

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**

# **Testing in IoT systems: From simulation to visual-based testing**

**Bernardo Ferreira dos Santos Aroso Belchior**



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: André Restivo

Second Supervisor: Hugo Sereno Ferreira

July 25, 2019



# **Testing in IoT systems: From simulation to visual-based testing**

**Bernardo Ferreira dos Santos Aroso Belchior**

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2019



# Abstract

The Internet of Things (IoT) is a network of uniquely identifiable devices (the things), that are continuously connected to the Internet; it is generally composed of heterogeneous programmable devices, that are capable of sensing and acting on their surroundings. IoT systems can be applied in both small-scale scenarios, e.g. smart homes, large-scale scenarios, e.g. large industrial complexes (the Industrial Internet of Things or IIoT), or even cities.

The heterogeneity of IoT systems — and also their increasing complexity and distributed nature — makes them more prone to errors; these can be caused by device failures or by mistakes in their configuration. When a factory relies on an IIoT system, having a failure may be catastrophic and cause huge losses in terms of productivity and revenue. This is why it is important to use testing tools and frameworks in order to increase the certainty levels regarding the correctness of the system.

Currently, there are multiple tools that provide testing capabilities to the IoT, ranging from unit and integration to acceptance and system testing, using different testing methods (black-, grey- and white-box testing) and focusing on different parts of an IoT system (edge, fog, and cloud testing). There are, however, a few gaps in the area of IoT simulation that would allow for the coexistence of simulated and physical devices. Particularly, there's a clear absence in the literature of solutions able to (1) test an IoT system without needing the whole physical environment, which would drive setup costs down by not having to buy devices beforehand; (2) mock a specific condition, allowing for reproducibility of rare environment states (e.g. waiting until there's an earthquake); and (3) test a system closer to reality (i.e. with physical devices that suffer from network conditions, e.g. packet losses), providing higher confidence on the correctness of the system than a simulated environment would.

Furthermore, multiple visual programming languages (VPLs) exist with the aim of simplifying and accelerating the development process. The IoT is one heavily targeted field by these tools as its usual message-based architecture fits very well in the model of VPLs. One of such tools is Node-RED, a graph-based platform for visual programming of systems comprised of hardware devices, APIs and web services, with IoT being one of its possible applications.

In order to achieve the benefits of the simulation of the IoT with the advantages of a visual programming language, this thesis will provide: (1) a simple language-agnostic tool that allows the simulation of an IoT system with both physical and virtual devices; (2) a Node-RED extension to integrate the simulator into the platform, allowing the end user to leverage the extensive community functionality that Node-RED already possesses; and (3) an extension for Node-RED to provide automated testing of IoT systems using graphical elements, supporting real-time feedback to the user and incorporating remote control and periodic running of tests.



# Resumo

A Internet das Coisas (IoT) é uma rede de dispositivos unicamente identificáveis (as coisas), que estão continuamente ligados à Internet; geralmente é composta por dispositivos heterogêneos programáveis, capazes de detetar e executar ações nos seus arredores. Os sistemas de IoT podem ser aplicados tanto em cenários de pequena escala, p.e. casas inteligentes, assim como em cenários de grande escala, p.e. grandes complexos industriais (a Internet das Coisas Industrial ou IIoT), ou mesmo cidades.

A heterogeneidade dos sistemas de IoT - e também sua crescente complexidade e natureza distribuída - torna-os mais propensos a erros; estes podem ser causados por falhas no dispositivo ou por erros na sua configuração. Quando uma fábrica depende de um sistema IIoT, ter uma falha pode ser catastrófica e causar grandes perdas em termos de produtividade e receita. É por isso que é importante usar ferramentas de teste e *frameworks* para aumentar os níveis de certeza quanto ao correto funcionamento do sistema.

Atualmente, existem várias ferramentas que fornecem a possibilidade de testar a IoT, variando desde testes de unidade e integração a aceitação e testes de sistema, usando diferentes métodos de teste (*black-*, *grey-* and *white-box testing*) e concentrando-se em diferentes partes de um sistema IoT (*edge*, *fog* e *cloud testing*). Existem, no entanto, algumas lacunas na área da simulação de IoT que permitam a coexistência de dispositivos simulados e físicos. Particularmente, há uma clara ausência na literatura de soluções capazes de (1) testar um sistema IoT sem precisar de todo o ambiente físico, o que reduziria os custos de instalação por não ter que comprar dispositivos antecipadamente; (2) falsificar uma condição específica, permitindo a reprodutibilidade de estados raros do ambiente (por exemplo, esperar até que haja um terremoto); e (3) testar um sistema mais próximo da realidade (isto é, com dispositivos físicos que sofrem dos problemas da rede, por exemplo, perdas de pacotes), proporcionando maior confiança no correto funcionamento do sistema do que um ambiente simulado.

Além disso, múltiplas linguagens de programação visual (VPLs) existem com o objetivo de simplificar e acelerar o processo de desenvolvimento. A IoT é um campo altamente visado por essas ferramentas, uma vez que sua arquitetura usualmente baseada em mensagens se encaixa muito bem no modelo de VPLs. Uma dessas ferramentas é o Node-RED, uma plataforma baseada em gráficos para programação visual de sistemas compostos de dispositivos de hardware, APIs e serviços da Web, sendo a IoT uma das suas possíveis aplicações.

A fim de obter os benefícios da simulação da IoT com as vantagens de uma linguagem de programação visual, esta tese fornecerá: (1) uma ferramenta simples e agnóstica a nível da linguagem de programação que permite a simulação de um sistema de IoT com dispositivos físicos e virtuais; (2) uma extensão Node-RED para integrar o simulador na plataforma, permitindo que o utilizador final aproveite a ampla funcionalidade providenciada pela comunidade que o Node-RED possui; e (3) uma extensão para o Node-RED para fornecer testes automatizados de sistemas de IoT usando elementos gráficos, suportando *feedback* em tempo real para o utilizador e incorporando controlo remoto e execução periódica de testes.





# Acknowledgements

First, I would like to express my appreciation towards my supervisor André Restivo and second supervisor Hugo Sereno Ferreira for guiding me, correcting my wrongs and helping me learn throughout the writing of this document. I would also like to thank João Pedro Dias for the assistance given, even though he was not required to, which highlights his great character.

I would also like to thank my parents, my sister and my family who have always been very supportive and understanding so that I could be where I am now. Without their encouragement and incentives, I fear my life would not be nearly as successful. Moreover, I would like to express my gratitude towards all my friends, as they have always presented me with good moments and excellent memories, helping me push forward and become a better version of myself.

Finally, I would like to thank everyone who happened to cross my life, as it has made me who I am today.

Bernardo Belchior



*“Human felicity is produc’d not so much by great pieces of good fortune  
that seldom happen, as by little advantages that occur every day”*

Benjamin Franklin



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Motivation . . . . .	2
1.3	Problem Definition . . . . .	3
1.4	Goals . . . . .	3
1.5	Document Structure . . . . .	4
<b>2</b>	<b>Fundamental Concepts</b>	<b>5</b>
2.1	Internet of Things . . . . .	5
2.2	Testing . . . . .	7
2.3	Simulation of the Internet of Things . . . . .	7
2.3.1	Common Architectures . . . . .	8
2.3.2	Important Concepts . . . . .	8
2.4	Visual Programming . . . . .	8
2.5	Summary . . . . .	10
<b>3</b>	<b>State of the Art</b>	<b>11</b>
3.1	Systematic Literature Review . . . . .	11
3.1.1	Methodology . . . . .	11
3.1.2	Results . . . . .	14
3.1.3	Expanded Search . . . . .	20
3.1.4	Analysis & Discussion . . . . .	22
3.1.5	Conclusions . . . . .	26
3.2	Related Work . . . . .	26
3.3	Visual Programming Languages . . . . .	28
3.3.1	VPLs for the Internet of Things . . . . .	29
3.4	Summary . . . . .	31
<b>4</b>	<b>Problem Statement</b>	<b>33</b>
4.1	Current Issues . . . . .	33
4.2	Desiderata . . . . .	34
4.3	Scope . . . . .	34
4.4	Use Cases . . . . .	35
4.5	Research Questions . . . . .	36
4.6	Validation . . . . .	37
4.7	Summary . . . . .	37

