (tdd), behavior-driven development (bdd) and acceptance test–driven development (atdd)," *International Journal of Trend in Scientific Research and Development*, pp. 231–234, 2019.

[19] H. L. Tavares, G. G. Rezende, V. M. d. Santos, R. S. Manhaes, and R. A. de Carvalho, "A tool stack for implementing behaviour-driven development in python language," *arXiv preprint arXiv:1007.1722*, 2010.

[20] C. Solis and X. Wang, "A study of the characteristics of behaviour driven development," in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2011, pp. 383–387.

[21] M. Alhaj, G. Arbez, and L. Peyton, "Approach of integrating behaviour-driven development with hardware/software codesign."

[22] B. R. Jens Engel and R. Jones. (2021) Behave documentation. [Online]. Available: https://github.com/behave/behave

[23] J. M. Williams, R. Khanna, J. P. Ruiz-Rosero, G. Pisharody, Y. Qian, C. R. Carlson, H. Liu, and G. Ramirez-Gonzalez, "Weaving the wireless web: toward a low-power, dense wireless sensor network for the industrial iot," *IEEE Microwave Magazine*, vol. 18, no. 7, pp. 40–63, 2017.

[24] MQTT.org. (2021) Mqtt: The standard for iot messaging. [Online]. Available: https://mqtt.org/

[25] json.org. (2021) Ecma-404 the json data interchange standard. [Online]. Available: https://www.json.org/

[26] man7.org. (2021) Fifo(7) linux progreammer's manual. [Online]. Available: https://man7.org/linux/man-pages/man7/fifo.7.html/

[27] developer.android.com. (2021) Ndk guides - android studio. [Online]. Available: https://developer.android.com/ndk/guides/concepts

[28] ——. (2021) Intents and intent filters - android studio. [Online]. Available: https://developer.android.com/guide/components/intents-filters

[29] ——. (2021) Android debug bridge - android studio. [Online]. Available: https://developer.android.com/studio/command-line/adb

[30] ——. (2021) Logcat command-line tool bookmark border - android studio. [Online]. Available: https://developer.android.com/studio/command-line/logcat

[31] gnu.org. (2021) Gnu make. [Online]. Available: https://www.gnu.org/software/make/manual/make.html

[32] I. S. 802.15.4-2006, *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, 2006.

[33] M. Goyal, S. Prakash, W. Xie, Y. Bashir, H. Hosseini, and A. Durresi, "Evaluating the impact of signal to noise ratio on ieee 802.15. 4 phy-level packet loss rate," in *2010 13th International Conference on Network-Based Information Systems*. IEEE, 2010, pp. 279–284.

[34] S. Haykin, *Digital Communication Systems, 4th Edition*. John Wiley and Sons, 2001.

[35] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.