Towards a Thing-In-the-Loop approach for the Verification and Validation of IoT systems

Domenico Amalfitano
University of Naples Federico II
Naples, Italy
domenico.amalfitano@unina.it

Nicola Amatucci
University of Naples Federico II
Naples, Italy
nicola.amatucci@unina.it

Vincenzo De Simone
University of Naples Federico II
Naples, Italy
vincenzo.desimone2@unina.it

Vincenzo Riccio
University of Naples Federico II
Naples, Italy
vincenzo.riccio@unina.it

Fasolino Anna Rita
University of Naples Federico II
Naples, Italy
fasolino@unina.it

ABSTRACT

The Internet of Things (IoT) is rapidly increasing its diffusion, posing great challenges to the research community. IoT systems are composed by smart objects (Things) that are interconnected in order to provide new products and services. The interaction of heterogeneous and distributed smart things guided by software with the physical world brings new sources of safety issues. To this reason, providing valuable and effective solutions to support the verification and validation of such systems is needed. In this paper we introduce a model-driven Thing-In-the-Loop verification and validation approach that transfers the best practices adopted in different embedded system domains towards the IoT world. Starting from models and scenarios representing the structure and behaviors of the IoT system as well as models of its context our approach generates appropriate test cases that are executed in accordance with Model-in-the-Loop, Software-in-the-Loop and Hardware-in-the-Loop techniques. We preliminarily evaluated the feasibility of our approach by applying it in the context of a Smart Mobility case study.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; • **Software and its engineering → Software verification and validation**;

KEYWORDS

Model-driven engineering, verification and validation, Internet of Things, Thing-In-the-loop

ACM Reference Format:
Domenico Amalfitano, Nicola Amatucci, Vincenzo De Simone, Vincenzo Riccio, and Fasolino Anna Rita. 2017. Towards a Thing-In-the-Loop approach for the Verification and Validation of IoT systems. In *Proceedings of Proceedings of the First ACM Workshop on the Internet of Safe Things (SafeThings'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3137003.3137007

SafeThings'17, November 5, 2017, Delft, The Netherlands

ACM ISBN 978-1-4503-5545-2/17/11…$15.00
https://doi.org/10.1145/3137003.3137007

1 INTRODUCTION

The Internet of Things (IoT) is a paradigm that is rapidly gaining ground in both academic and industrial communities. The basic idea of IoT is the pervasive presence around us of a variety of things or objects, e.g. tags, sensors, actuators, mobile phones, just to list a few, that are able to interact with each other and cooperate with their neighbors to reach common goals [6].

Industry and research are more and more interested in IoT systems; the total number of Internet connected devices worldwide is forecast to surpass 75 billion in 2025[1].

There are a lot of contributions from the research community for the definition of new models and architectures for IoT systems, new methodologies for their development [11, 15], testbeds for supporting their testing on the field [3, 8, 16]. On the other hand, there are few contributions related to approaches for the verification and validation of IoT systems, that are crucial activities, required also for guaranteeing their safety. This aspect needs to be taken into proper account, since the IoT applications are more and more adopted in safety and mission critical systems, such as telecommunications grids, water supply chains, electrical power systems, road transportation systems, railway transportation systems, power plants, air transportation networks, public safety services, and health-care systems.

Specific techniques need to be applied across the entire development lifecycle of a IoT system in order to guarantee that it operates correctly in response to its inputs, including the safe management of likely operator errors, hardware failures and environmental changes.

Industrial international standards, such as the IEC 61508 [1], specify the techniques and the best practices that should be used for each phase of the development lifecycle of safety-critical systems. These standards leverage on a well defined V development process and define strict testing requirements because high-quality testing systems can improve products' quality, reliability, and performance. Just to give an example, in order to guarantee the safety of the system, the *IEC 61508* standard requires to execute Unit Testing of

[1] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

the system to ensure that the software is fully tested at the function level and that all possible branches and paths are taken through the software or, more thoroughly, that a MCDC code coverage criterion is achieved rather than a simple branch coverage.