## CONTENTS

<b>5</b>	<b>Solution</b>	<b>39</b>
5.1	Simulator for the Internet of Things . . . . .	39
5.1.1	Architectural Overview . . . . .	39
5.1.2	Component Characterization . . . . .	42
5.2	Node-RED IoT Simulation . . . . .	45
5.2.1	Component Characterization . . . . .	46
5.3	Node-RED Testing Framework . . . . .	47
5.3.1	Solution Overview . . . . .	47
5.3.2	Custom Nodes . . . . .	47
5.3.3	Limitations . . . . .	50
5.4	Summary . . . . .	51
<b>6</b>	<b>Evaluation</b>	<b>53</b>
6.1	Test Scenarios . . . . .	53
6.1.1	Scenario A . . . . .	53
6.1.2	Scenario B . . . . .	54
6.2	Assessment . . . . .	55
6.3	Research Questions . . . . .	56
6.4	Conclusions . . . . .	57
<b>7</b>	<b>Conclusions</b>	<b>59</b>
7.1	Difficulties . . . . .	59
7.2	Contributions . . . . .	59
7.3	Conclusions . . . . .	60
7.4	Future Work . . . . .	60
	<b>References</b>	<b>63</b>
<b>A</b>	<b>Systematic Literature Review</b>	<b>71</b>

# List of Figures

1.1	Distribution of bug impacts. . . . .	2
2.1	The various domains the Internet of Things can be applied to. [ <a href="#">Ray18</a> ] . . . . .	6
3.1	CPS simulator architecture by Li et al. [ <a href="#">LWZ<sup>+</sup>13</a> ] . . . . .	15
3.2	KhronoSim high-level architecture. [ <a href="#">CMB<sup>+</sup>18</a> ] . . . . .	17
3.3	COOJA can run at multiple simulation levels. [ <a href="#">ÖDE<sup>+</sup>06</a> ] . . . . .	21
3.4	OMNeT++ Internal Architecture. [ <a href="#">VH08</a> ] . . . . .	21
3.5	Publications per year of IoT simulators. . . . .	25
3.6	Data oriented architecture.[ <a href="#">CMB<sup>+</sup>18</a> ] . . . . .	28
3.7	Node-RED UI. . . . .	29
3.8	<i>Noodl</i> web interface. [ <a href="#">noo</a> ] . . . . .	30
3.9	Zenodys web interface. [ <a href="#">zen</a> ] . . . . .	31
5.1	One Queue Architecture. . . . .	40
5.2	Two Queue Architecture. . . . .	41
5.3	Node-RED IoT Simulation Extension. . . . .	45
5.4	Node-RED Testing Framework running tests. . . . .	48
5.5	<i>Mock</i> node configuration interface. . . . .	49
6.1	Test Scenario A. . . . .	53
6.2	Test suite for Scenario A. . . . .	54
6.3	Test Scenario B. . . . .	54
6.4	Subset of the test suite for Scenario B. . . . .	55

## LIST OF FIGURES



# List of Tables

3.1	Search results per database . . . . .	13
3.2	Publications per step . . . . .	14
3.3	Simulators and their properties . . . . .	19
3.4	Simulators and their properties . . . . .	23

## LIST OF TABLES

# Abbreviations

DDS	Data Distribution Service
DOA	Data Oriented Architecture
E2E	End-to-end
FIFO	First-in first-out
IDE	Integrated Development Environment
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
RFID	Radio-frequency identification
ROM	Read-only memory
SUT	System Under Test
VPL	Visual Programming Language
WSN	Wireless Sensor Network



# Chapter 1

## Introduction

This chapter introduces the dissertation by revealing its context, reach and structure. Section 1.1 serves the purpose of detailing the context of this publication. Section 1.2 (p. 2) explains why it is important to focus on these areas of work. Subsequently, the problem intended to solve is defined in Section 1.3 (p. 3), while the goals of this dissertation are described in Section 1.4 (p. 3). Finally, an explanation of the structure of the publication is provided through the means of Section 1.5 (p. 4).

### 1.1 Context

The Internet of Things is a paradigm where computational devices ("things") are connected to the Internet and are uniquely identifiable and globally accessible [MBR15]. This system of smart devices is built on different hardware, with distinct software and dynamic networks. This heterogeneity allows a great level of combinations for all kinds of purposes. This explains why, currently, the IoT has an extensive set of domains that it can be applied to, ranging from smart homes and smart cities to the Industry 4.0, from intelligent fleet management to agriculture, from smart grids to IoT-assisted hospitals [CXL<sup>+</sup>14].

This extreme level of applicability may explain why the IoT has been growing massively. In 2016 there were between 6.4 billion and 17.6 billion devices (excluding smartphones, tablets and computers) connected to the Internet and it is expected that, by 2020, the number of IoT devices will be in the range of 20-30 billion [Nor16].

IoT systems are comprised of a huge plethora of different devices, ranging from low-end low-power cheap microcontrollers to high-tech extremely performant computers. This heterogeneity intrinsic to IoT systems in conjunction with the quantity of devices raises several issues, akin to those distributed systems face, but enhanced given the particularities of the hardware beneath. Li et al. [LXZ15] identifies some of these problems, namely the huge array of communication protocols to choose from, security and privacy concerns and the lack of standards.

Visual Programming Languages (VPLs) is the adaptation of regular textual programming languages into the visual realm by providing graphical abstractions [JKBL14]. VPLs are usually based on boxes and arrows to indicate the flow of the program and their uses range from education and multimedia to video games and automation. The most popular is Scratch, the educational language created by the Massachusetts Institute of Technology that is based on pieces that are joined together and follow a specific flow [MRR<sup>+</sup>10].

## 1.2 Motivation

Almost every device we have is run by software, be it mobile phones, computers, cars, ATMs, etc. [Cha05] The amount of software that is present in the life of an individual is astonishing, and so are the implications caused by its failure. Most IT experts agree that software failures happen much more often than they should [Cha05]. Charette [Cha05] presents a list of software errors that resulted in losses of billions of dollars prior to 2005.

The Internet of Things augments the problems of software failure since it extends the range of concepts that one must have in mind, e.g., different devices, communication protocols. Furthermore, the large scale of the IoT also acts as a multiplier to the number of issues that can arise, due to the big increase in software complexity. Such scale makes IoT simulation a field to be explored due to its potential benefits regarding cost savings and time-to-prototype reduction. Moreover, testing can also become an interesting path to traverse, as it offers more confidence on the correct behavior of a system, reducing the probability of catastrophic failures, such as NASA's \$125 million lost satellite due to different teams using English units of measurements and the metric system without conversion [Llo].

In fact, according to [TLL<sup>+</sup>14], most software bugs are caused by faulty implementations of features that do not meet the pre-defined requirements. Automated testing can prove itself useful in such circumstances as unexpected scenarios can be reproduced and validated against.

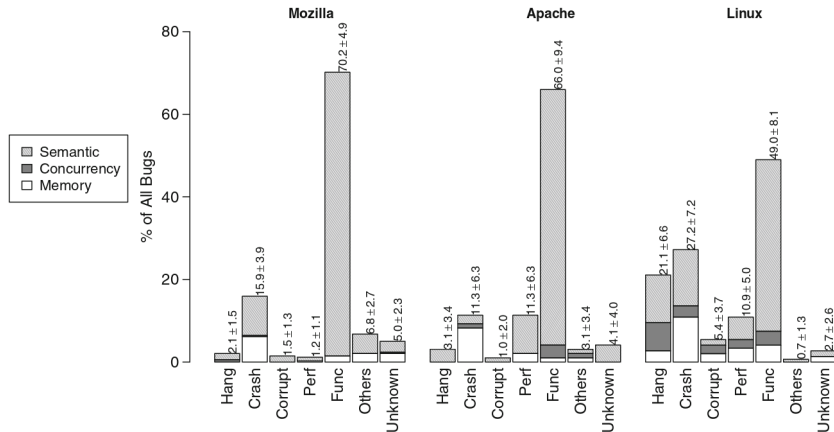


Figure 1.1: Distribution of bug impacts from a subset of bugs filed to Mozilla, the Linux kernel and Apache. *Hang* refers to the program hanging; *Crash* to the program crashing; *Corrupt* to data corruption; *Perf* to performance problems and *Func* to unexpected behavior. [TLL<sup>+</sup>14]

In addition, given the physical reality of the IoT, it is possible that a system's state may not repeat in a reasonable time frame (e.g., waiting for an earthquake to happen). In such cases, having a simulation platform to test edge cases before they happen may prove very useful in anticipating possible issues that would have manifested themselves in a real-world scenario.

### 1.3 Problem Definition

The growth of the Internet of Things coupled with the fact that multiple systems have different architectures, distinct devices and form unique networks increases the complexity of the system as a whole. In a smart home scenario, the number of devices is relatively small and the interaction between them may not be too hard to imagine in one's head; however, a smart city, with device count by the thousands, developers may have a more difficult time testing and validating the correct functioning of the system. There are numerous tools that provide simulation and testing capabilities, although most focus on specific domains, ranging from performance measurement or power monitoring to accurate network modeling. Some frameworks support the integration of hardware into the simulation itself, which is a great feature for developers, as it helps bridge the gap between simulated and real environments. However, none of those tools are free nor open-source. Furthermore, the lack of testing tools that can be automated also constitutes a deficiency in this space.

Chapter 4 (p. 33) explains in bigger detail the problem identified while also including a *desiderata*, defining the scope of the issue, laying out use cases and identifying the research questions.

### 1.4 Goals

The first goal of this dissertation is to lower the barrier of building and integrating IoT systems, while also providing the benefits of simulation, such as lower costs and faster feedback loop. The second objective focuses on creating a way that leverages visual programming languages to not only build IoT systems but also to develop a robust test suite to validate the system against.

With such an objective in mind, a solution that supports hardware to be integrated in the simulation will be devised. It should also allow physical nodes to be simulated for an arbitrary amount of time, without restarting the simulation. Moreover, the solution should be loosely coupled from the devices, meaning that the software running on a device (either virtual or not) can be written in any language that supports the messaging protocol in use. Also, an automated testing approach for the IoT will be devised and developed that will be integrated into a visual programming environment, with support for continuous regression testing.

With the development of this solution, it is expected that the cognitive load suffered to develop an IoT system will decrease and that integration between physical and virtual environment will become simpler, without losing the benefits of IoT simulation, in particular lower development costs and faster feedback loop. The advantages provided by automated testing shall also reduce

behavioral errors that emerge during the development process and help monitor the system while running.

### 1.5 Document Structure

This dissertation is structured and divided into chapters agglomerating different parts of its creation process. Chapter 2 (p. 5) introduces the background for the current document, along with important concepts for the full understanding of this dissertation. Chapter 3 (p. 11) details the state of the current ecosystem of simulation for the Internet of Things through a systematic literature review and outlines the best solutions found according to the defined criteria. The chapter is complemented by the current state of the state of existing visual programming languages. Chapter 4 (p. 33) explains the problem this project is solving and the approach taken to resolve it. It also contains a list of use cases to fulfill. Chapter 5 (p. 39) details the implementation efforts done, describing the choices and solutions chosen for each problem stated before. Chapter 6 (p. 53) validates the solution implemented and assesses to which extent the research questions were answered and the use cases fulfilled. Chapter 7 (p. 59) concludes the dissertation by presenting a summary of the advancements made and detailing its successes and future work.



## Chapter 2

# Fundamental Concepts

This chapter describes in large detail the crucial foundations regarding simulation tools for the Internet of Things. Section 2.1 establishes a background of the Internet of Things and several important concepts in that field. Subsequently, Section 2.2 (p. 7) details the area of testing, including dividing it into sub-types and characterizing them. Section 2.3 (p. 7) explains some notions regarding simulation of the Internet of Things. Section 2.4 (p. 8) mentions visual programming languages and explaining their use, benefits and disadvantages. Ultimately, Section 2.5 (p. 10) summarizes the findings of the previous sections.

### 2.1 Internet of Things

The Internet of Things is a paradigm where computational devices ("things") are connected to the Internet and are uniquely identifiable and globally accessible [MBR15]. This new paradigm can be extensively applied to an immense amount of different domains, such as agriculture, social networks, smart cities, healthcare, etc. A more in-depth scheme is available in Figure 2.1 (p. 6).

From a technical standpoint, IoT systems are usually divided into three separate layers that increase abstraction over the previous ones [SKH18]:

1. **Perception layer** is the lower-level layer that collects data from the environment. This is where the temperature sensors, scales and RFID readers are contained;
2. **Network layer** allows for a higher level abstraction for the higher layers which do not have to understand the underlying network specifics, such as communication protocols and heterogeneous devices;
3. **Application layer** is the highest layer that connects the different functions the system provides and interfaces with the end user.

## Fundamental Concepts

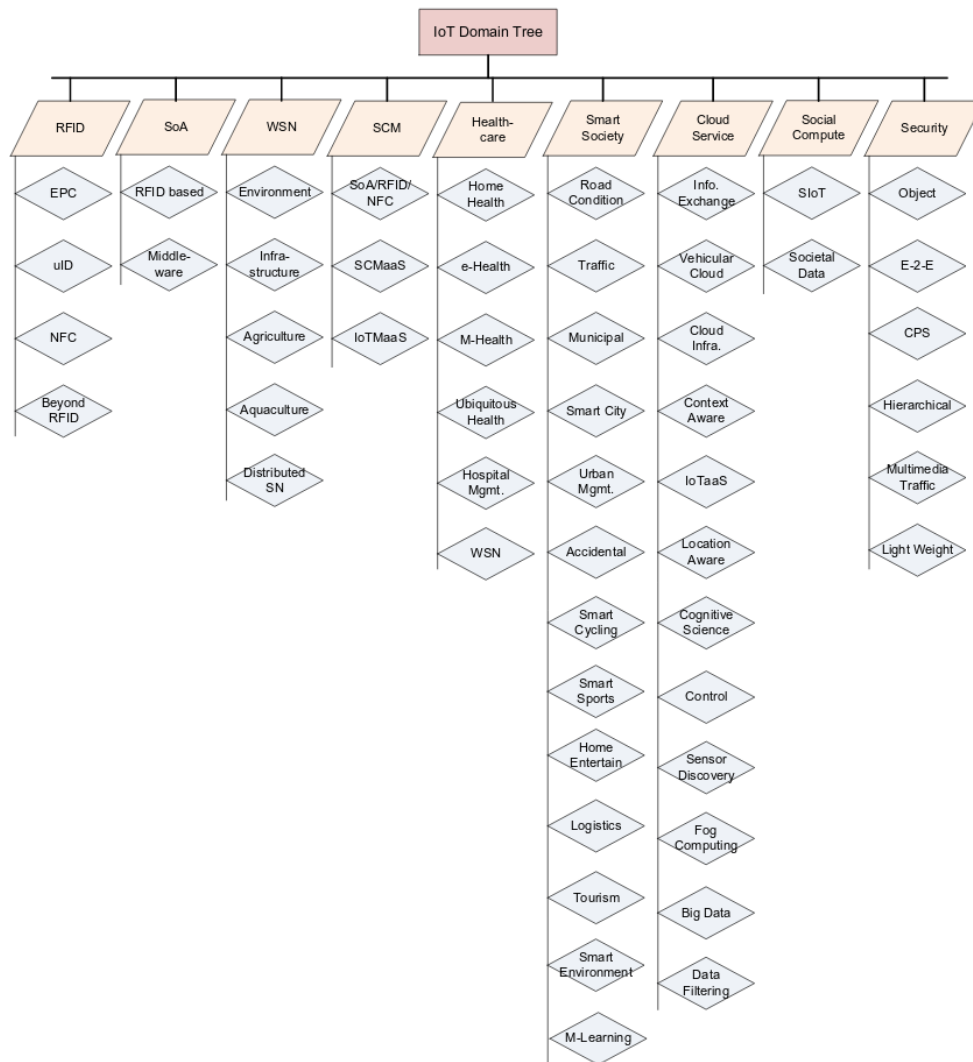


Figure 2.1: The various domains the Internet of Things can be applied to. [Ray18]

Although there are three layers, not every device needs to belong to one. It is possible that a constrained device be a part of the three above-mentioned layers, but a higher-end computer, such as a server, only belongs to the network and application layer.