Nowadays, traditional development processes are combined with Model-driven engineering (MDE) approaches, adopted to tackle the essential complexities, including safety concerns, of complex software development process [12]. Model-driven development and virtual verification are highly desirable in the development of safety-critical embedded systems because they allow to test the system in a virtual environment aiding the detection of both functional and non-functional issues in the early development stages, at different abstraction levels, before the software is actually integrated into the final hardware. More precisely, the x-In-the-Loop (xIL) simulation approach relies on the virtualization of the physical plants to test the embedded system at the model, software and hardware level respectively and, thus, achieves faster development cycles. This approach becomes more and more significant because the cost and complexity of the verification and validation process grow with the code size of the controller[20].

In this work we addressed the problem of transposing the xIL approach towards the IoT domain and proposed a novel Thing-In-the-Loop (TIL) verification and validation approach. The approach focuses on the software deployed on the Things composing an IoT system. To this aim, we had to tackle two main challenges:

- the heterogeneity of the platforms, languages and communication protocols that could be adopted for the development of the software running on each Thing composing an IoT system;
- the influence of the context on the behavior of the Thing under test.

To solve these challenges, TIL relies on: (1) an abstract representation of the test cases and of the context of the entire IoT system; (2) the generation of concrete test cases for the specific languages, hardware platforms and communication protocols of the Thing to be tested, and their execution; (3) the simulation of the context in which the IoT system is immersed during the test execution.

The remainder of the paper is organized as follows. Section 2 reports related work, whereas Section 3 describes the proposed Thing-in-the-Loop approach. Section 5 reports a case study where we preliminarily analyzed the feasibility of our approach with respect to a Smart Mobility IoT application. Finally, conclusions and future work are discussed in Section 6.

2 RELATED WORK

In the last years, the research community has focused on the different aspects related to the testing of IoT systems. Sand reports in [17] the main challenges that emerge in testing IoT systems and proposes possible solutions. The identified challenges are related to the great amount of details that need to be taken into account for testing IoT systems due to the endless number of things, processes, hardware, software. To overcome these challenges there is the need to define a comprehensive testing strategy that oversees and controls a unified testing lifecycle. Different work proposes testbeds to support the evaluation of IoT applications in real environments with real-world conditions. Examples of such testbeds are IoT-Lab [3], SmartSantander [16] and Web of Things TestBed (WoTT) [8].

Different approaches for IoT testing exploit Model-based technology. Ahmad *et al.* propose a Model-Based Testing As A Service (MBTAAS) solution for the systematic testing of data an IoT platforms [4]. It combines Model-Based Testing (MBT) technique and a service-oriented solution and was experimented on FIWARE, one of the EU most emerging IoT enabled platforms.

Abu Oun *et al.* [2] develop a Cross-Platform Scenario module that solve the problem of testing different versions of the same IoT application avoiding the need to develop/redevelop the same scenarios for each application version. Their solution is based on the separation of the testing scenarios from the application executing them on the objects. The testing scenario is defined in this work exploiting an XML representation.

An approach that can be exploited in testing IoT communication was proposed by Tappler *et al.*. Their approach is based on an active automata learning approach for testing reactive systems that can be employed for testing IoT Communication [19]. They adopted their solution for testing five freely available implementation of MQTT broker identifying 18 bugs.

Esquiagola *et al.* present a test methodology for stress testing IoT applications. In order to simulate different stress conditions they exploited a load generation tool called Tsung. They exploited the proposed methodology to evaluate the performance of their IoT platform on different hardware platforms [10].

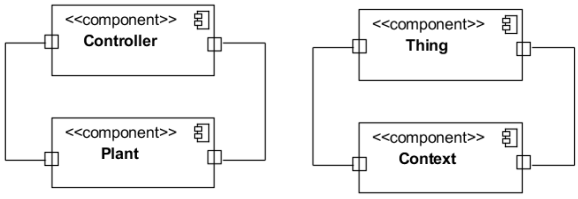
3 THE THING-IN-THE-LOOP APPROACH FOR TESTING IOT SYSTEMS

The peculiarities of IoT systems need to be taken into account for verifying and validating them. IoT systems are composed by heterogeneous and distributed Things that may interact with humans (Thing-to-Human), among each other (Thing-to-Thing) and with the environment surrounding them [13]. Things may be interconnected through a variety of different communication channels. Smart things can autonomously process information, self-configure, self-maintain, self-repair, make independent decisions, but also interact and exchange information by themselves [18]. Things may be developed and deployed on different platforms, that may feature different computational characteristics, interfaces, component and software.