## 2.2 Testing

Testing, specifically software testing, is the process of ensuring that said software does not behave in manners not expected by its developer. The importance of testing comes from the possible catastrophic accidents that can happen, such as a plane crash due to integer overflows or the loss of a \$125 million satellite due to unit conversion errors [Llo]. Testing can help prevent these disasters by reducing the number of error scenarios that were not accounted for. In order to achieve this, there are various types of test that can be executed:

- The purpose of **unit tests** is to verify small units of code and their correct functionality. They are usually fast, which is why, according to the test pyramid [Coh09], they should be the most numerous in a software test suite;
- **Integration tests** test the interaction between different and separate components. They usually take longer than unit tests. On the other hand, they provide more confidence to the tester that a specific set of components are working. These should appear in less quantity than unit tests due to their slowness;
- **End-to-end tests**, sometimes also called *system tests*, provide the most confidence to the tester as they test the whole system. These are more often than not slow and costly, therefore, their usage should be limited to critical paths of execution in a program.

While most of these tests can be executed manually, automated testing is becoming more and more widespread between the software development community, mainly due to its efficiency. Using automated tests, the developer does not need to keep testing the system after every change, and can instead rely on an external program to repeat the steps the developer would have taken. Besides, by having automated tests, more use cases can be tested after each change, possibly discovering faults the developer could not have thought about.

## 2.3 Simulation of the Internet of Things

The Internet of Things is generally comprised of heterogeneous embedded devices running on different hardware, distinct operating systems and with various peripherals. This diversity in conditions, along with the scale of IoT systems create some challenges that are left unanswered, especially regarding standardization and security. As a result of the growth of IoT, simulation is a promising field and currently being explored by many researchers. Its potential benefits, such as cost savings and time-to-prototype are, obviously, seen as desirable. By simulating the Internet of Things it is possible to test systems without actually buying the hardware, and there is also

a smaller feedback loop when compared to using real hardware, providing the aforementioned advantages.

### 2.3.1 Common Architectures

There are several architectures in common among IoT simulators. In this field, two seem to be more prominent than others:

**Multi Agent System (MAS)** is an environment where single, autonomous entities, called agents, act and interact with each other. Benefits of this architecture include autonomy for each agent and decentralization of a system. This model fits well with the IoT principle, where each device is uniquely identifiable;

**Discrete Event Architecture** is driven by events and the state of the system is only known by deriving it from a series of events. It is often used in conjunction with state machines, where a new state is the result of a function taking the current state and an event. This maps well to the IoT concepts as an Internet of Things system can be seen as a stream of data (events) being collected sequentially.

### 2.3.2 Important Concepts

There are some important concepts to grasp before fully understanding IoT simulation. Terms and expressions such as "hardware-in-the-loop simulation" and "multi-level simulation" are explained in the following section:

**Hardware-in-the-loop** simulation is a type of simulation is used to test physical devices. It tricks the hardware into thinking that it is actually part of a system, either by connecting the hardware to the simulator and sending electrical signals [nih] or by simulating how the physical system would communicate with the device under test [Bac05];

**Multi-level simulation** is a paradigm where different devices can be simulated at different levels of detail [CBBZ18]. One device could have its application layer simulated, but another could have not only its application layer simulated but also its network and/or physical layer. This idea is useful in large-scale simulations where it is not feasible to simulate every aspect of every node with the highest level of detail, so instead one could opt to simulate a smaller part of the system with finer detail. A multi-level simulation tool provides the user with the option to adjust the trade-off between simulation speed and simulation correctness.

## 2.4 Visual Programming

*Visual Programming* consists of using graphical elements such as blocks, arrows or symbols to create programs, replacing the default textual programming languages, which use text as their means to build a program [CRV15]. The goal of a visual programming language is to ease the

cognitive effort of programming, both for new and experienced users. Since this objective cannot be achieved by merely substituting text statement with pictorial components on a one-to-one basis, VPL designers were incentivized to create better visual abstractions. According to Burnet et al. [BBB<sup>+</sup>95], the employment of these modern visual techniques may foster one or more of the following characteristics:

**Simplicity** Simplicity refers to reducing the number of concepts the programmer must understand and remember in order to develop a correct program, e.g., pointers, variable declaration, etc.;

**Concreteness** Concreteness is the user's possibility to explore, visualize and modify data values;

**Explicitness** Explicitness is achieved when the relationship between components is explicitly defined;

**Responsiveness** Responsiveness, in the context of VPLs, represents the instant feedback given to the developer, where the edition of a program automatically recomputes the affected values and displays them, without having an explicit compilation step.

Due to the similarity between textual and visual programming languages, classification parameters can be applied to both, such as whether a language is imperative, declarative, functional, etc. Nonetheless, there are also some properties specific to VPLs, namely whether a language is spreadsheet-based or centered around data-flow. Burnett et al. [BB94] construct a classification system for visual programming languages that characterize them according to paradigms, visual representations, language features, etc. However, a higher-level classification of VPLs is presented by Downes et al. [DB97]:

**Purely Visual Languages** are characterized by not having an underlying text syntax and are compiled based only on their visual appearance;

**Hybrid Systems** function by mixing textual and visual representations and can be converted from one type to the other. Some existing tools [nod, min] leverage already existing programming languages compilers (or runtimes) and translate the pictorial diagrams into code;

**Programming-by-Example Systems** focus on extracting the goal of the program by generalizing examples. Adobe's Photoshop Action system [psa] is an illustration of a programming-by-example system;

**Constraint-Oriented Systems** is based on constraints defined by the programmer. It is commonly utilized in simulation design where objects are modeled in a way similar to reality by creating constraints that simulate physical laws [DB97];

**Form-based languages** are defined as cells that are connected to others and influence the state of each other over time. Spreadsheet applications, such as Microsoft Excel, are a prominent example of form-based languages.

It is important to note that these characteristics are not mutually exclusive, which implies that languages may be classified as a combination of the above features.

## 2.5 Summary

This chapter introduces very important concepts for the full understanding of this document. The Internet of Things is explained, its domains are exposed and a general architecture is presented. Software testing is then introduced along with its sub-types and characteristics. Subsequently, the simulation of the IoT is described, along with commonly occurring patterns and terms such as *hardware-in-the-loop* and *multi-level simulation* are thoroughly detailed. Finally, some background on visual programming languages is given to the reader.

## Chapter 3

# State of the Art

This chapter describes in large detail the state of the art of simulation tools for the Internet of Things. Section 3.1 presents a systematic review of the IoT simulation literature, which focuses on answering the research questions defined in Section 3.1.1.1. Subsequently, Section 3.2 (p. 26) explores the related work that is being developed in the field of IoT simulation. Section 3.3 (p. 28) analyzes the state of the art of Visual Programming Languages and its marriage with the field of IoT. Ultimately, Section 3.4 (p. 31) summarizes the findings of the previous sections and offers an overview of the current state of the field.

### 3.1 Systematic Literature Review

A systematic literature review was conducted with the intent of reliably gathering information on the state of the art of the IoT simulation field. This kind of review differs from a regular literature review due to the fact that there is a pre-established procedure to follow, which should help reduce the reviewer bias [KC07]. This idea comes from the fields of medicine and sociology, where evidence-based reviews are already established and is being modified and adapted to Software Engineering [KPB<sup>+</sup>10].

#### 3.1.1 Methodology

In this systematic review, we follow a specific and objective methodology to reduce bias [KC07]. The first step taken was to define the survey research questions to answer and the data sources to utilize. Afterward, the search query was constructed, and the inclusion and exclusion criteria laid out.

##### 3.1.1.1 Survey Research Questions

In this work, we intend to address the following questions:

**SRQ1. What relevant IoT simulation platforms exist?** IoT simulation can provide several benefits by not requiring devices to be physically present in an environment to test a system. Bearing this in mind, having a collection of simulation tools to choose from can help reduce costs, development time and the number of bugs in an IoT system;

**SRQ2. Of these, which ones support testing and validation with hardware-in-the-loop?** The lack of physical devices can be advantageous for prototyping phases, however it removes some of the confidence of testing using real devices, as simulations usually do not mirror the physical reality with 100% accuracy. Thus, having the possibility to include real hardware in the validation procedure can prove beneficial in reducing unexpected behavior and other kinds of issues;

**SRQ3. What is the scope of the tools found in SRQ1?** An IoT system is usually created with a specific use case in mind (e.g., smart cities, smart homes, IIoT), hence it is important to identify the scope of each tool so that it is easier to understand which tools should be used under different circumstances;

**SRQ4. Of those found in SRQ1, which ones support the automation of tests?** Automated tests are a practice widely used in the software world that removes the burden of manually testing the system from the developer. This shift of testing from humans to machines provides a higher confidence degree of correctness, as tests can be executed more regularly and more extensively, while also freeing time from the programmers to work on higher-priority matters.

In this context, *relevant* means publications not excluded by the methodology explained in the next sections. By *scope*, we mean the specific use case, if any, that each tool focuses on. For abbreviation purposes, *simulation* refers to any kind of simulation, emulation, or another type of approach that allows the test and validation of hardware/software IoT systems.

#### 3.1.1.2 Publication Databases

For this research, publications were retrieved from the following databases, which are regarded as good sources for software engineering [PVK15]:

- IEEEExplore Digital Library
- Scopus
- Compendex

#### 3.1.1.3 Search Process

A search query was created to narrow down the results. It was designed to include the field of research, Internet-of-Things, combined with the more specific part the paper will review: simulation. The search expression used was:



## State of the Art

```
(internet-of-things OR IoT OR "Internet-of-Things") AND (simulator  
OR (simulation AND (tool OR library OR framework)))
```

The search was made in December 2018 and revealed the results available in Table 3.1. The downloaded results were the ones deemed "most relevant" by each of the publication databases.

Table 3.1: Search results per database

Databases	Filters	Total Results	Downloaded Results
Compendex	Subject/Title/Abstract, English only	3962	500
Scopus	Title, Abstract, Keywords	1510	400
IEEEExplore	All	1422	500

### 3.1.1.4 Inclusion Criteria

Publications are put through the inclusion criteria to define whether or not they should be included in the results. If a publication does not meet *all* of them, it should not be included. The inclusion criteria are the following:

1. Original research study (including patents and grey literature for completeness);
2. Review papers;
3. Publications of code testing simulation platforms for the Internet-of-Things;
4. The study content must be in English.

It should be noted that, although review papers are included, they are processed differently. After reviewing the simulation tools retrieved from this systematic review process, the surveys will be studied to include other simulators that were not covered by the methodology. This *expanded search* allows a critical analysis of the selection process.

### 3.1.1.5 Exclusion Criteria

Papers are filtered by understanding whether or not they violate any of the exclusion criteria. If they do fail to comply with one criterion, they will be excluded. The exclusion criteria are defined as follows:

1. Secondary research and other non-relevant publications;
2. Publications presenting just ideas, magazine publications, interviews, and discussion papers;
3. Duplicate publications;
4. Publications on the topic of Internet-of-Things testing whose only focus is security, privacy, power efficiency or performance.

### 3.1.1.6 Selection Process

The publication selection process follows three steps to obtain the final result:

1. Check if the publication's title meets the inclusion and exclusion criteria;
2. Verify if the publication's abstract meets the inclusion and exclusion criteria;
3. Read the whole content of the paper and analyze whether or not it meets the inclusion and exclusion criteria.

The results of each phase can be seen in Table 3.2.

Table 3.2: Publications per step

Step	No. of Publications	Excluded
Search	1400	N/A
No Duplicates	1206	194
Inclusion/Exclusion Criteria (Title)	62	1144
Inclusion/Exclusion Criteria (Abstract)	32	30
Specificity	20	12

### 3.1.2 Results

Of the 20 selected publications, [JJW17] and [CBBZ18] were surveys on simulators. Jung et al. [JJW17] surveys various IoT simulation frameworks in search of dynamic (i.e., possibility to integrate new entities during the simulation) and decentralized tools that allow for heterogeneous simulation (i.e., simulation using different simulators) of devices. After finding none, a conceptual agent-based solution is proposed with hardware-in-the-loop capabilities. Chernyshev et al. [CBBZ18] perform an in-depth review of simulators and testbeds for the Internet-of-Things with each tool being assessed following several criteria such as the evaluated scale of the simulated system, last update to the tool, built-in IoT standards, etc. Since simulators are generally specialized in one aspect of IoT simulation, the authors conclude that a combination of the distinct IoT simulation tools may provide simulation with higher fidelity [CBBZ18]. However, this review takes on the challenge of identifying the different types of tests and programming languages that each simulator supports, while also analyzing if they can run automated testing and to integrate hardware-in-the-loop.

The remaining 18 publications were simulation frameworks, 4 of which refer to the same tool, producing 15 unique results. These are:

1. The cyber-physical system (CPS) simulator proposed by Li et al. [LWZ<sup>+</sup>13] uses the concepts of physical, computation, and interaction entities to achieve simulation of a CPS. A

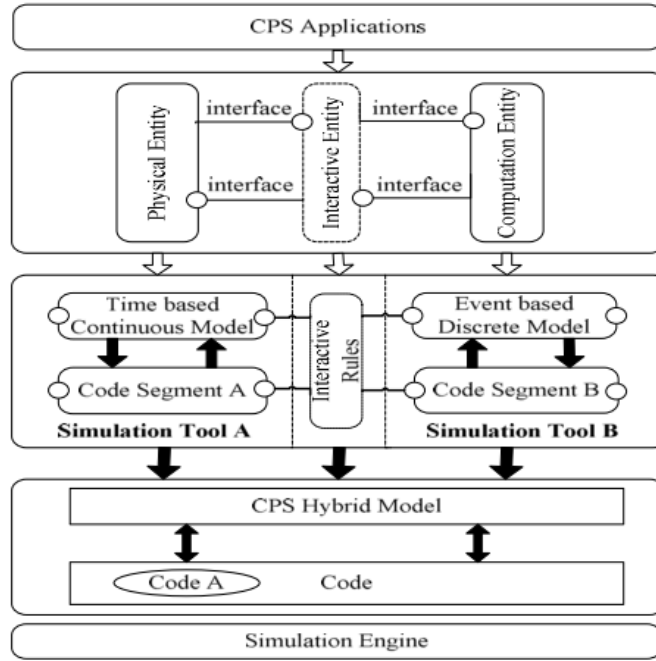


Figure 3.1: CPS simulator architecture by Li et al. [LWZ<sup>+</sup>13]

*physical entity* represents the physical aspect of a device, with spatial and temporal awareness, and is usually based on laws of physics. It is modeled using Simulink [sim]. Computation entities are modeled as finite state machines using UML in IBM Rational Rhapsody [ibm]. There is also an *interaction entity* that works as the interface of the two other entities, effectively allowing them to communicate with each other. Every component is simulated separately, allowing for model heterogeneity, but there is no mention of support for the integration of real and simulated devices. IBM Rational Rhapsody supports OXF [rha], allowing C++ code not generated by Rhapsody to be used in the simulation. Because of this, C++ can be considered as a supported language;

2. **COSSIM** [BTN<sup>+</sup>18] is an IoT emulator capable of handling networks, processors and peripherals, while also providing power consumption information and allowing security audits by integrating gem5 [BBB<sup>+</sup>11], OMNeT++ [OMN] and McPAT [mcp] tools under a single interface. The integration allows the "simulation of an actual system of systems, including its complete software stack, network dynamics and energy aspects" [BTN<sup>+</sup>18]. As such, it is regarded as an emulator. The framework takes care of the synchronization between the different simulators under the hood. In [BTN<sup>+</sup>18], the tool's performance, and correctness are assessed using two different case studies;
3. **CupCarbon** [BCC<sup>+</sup>18] is a tool to "design and simulate Wireless Sensor Networks dedicated to Smart-city and IoT applications" [BCC<sup>+</sup>18]. It is comprised of a radio channel, which integrates two radio propagation models, an interference module, randomly generating communication interference in networks and a 2D/3D environment, useful for designing

smart territories, e.g., smart cities. The framework supports the visualization of the simulation results while modeling radio propagation and interference;