Due to all these peculiarities, testing the software of IoT systems becomes an overly complicated activity and thus requires the definition of specific methodologies and approaches.

The testing of IoT systems should be carried out at different levels. Firstly, each Thing should be tested in isolation, then their integration and, finally, the entire system will have to be tested considering its Context. If we focus on a single Thing of an IoT system, the Context may abstract all the other Things, the humans, and the physical environment that are part of the overall IoT system.

The testing of a Thing can be compared to the testing of an embedded system. As shown in Figure 1(a), an embedded control system makes decisions on the basis of the feedbacks it receives from the hardware under control, i.e. the plant. Analogously, we can assume that a Thing interacts with its surrounding Context, as reported in Figure 1(b).



(a) Embedded System (b) IoT System

Figure 1: Comparison between Embedded and IoT systems

Starting from this analogy, our approach applies the xIL methodology for the verification and validation of the software of single Things belonging to IoT systems. In the xIL approach, the testing activities can be done with respect to the different artifacts produced during the development process. More in detail, *Model-in-the-Loop (MiL)* testing can be done in the early stages of development. During this phase the dynamics of the plant is captured to test the model of the controller. The controller and the plant are then simulated to verify the functionality, in the modeling framework, without any physical hardware components [21]. On the other hand, *Software-in-the-Loop (SiL)* testing targets the source code being generated from the controller models. This code is then tested with the simulated plant, without any hardware, to test how well the software can interact with the plant. Finally, *Hardware-in-the-Loop (HiL)* allows to test the software of the controller deployed on the real hardware platform while communicating with software models that simulates the plant. In this context, the controller under test, deployed on the real hardware platform, responds to simulated signals as if they were generated by the actual remaining parts of the real system [14].

Our Thing-in-the-loop approach adopts the MIL, SIL, and HIL testing stages and extends them to the IoT domain exploiting abstractions of both (1) the tests for verifying and validating the features exposed by the Things and (2) the context in which the Thing under test is immersed. The approach relies on transformation processes suitable to generate tests for the different languages and platforms on which the Thing will be deployed. Moreover, like the xIL approach, the execution of the testing process requires the simulation of the context.

The approach should be applied in a whole model-driven development process and it has to exploit development models of the software of the Things composing the IoT system. The development models should define, through a proper formalism, the structural and behavioral characteristics of the Thing.

3.1 Approach Description

As described by the Figure 2, our Thing-in-the-Loop approach consists of three main activities: Abstract Test Case Generation, Concrete Test Case Realization and Test Case Execution. The Figure shows also the artifacts that are required and produced by each activity.

The *Abstract Test Case Generation* allows the automatic generation of abstract test cases for the testing of each Thing composing the IoT system, starting from Test Models and/or from appositely defined Scenarios. An *Abstract Test Case* describes the preconditions, the actions, the expected outcomes and postconditions in an abstract way. In the Abstract Test Cases no assumption are made about the implementation details of each Thing. The Abstract Test Cases only test the abstract behavior of the Thing, without depending on concrete data encoding and a concrete message passing mechanisms. Abstract Test Cases may be formalized adopting an abstract notation. The test cases generated from the Test models are aimed at exercising all the Thing behavior according to defined coverage criteria.

The *Concrete Test Case Realization* is aimed at producing Concrete Test Cases from the Abstract ones. Concrete Test Cases should be specific for a given platform and for one of the three xIL levels of testing. In order to adapt the Abstract Test Cases to the concrete Thing under test, appropriate adapters need to be developed. More in detail, the adapter can be decomposed into a stimulus adapter and a response adapter. The stimulus adapter converts the abstract events into concrete events (message or signals), specific for the platform of the Thing Under Test. The response adapter waits for responses of the Thing under test and convert them into results that are evaluated according to the defined test oracles.