4. **CupCarbon-Lab** [BML<sup>+</sup>18] is an extension of the CupCarbon simulator and uses less powerful devices, such as the Raspberry Pi or Android smartphones, to act as a network of physical nodes, providing hardware-in-the-loop simulation. It uses the host computer as a server where the system can be monitored and visualized. This host functions as the remote control of the simulation environment and is also responsible for injecting code into the physical devices.
5. The solution proposed by **D'Angelo, Ferretti et al.** [FDGM17, DFG17a, DFG17b, DFG18] makes use of different established simulators to simulate a whole system with varying levels of detail. It uses an agent-based simulator (Smart Shire Simulator [DFG17b]) for the high-level, low-detail simulation and a transportation system simulator based on ADVISOR [adv] and a network simulator (OMNeT++ [OMN]) when higher levels of granularity are needed. Despite using three distinct simulation tools, the framework has only two levels of simulation, as the transportation system and network simulators simulate mutually exclusive devices, i.e., a device is either simulated by one or the other;
6. **DPWSim** [HLC<sup>+</sup>15] is a simulator using the standard Devices Profile for Web Services (DPWS) [dpw09]. DPWSim is bundled with (1) DPWS Explorer, a tool for analyzing and exploring DPWS service and (2) DPWSim Web, a web interface for controlling DPWSim devices on different devices, e.g., mobile phone. DPWSim follows a discrete event architecture where devices have *events* that they fire or *operations* that can be executed. The only way to customize device functionality is either through its GUI or a `.dpws` file;
7. The agent-based domain-agnostic solution presented by **Jung et al.** [JSW18] applies a different architecture allowing runtime changes and integration of distinct simulation tools simultaneously. The simulator requires a communication and synchronization interface between the varying simulation frameworks so that they can communicate and behave as expected. In [JSW18] a communication interface was built for Simulink [sim] and OpenModelica [ope] to permit agents simulated in both tools to communicate with each other. Due to this integration, the tool allows for hardware-in-the-loop testing, given that the underlying simulators in use have support for it. The publication has no mention about whether or not the tool can be automated;
8. **VisibleSim** [DPB13] is a discrete event simulator aimed with a 3D environment that can display objects (exported at `.obj` files) or simple blocks. It can simulate forces applied by objects on others using a physics engine. According to [DPB13], one of its main advantages is the ability to simulate 2 million nodes at a rate of 650,000 events per second on a Xeon-based processor (base-frequency of 2.56GHz) with 12GB of RAM;

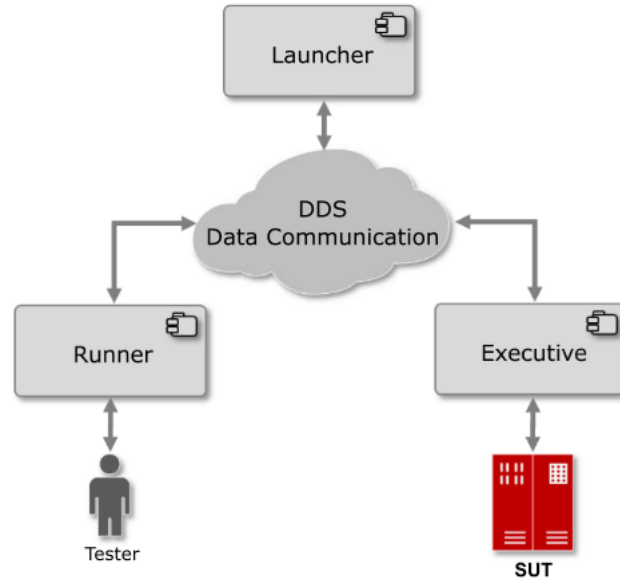


Figure 3.2: KhronoSim high-level architecture. [CMB<sup>+</sup>18]

9. **IoTTest**, proposed by Bures [Bur17], is an approach to device simulation that focuses on integration and end-to-end (e2e) testing by borrowing technical concepts from JUnit [JUn]. It is capable of emulating some IoT devices while integrating real physical hardware into the system, effectively providing hard-in-the-loop simulation;
10. **KhronoSim** [CMB<sup>+</sup>18] is a distributed data-oriented simulator for testing the IoT. It provides "progressive development" [CMB<sup>+</sup>18] by allowing the user to convert simulated nodes into physical devices progressively. It's divided into three main components: (1) Launcher, acting as a lock on the system under test; (2) Runners, that are responsible for executing a test suite and must wait on Launcher's permission to run; and (3) Executive, which, in the end, executes the commands (such as sending messages and reading electrical signal) requested by a Runner. KhronoSim [CMB<sup>+</sup>18] can be used in conjunction with Simulink [sim], effectively giving it spatial and temporal context awareness;
11. **MAMMotH** [LODYJ13] is a massive-scale Internet-of-Things emulator, with the aim of emulating 20 million devices. It uses a distributed architecture based on virtual machines (VMs), which can emulate 10,000 nodes per VM and uses ns-2 [ns2] models for emulating network protocols;
12. The simulator proposed by Fortino et al. [FGRS17] makes use of a multi-agent system to model nodes in a network. Using the INET Framework [ine] for OMNeT++ [OMN], the simulator can model WSNs, while the integration of ACOSO [FGR12] provides the platform to develop and run the multi-agent system;

13. The tool presented by **Zouai et al.** [ZKHS17] is an agent-based simulation tool for smart houses, capable of controlling smart devices from an external network by using an IoT Gateway that connects the house to the Internet;
14. The **SWE Simulator** proposed by **Gimenez et al.** [GMPE13] uses a Sensor Web Enabled [swe] (SWE) architecture component, Sensor Observation Service (SOS), that serves as a sensor information database, providing standard data insertion and retrieval methods, via HTTP, for all sensors. The tool also features a Control Center, where actions are handled as responses to events and where a web platform, used for monitoring the system, is served from;
15. **WoTSIM** [MCFL14] is a Web of Things (WOT) simulator allowing static, moving, and offline devices. Things are described as XML configuration documents at the start of the simulation, and more XML documents are created during runtime and stored in "Real-Time IR Test Collection" [MCFL14]. Running a simulation with 6000 sensors for 19 hours resulted in a 41.1 MB collection size [MCFL14].

We divided the testing platforms into the following categories, according to their characteristics:

**Type** The type of tool: simulator or emulator. Possible values: *Simulator* or *Emulator*;

**Scope** Some tools are built for a specific use case (e.g., smart cities, smart grids, etc.). Therefore, knowledge of the scope of each platform is useful to assess whether or not they may solve a problem one may have. Example values: *smart city*, *smart grid*, *network*, *device*;

**Architecture** Simulators for the Internet-of-Things tend to gravitate towards Multi-Agent Systems or Discrete Event architectures, with each one having its benefits and disadvantages;

**License** The license of a software product may impede its utilization or be a strong point in favor of its adoption. For example, an open source simulator allows for high degrees of customizability by giving users the power to change and alter the software. Possible values: the name of the license or *N/A* if the software license is not known;

**Scalability** Defines how the platform scales. Due to the different performance metrics used by the authors, this characteristic is only an approximation. Possible values are *very low*, *low*, *medium*, *high* or *very high*; or *N/A* if there is no information regarding the scalability of the tool;

**Tier** IoT systems usually follow the three-tier architecture, as explained in Section 2 (p. 5). Aggregating the simulation of all tiers under a single simulator is generally out of scope for most tools, so the project usually selects one tier to focus on, which is what this parameter describes. Its values can be the permutations of *Cloud*, *Fog* and *Edge*;

**Context Awareness** The context of a simulator represents whether or not the simulated devices know about their time and/or spatial position. Possible non-mutually exclusive values: *Spatial*, *Temporal*;

**Hardware-in-the-loop** A simulator is said to be a hardware-in-the-loop simulator if it can support physical devices [Bac05]. Such a tool has the advantage of testing a system closer to the real environment, which provides more confidence in its correctness. Possible values: *yes*, if the software supports physical and virtual devices simultaneously; *no* otherwise;

**Can be Automated?** Automated testing reduces the time spent by developers testing software and, therefore, platforms with this capability are advantageous for their users. Possible values: *yes*, *no* or *N/A* if there is no information available;

**Programming Language** Programming language(s) that are required for the usage of the simulator.

Table 3.3: Simulators and their properties. A hyphen (-) signals that the information is not available. A small circle (●) means "yes"; the lack of any element means "no".

Solution	Type	Scope	Architecture	License	Scalability	Tier	Context Awareness	Hardware-in-the-loop	Can be Automated?	Programming Languages
Li et al. [LWZ <sup>+</sup> 13]	Simulator	Device	Event-based	-	-	Edge	Spatial, Temporal	-	-	C++ <sup>a</sup>
COSSIM [BTN <sup>+</sup> 18]	Emulator	-	Event-based	BSD 2-Clause	-	Edge	-	-	-	Any
CupCarbon [BCC <sup>+</sup> 18]	Emulator	Smart City	Event-based	No	-	Fog	Spatial	-	-	SenScript [sen] or Python
CupCarbon-Lab [BML <sup>+</sup> 18]	Emulator	Smart City	Event-based	-	-	Fog	Spatial, Temporal	●	-	SenScript [sen] or Python
D'Angelo, Ferretti et al. [FDGM17]	Emulator	Smart City	Agent-based	-	High	Fog	Spatial, Temporal	-	-	-
DPWSim [HLC <sup>+</sup> 15]	Simulator	-	Event-based	No	Low	Edge	Spatial	-	-	None <sup>b</sup>
Jung et al. [JSW18]	Simulator	Device	Agent-based	-	-	Edge	Spatial, Temporal	●	-	Any <sup>c</sup>
VisibleSim [DPB13]	Simulator	Virtual World	Event-based	Apache 2.0	Very High	Edge	Spatial, Temporal	-	-	C++
IoTTest [Bur17]	Emulator	-	-	-	-	Edge	-	●	●	-
KhronoSim [CMB <sup>+</sup> 18]	Simulator	-	-	-	-	Edge	Spatial, Temporal	●	●	Any
MAMMoH [LODY13]	Emulator	-	-	-	Very High	Edge, Fog	-	-	-	-
Fortino et al. [FGRS17]	Emulator	-	Agent-based	-	High	Fog	-	●	-	-
Zouai et al. [ZKHS17]	Simulator	Smart House	Agent-based	-	-	Fog	Spatial, Temporal	-	-	-
SWE Simulator [GMPE13]	Emulator	-	Event-based	-	-	Edge, Fog	Spatial, Temporal	●	-	Any <sup>d</sup>
WoTSIM [MCFL14]	Simulator	-	Event-based	- <sup>e</sup>	High	Edge	Spatial, Temporal	-	-	-

<sup>a</sup>Although C++ does not have first class support, the paper states that "C++ code generated in different environments can communicate through interface under Rhapsody/OXF support, thus implement integrated co-simulation of physical and computation processes." [LWZ<sup>+</sup>13]

<sup>b</sup>Sensors are programmed using the DPWSim GUI [HLC<sup>+</sup>15].

<sup>c</sup>Supports the programming languages used by the underlying simulation systems.

<sup>d</sup>Since data is inserted and retrieved using HTTP virtually any language can be used

<sup>e</sup>Reference [MCFL14] states the simulator would be open-sourced, but its code could not be found at the time of this writing.

### 3.1.3 Expanded Search

The systematic review protocol results are detailed in Section 3.1.2 (p. 14). However, when comparing these tools to non-systematic surveys [JJW17, CBBZ18], these display some additional information that was not found by the search terms established. Multiple reasons may apply for this divergence in results, such as:

1. the tools not having an associated academic publication (e.g. AWS IoT Device Simulator [aws], IoTIFY [iot], obtained from Google by searching *IoT Simulator*), causing them not to be returned by the publication databases outlined in Section 3.1.1 (p. 11);
2. the publications may not use the keywords *IoT* or *Simulator* — including variants —, but rather employ *Wireless Sensor Networks* or *Emulation*, e.g., CupCarbon [MLBK14], resulting in them not being included in the search results.

#### 3.1.3.1 Expanded Results

To complete this survey, the unique results from the found reviews [JJW17, CBBZ18] were selected. From this selection, the tools were assessed against the selection process defined in Section 3.1.1.6 (p. 14) and reviewed according to the parameters established in Section 3.1.2 (p. 14). Using the aforementioned methodology, the following tools remain:

1. The Java-based urban IoT simulator presented by **Brambilla et al.** [BPC<sup>+</sup>14] unites the COOJA [ÖDE<sup>+</sup>06] and ns-3 [ns3] network simulators using DEUS [AAZ09], a discrete event simulator, to aggregate information, such as transmission delays, energy consumption and schedule communication events [BPC<sup>+</sup>14]. The OSMobility [osm] simulator is included to allow the simulation of vehicles and pedestrian within the urban environment;
2. **Karnouskos et al.** [KT08] proposes a DPWS-based [dpw09] multi-agent enterprise-grade simulator with the goal of providing transparent simulation, mobility support, and dynamic environments capabilities, while also supporting micro and macro simulation and *self-X* (e.g., self-configuration, self-healing, etc.) behavior [KT08]. The tool uses agents that represent individual simulated devices as well as physical devices, while also providing a communication bus so that the transmission of information can be made regardless of the type of device, i.e., if the device is simulated or not [KT08];
3. **SenseSim** [DNP15] is a simulator for the Internet-of-Things based on agents and discrete events that, using its layered architecture, intends to be capable of running the same application regardless of whether the devices executing it are simulated or not. To achieve such a goal, each simulated sensor is composed of four different components handling distinct parts of the logic: from changing the sensor's internal state or providing an API to manipulate to the sensor to a middleware that runs an application on the sensor [DNP15];



## State of the Art

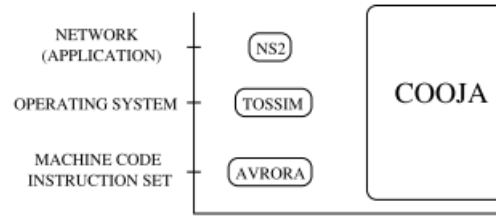


Figure 3.3: COOJA can run at multiple simulation levels. [ÖDE<sup>+</sup>06]

4. **iFogSim** [GDGB16] is an extension of CloudSim [CRB<sup>+</sup>11], and so inherits its functionalities. However, unlike CloudSim, which is only focused on the cloud, iFogSim adds a novel *edge-ward placement* [GDGB16] strategy. This strategy favors the use of edge and fog devices over the cloud. Its GUI allows the creation of a network with edge, fog and cloud devices and its connections. A special class represents not only a communication packet but also defines the processing requirements and the length of the data it encapsulates. This characteristic is what allows the edge-ward placement strategy to decide whether or not to route the tuple to a device closer to the edge;
5. **COOJA** [ÖDE<sup>+</sup>06] is a sensor network simulator for the Contiki [DGV04] operating system. It emulates devices running the code without needing modifications so that transferring to a physical device can be made painlessly. COOJA also emulates radio models, allowing different types of radio wave propagation. COOJA makes use of specific features of the Contiki kernel (e.g., its event-driven nature) to emulate devices, which makes it incompatible with other operating systems;
6. **OMNeT++** [OMN] is a discrete event simulator for communication networks and distributed systems containing not only a GUI but also a domain-specific C++-like programming language named *NED* for defining the network topology. The behavior of models is written in C++ in a separate file from the topology. While the simulator is not tailor-made for the Internet-of-Things, the fact that it can model the topology of a network and is event-based makes it suitable for this use case;

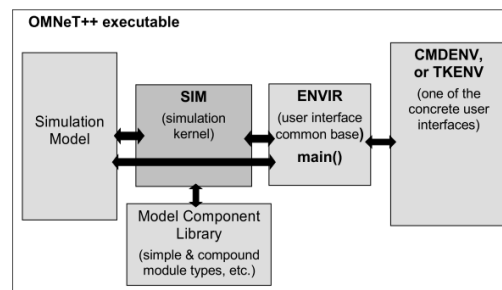


Figure 3.4: OMNeT++ Internal Architecture. [VH08]

7. **ns-3** [HLR<sup>+</sup>08] is a network simulator with the aim of succeeding the widely used ns-2 [ns2], but with added realism and ability to plug real devices into the simulation. ns-3 also provides C++ and Python API to plug into, while also supporting a distributed simulation setting for large scale simulations [RH10]. Although being lower-level when compared to other simulators, when performance and network emulation is needed, ns-3 seems to be a promising solution;
8. **QualNet** [qua] is a commercial network emulator with spatial awareness that allows the building of large wired and wireless networks and visualization by plugging into other tools. It is also possible to connect QualNet to real networks so that real hardware can be tested.