Transformations that are specific for the considered platforms and languages should be defined and implemented to enable the realization of the concrete test cases.

In the *Test Case Execution* activity, the Concrete test cases are actually executed on the target platform at the chosen level of xIL abstraction. To carry out this activity a *Context Simulator* component is needed. It is connected to the Thing Under Test and interacts with it at runtime, by producing and receiving stimuli (messages or signals). The *Context Simulator* is able to simulate *Context Models* that abstract the environment, the users, and the other Things composing the IoT system itself.

At the end of the testing execution activity, several reports about coverage and testing results are produced. More in detail, regarding the coverage, for the testing at MIL level a report about the reached model coverage is produced, whereas for the testing at SIL and HIL the obtained code coverage is reported.

3.2 Prototype Implementation Details

We developed a prototype architecture in order to support the application of the proposed Thing-in-the-Loop approach.

In our implementation, Development models of the Things are expressed as UML Component and State Machine diagrams, using the TextUML toolkit[2] textual notation. Test models should be defined using the GraphML[3] notation. They can be provided as an input or can be automatically generated from the defined development models, as reported in [5].

The transformation of Test models into Abstract Test Cases was realized exploiting the features offered by GraphWalker[4], a tool able to generate paths on state machine diagrams driven by different coverage objectives.

[2] https://abstratt.github.io/textuml/readme.html
[3] http://graphml.graphdrawing.org/
[4] http://graphwalker.github.io/

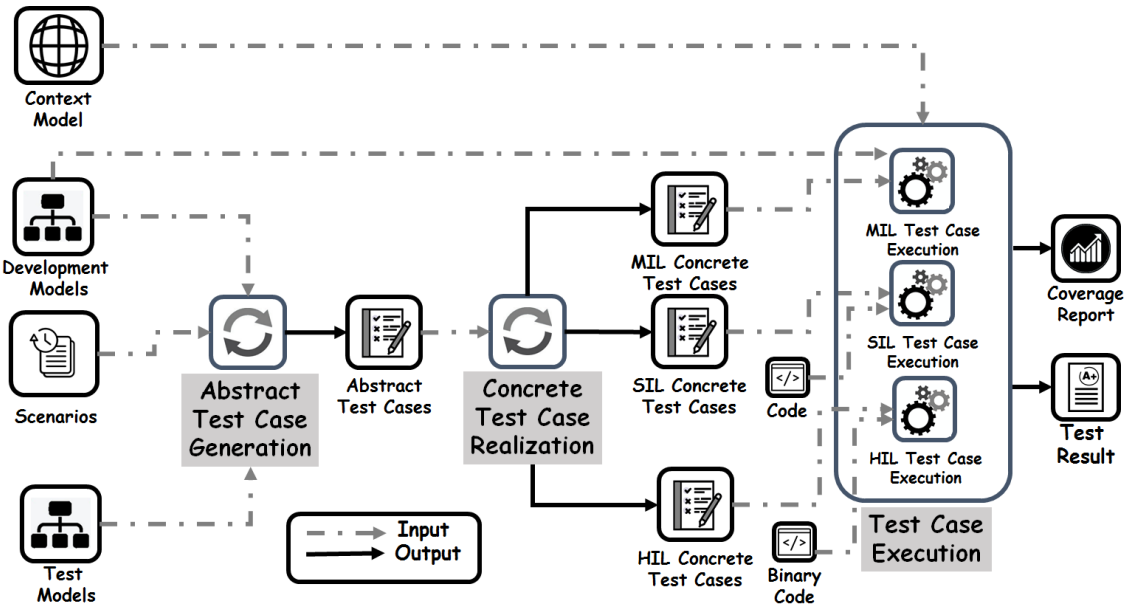


Figure 2: The Thing-in-the-Loop Testing Approach

As regards to Scenarios, instead, we relied on an instance of the Gherkin language [5], that is a domain specific language that allows an high level description of the system behavior. Each scenario is described by a triple: *Given-When-Then*. The purpose of *Given* steps is to set the precondition of the system before an external user interacts with it; *When* steps describe the action that are performed by the external user; *Then* steps, instead, allows specifying the expected outcomes. The translation of scenarios into abstract test cases, required the definition of a mapping of the high-level concepts to the one reported in the development and context models.

With respect to the Concrete Test Case Realization, we defined different transformation rules. The MIL Concrete Test Cases are expressed in txtUML [9] format. We defined specific transformation rules for translating abstract test cases into jUnit[6] Test Cases at SIL and HIL levels . For addressing the testing of Things deployed on different platforms and languages, other appropriate transformation rules should be defined.

For the Test Case Execution, we employed different Test Execution environments, specific for the platforms on which the Things were intended to be deployed. The features offered by txtUML were exploited to execute the MIL concrete Test cases. The SIL Concrete Test Cases were executed exploiting the jUnit Test Runner. As a target platform for our HIL concrete Test cases execution, we exploited the Raspberry Pi 3 board[7].

Context Models are expressed exploiting UML Timing, Sequence and State Machine diagrams, representing the physical environment, the user and the other Things composing the IoT system. We implemented a prototype Context Simulator component that is able to execute the defined context models sending and receiving stimuli (signals or messages) to and from the Thing Under Test at the different levels of xIL testing. To enable the communication among the Context Simulator and the target platform we built an hardware interface based on the Arduino board[8].

4 IMPLEMENTATION DETAILS

We developed a prototype architecture in order to support the application of the proposed Thing-in-the Loop approach.

Considering the state machine diagrams (described using the TextUML toolkit [9]) specifying the behavior of a TUT, they are translated into Abstract Test Models, as reported in [5]. **Abstract Test Cases are expressed exploiting the Abstract Syntax Notation-one [7]**. These models are represented exploiting a GraphML[10] notation and are transformed into Abstract Test Cases exploiting the features offered by GraphWalker[11] that is able to generate paths on the state machines driven by different coverage objectives. The Test Case generator also offers features for translating the provided scenarios written in the Gherkin language into *Abstract Test Cases*.

As regards to Scenario definition we relied on an instance of the Gherkin language [12], that is a domain specific language that allow expressing an high level description of the system behavior. Each scenario is described by a triple: *Given-When-Then*. The purpose of *Given* steps is to set the precondition of the system before an external user interacts with it; *When* steps describe the action that are performed by the external user; *Then* steps, instead, allows specifying the expected outcomes.

[5]https://github.com/cucumber/cucumber/wiki/Gherkin
[6]http://junit.org/
[7]https://www.raspberrypi.org/
[8]https://www.arduino.cc/
[9]https://abstratt.github.io/textuml/readme.html
[10]http://graphml.graphdrawing.org/
[11]http://graphwalker.github.io/
[12]https://github.com/cucumber/cucumber/wiki/Gherkin

In the Concrete Test Cases Realization, the MIL Concrete Test Cases are produced exploiting the txtUML [9] language. The SIL and HIL Concrete Test Case are defined according to the chosen deployment platform and language. As an example, for Java code, the Concrete SIL Test cases are defined as jUnit Test Cases. The Text Case Execution activity relies on a Context Simulator, that is able to execute the Context Model of the Thing under Test. Moreover, in the different stage of the xIL testing, it interacts with the model, the software or the hardware platform of the Thing under test.

5 A CASE STUDY ON A SMART MOBILITY IOT SYSTEM

In the following we report how we applied the proposed TIL approach for the verification and validation of a Smart Mobility IoT system.

5.1 IoT System Requirements Description

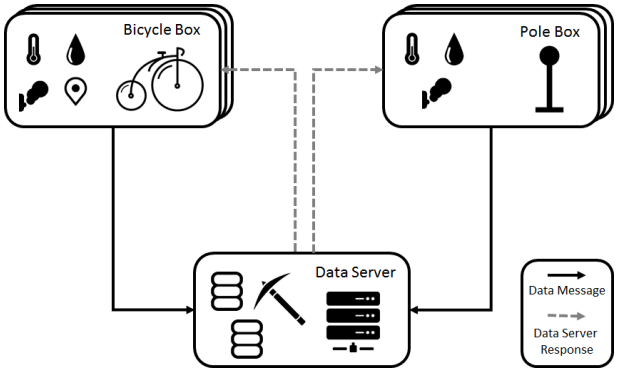


Figure 3: Smart Mobility IOT System

The Smart Mobility IoT System, reported in Figure 3, is a cooperative mobile sensing system aimed at evaluating the pollution levels of cycling routes in an urban context. It relies on a fleet of smart bicycles, a central data server and a set of smart poles.

Each bicycle is equipped with a *Bicycle Box* (BiBox) that senses the environment through temperature, humidity and air quality sensors. This data is tagged with the location acquired by a GPS receiver and periodically sent to the Data Server via a public WiFi connection. No information about the bicycle owner is sent to the Data Server, in order to protect the end user privacy. The BiBox, through the WiFi connection, also receives information about the air pollution level from the Data Server. Moreover, the Bicycle Box provides two led displays. The former shows the pollution level received by the Data Server, the latter renders a warning message when the received level is different from the one locally measured. The smart poles are equipped with a *Pole Box*, that senses the environment conditions through temperature, humidity and air quality sensors. The sensory data is tagged with its unique identifier and periodically sent to the Data Server via a public WiFi connection. The Pole Box has a display reporting the level of air pollution of the area, periodically received from the Data Server. The Data server gathers the data received from all the connected BiBoxes and Pole Boxes. This data is analyzed and fused for evaluating the air pollution level, that can assume three values, i.e., L1 for low, L2 for medium and L3 for high. The evaluated pollution level is sent back to each BiBox and Pole Box.

5.2 Application of the TIL Approach

In this section we show how the TIL approach was applied for the verification and validation of the BiBox software, developed through a model-driven development approach by a team of 4 master students. More precisely, we provide examples on how the TIL approach was exploited for testing the software implementing the control logic of the two led displays.

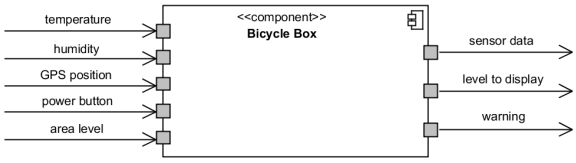


Figure 4: Bicycle Box - UML Component Diagram

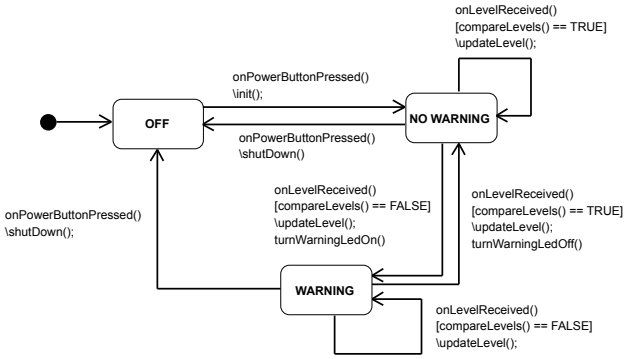


Figure 5: Bicycle Box Led Management Behavior

Figure 4 and Figure 5 report the development models of the BiBox component, in the form of UML Component and Stat Machine diagram, respectively.

Moreover, other three students, starting from the requirements of the system, defined a set of test models and scenarios to test the considered feature. Listing 1 reports one of the defined scenarios designed to test the Warning Led activation.

Listing 1: Bicycle Box Warning Led Activation Test Scenario

```
Given that the Bicycle box is powered on
      And 3 Bicycle box are inside the Area 1
      And 1 Pole Box are inside the Area 1
      And the value of air pollution in the Area 1 is L3
When an area level Message is received from the Data Server
      And the humidity value measured is 0
      And the temperature value measured is 0
      And the air quality value measured is 0
Then the L3 led should be active
      And the WARNING led should be active
```

Starting from the test models and the scenarios a set of Abstract Test Cases has been automatically produced.

Then, the produced Abstract Test Cases were translated into Concrete Test Cases for each xIL level. An example of a produced SIL jUnit Test Case is reported in Listing 2.

Listing 2: SIL jUnit Test Case Example

```
@Before
public void setUp() {
    DataServer ds = Context.getThing("DataServerStub");
    Message message = new Message();
    message.setPayload("level", 3);
    ds.setResponseMessage(message);
    Signal constant = new ConstantSignal(0);
    Context.getEnvironment().setTemperature(constant);
    Context.getEnvironment().setHumidity(constant);
    Context.getEnvironment().setAirQuality(constant);
}

@Test
public void testWarningLedShouldBeOnWhenLevelsDiffer() {
    BicycleBox b = Context.getThing("BicycleBox");
    Component lvlLed = b.getComponent("Level3Led");
    Component wLed = b.getComponent("WarningLed");
    Context.sendEvent(b, BicycleBox.EVENT_POWER_BUTTON);
    Message m = b.receive();
    assertFalse(b.getLevel() == m.getPayload("level"));
    assertTrue(lvlLed.getValue() == 1);
    assertTrue(wLed.getValue() == 1);
}
```

In order to enable the test execution, the implemented Context Simulator was opportunely configured taking into account the interfaces of the component under test, shown in Figure 4.

We executed the suite of concrete MIL test cases and evaluated both their outcome and model coverage. The test cases covered all the states and the transitions of the model and did not find any issue. The validated models where automatically translated into concrete Java source code for the Raspberry Pi 3 platform, exploiting an appropriate code generator. Then, the concrete SIL test cases were executed against the generated source code. The code was compiled on the development platform using the Oracle Java 7 JDK and instrumented using the JaCoCo library [13] to obtain a code coverage report. All the generated SIL Concrete Test Cases were not able to guarantee the coverage of all the code under test. From the analysis of the source code coverage report, it emerged that not all the conditions were covered.

In order to reach an adequate level of code coverage, other scenarios should be defined to exercise the uncovered code.

The validated source code was deployed on a Raspberry Pi 3 board that was connected to the Context Simulator component through an Arduino board in order to execute the HIL concrete Test Cases. The Thing under test showed always the expected behavior.

This simple case study showed us the feasibility of our Thing-In-the-Loop approach for supporting the verification and validation of the BiBox at different stages of the development process. The approach allows to test the considered Things even when the remaining part of the system is not available, as long as a model of the Context is defined. Moreover, it does not require to manually define specific Test Cases for each platform on which the Thing is going to be deployed.

[13] http://www.eclemma.org/jacoco/

6 CONCLUSION & FUTURE WORK

In this paper we presented a Thing-in-the-Loop approach for the verification and validation of the software of Things composing IoT Systems. The approach was defined translating the xIL approaches for embedded systems verification and validation toward the IoT domain. The approach exploits context models in order to safely test also failure and harmful scenarios, that will be dangerous to be tested on the field.

In order to show the feasibility of the approach we applied it to a Smart Mobility case study. The application of the approach allowed us to verify its Bicycle box component, allowing us to identify an issue that affected both the model and the generated code.

As future work, we plan to extend the prototype implementation of the proposed approach, extending the supported platforms and languages. Standard notations, such as Testing and Test Control Notation version 3 (TTCN-3) and Functional Mock-up Interface (FMI), will be adopted in the implementation of our approach. Specific approaches for modeling the context of IoT system in a thorough way will be provided. Moreover, features for handling traceability among the involved artifacts will be introduced in order to support safety management. We intend to extend the approach for supporting integration, interoperability, performance, and safety testing of IoT systems.

We also plan to further extend the experimentation of the approach, applying it in several safety-critical industrial case scenarios, considering more complex IoT systems.

REFERENCES

[1] IEC SC 65A. 1998. *Functional safety of electrical/electronic/programmable electronic safety-related systems.* Technical Report IEC 61508. The International Electrotechnical Commission, 3, rue de Varembé, Case postale 131, CH-1211 Genève 20, Switzerland.

[2] Osama Abu Oun, Christelle Bloch, and François Spies. 2016. *Cross-Platform Scenario Module for Internet of Things Testing Architecture.* Springer International Publishing, Cham, 385–395. https://doi.org/10.1007/978-3-319-47075-7_43

[3] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT).* 459–464. https://doi.org/10.1109/WF-IoT.2015.7389098

[4] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. 2016. *Model-Based Testing as a Service for IoT Platforms.* Springer International Publishing, Cham, 727–742. https://doi.org/10.1007/978-3-319-47169-3_55

[5] D. Amalfitano, V. De Simone, A. R. Fasolino, and V. Riccio. 2015. Comparing Model Coverage and Code Coverage in Model Driven Testing: An Exploratory Study. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW).* 70–73. https://doi.org/10.1109/ASEW.2015.18

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787 – 2805. https://doi.org/10.1016/j.comnet.2010.05.010

[7] P. T. Barry. 1992. Abstract syntax notation-one (ASN.1). In *IEE Tutorial Colloquium on Formal Methods and Notations Applicable to Telecommunications.* 2/1–2/3.

[8] L. Belli, S. Cirani, L. Davoli, A. Gorrieri, M. Mancin, M. Picone, and G. Ferrari. 2015. Design and Deployment of an IoT Application-Oriented Testbed. *Computer* 48, 9 (Sept 2015), 32–40. https://doi.org/10.1109/MC.2015.253

[9] Juan de Lara, Peter J. Clarke, and Mehrdad Sabetzadeh (Eds.). 2016. *Proceedings of the MoDELS 2016 Demo and Poster Sessions co-located with ACM/IEEE 19th*

International Conference on Model Driven Engineering Languages and Systems (MoDELS 2016), Saint-Malo, France, October 2-7, 2016. CEUR Workshop Proceedings, Vol. 1725. CEUR-WS.org. http://ceur-ws.org/Vol-1725

[10] John Esquiagola, Laisa Costa, Pablo Calcina, Geovane Fedrecheski, and Marcelo Zuffo. 2017. Performance Testing of an Internet of Things Platform. In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security*. 309–314. https://doi.org/10.5220/0006304503090314

[11] F. Fleurey and B. Morin. 2017. ThingML: A Generative Approach to Engineer Heterogeneous and Distributed Systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 185–188. https://doi.org/10.1109/ICSAW.2017.63

[12] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In *2007 Future of Software Engineering (FOSE '07)*. IEEE Computer Society, Washington, DC, USA, 37–54. https://doi.org/10.1109/FOSE.2007.14

[13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* 29, 7 (Sept. 2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[14] R. McNeal and M. Belkhayat. 2007. Standard Tools for Hardware-in-the-Loop (HIL) Modeling and Simulation. In *2007 IEEE Electric Ship Technologies Symposium*. 130–137. https://doi.org/10.1109/ESTS.2007.372075

[15] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs. 2015. FRASAD: A framework for model-driven IoT Application Development. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 387–392. https://doi.org/10.1109/WF-IoT.2015.7389085

[16] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. 2014. SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks* 61 (2014), 217 – 238. https://doi.org/10.1016/j.bjp.2013.12.020 Special issue on Future Internet Testbeds - Part I.

[17] Benny Sand. 2016. *IoT Testing - The Big Challenge Why, What and How*. Springer International Publishing, Cham, 70–76. https://doi.org/10.1007/978-3-319-47075-7_9

[18] Lu Tan and Neng Wang. 2010. Future internet: The Internet of Things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, Vol. 5. V5–376–V5–380. https://doi.org/10.1109/ICACTE.2010.5579543

[19] M. Tappler, B. K. Aichernig, and R. Bloem. 2017. Model-Based Testing IoT Communication via Active Automata Learning. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 276–287. https://doi.org/10.1109/ICST.2017.32

[20] G. Tibba, C. Malz, C. Stoermer, N. Nagarajan, L. Zhang, and S. Chakraborty. 2016. Testing automotive embedded systems under X-in-the-loop setups. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/2966986.2980076

[21] A. Vidanapathirana, S. D. Dewasurendra, and S. G. Abeyaratne. 2013. Model in the loop testing of complex reactive systems. In *2013 IEEE 8th International Conference on Industrial and Information Systems*. 30–35. https://doi.org/10.1109/ICIInfS.2013.6731950