### 3.1.4 Analysis & Discussion

It is essential to take note that more tools were not included because they did not meet the criteria defined in Section 3.1.1 (p. 11). Specifically, tools that focus solely on simulating the cloud part of an IoT system were left out, while platforms that simulate the fog and the edge were included.

#### 3.1.4.1 Result Analysis

Having summarized the results from not only the systematic review but also from the surveys found, there are a total of 23 simulators for the Internet-of-Things available. It is possible to verify that they vary significantly in terms of characteristics, from emulator to event-based, from highly scalable to supporting hardware-in-the-loop. The tools found were collected and analyzed. The results from such analysis are laid out below:

**Type** Regarding whether the tools are simulated or emulate systems, ten simulators were found contrasting with the 13 emulators discovered;

**Scope** From the 23 platforms identified, the most common is scope is smart cities, followed closely by network simulators. The others either have little representation or have no particular scope and are deemed as generic purpose;

**Architecture** When it comes to architecture, event-based tools are the most common ones, while the agent-based simulators, as well as the ones without mention of their architecture, appear only six times. There is also a case where the framework follows both an event- and an agent-based architecture;

**License** Most of the tools found do not mention a license, the few that do are split between an open source one (e.g., GNU GPLv2, BSD 2-Clause, etc.) and no license. There is a clear gap in the open source simulation tools with permissive licenses;

**Scalability** The majority of solutions identified, do not have their scalability metric evaluated from performance benchmarks. The few that include such measurement are usually the

Table 3.4: Simulators and their properties. A hyphen (-) signals that the information is not available. A small circle (●) means "yes"; the lack of any element means "no".

Solution	Type	Scope	Architecture	License	Scalability	Tier	Context Awareness	Hardware-in-the-loop	Can be Automated?	Programming Languages
Brambilla et al. [BPC <sup>+</sup> 14]	Emulator	Smart City	Event-based	-	High	Edge, Fog	Spatial, Temporal	-	-	Java
Karnouskos et al. [KT08]	Simulator	-	Agent-based	-	High	Edge, Fog	Spatial	●	-	Any
SenseSim [DNP15]	Simulator	-	Agent-based & Event-based	-	-	Edge	Spatial, Temporal	●	-	-
iFogSim [GDGB16]	Simulator	-	Event-based	No	-	Cloud, Fog	-	-	-	Java
COOJA [ÖDE <sup>+</sup> 06]	Emulator	-	-	Contiki <sup>a</sup>	Medium	Edge	Spatial	-	-	C
OMNeT++ [VH08]	Emulator	Network	Event-based	Academic <sup>b</sup>	-	Edge	-	●	●	C++/NED
NS-3 [HLR <sup>+</sup> 08]	Emulator	Network	-	GNU GPLv2	Very High	Edge	-	●	-	C++/Python
QualNet [qua]	Emulator	Network	-	-	High	Edge	Spatial, Temporal	●	-	-

<sup>a</sup><http://www.contiki-os.org/license.html>

<sup>b</sup>Academic Public License: <https://omnetpp.org/intro/license>

ones high or very high and present it as an advantage of the tool. This may explain why there are many platforms with high scalability, while there are only a few that do not scale very well;

**Tier** It is possible to verify that most focus on the Edge tier of an IoT system. A substantial number of tools simulate only the fog tier, and there are a few that can support both tiers. Only one solution can simulate the cloud: the reason for that is that publications with tools that *only* have the ability to simulate the cloud tier have been excluded, as explained in Section 3.1.4 (p. 22);

**Context Awareness** Regarding context awareness, i.e., whether device are aware of time and their position, most tools support spatial awareness, with some also providing temporal awareness. However, there are also a few platforms that do not explicitly mention to which extent they provide simulated devices with context awareness;

**Hardware-in-the-loop** Out of the 23 solutions analyzed, 11 provide some way of connecting the simulation to real hardware, while 12 do not mention whether or not it is supported;

**Can be Automated?** The automation of simulation tools can prove useful for integrating autonomous processes such as continuous integration/delivery pipelines or batch operations. However, only IoTTest [Bur17], KhronoSim [CMB<sup>+</sup>18] and OMNeT++ [VH08] are capable of such automated procedure;

**Programming Language** Tools regularly introduce restrictions such as the programming language to code the devices with so that higher performance gains can be achieved. While performance is regarded as good, the restriction can also be too limiting and prohibit users from choosing a certain platform. From the programming languages that the discovered simulators support, C++ repeatedly emerges, as well as Java. There are also some frameworks that are programming language agnostic and some others that require programs to be written in a language created just for the simulator.

### 3.1.4.2 Evolution of Publications

To understand how the field of simulation for the Internet-of-Things, it is important to perceive how it is developing. One metric to be used is the number of publications of the field over the years. Starting in 2006, the year the first IoT simulation tool present in the survey was published, and until 2019, publications were aggregated and their frequency measured.

### 3.1.4.3 Survey Research Questions

The research questions posed in Section 3.1.1 (p. 11) serve as guidance for the development of this publication, and its answers are crucial to assess the state of the art of the field of IoT simulators. Such answers are provided below.

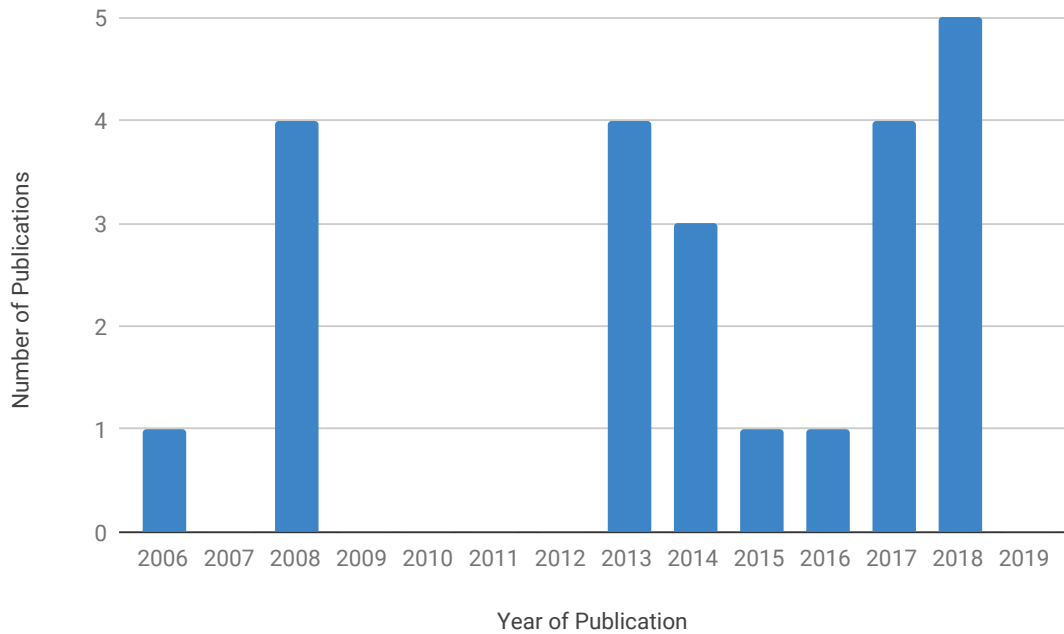


Figure 3.5: Publications per year of IoT simulators.

**SRQ1 What relevant IoT simulation platforms exist?** As seen in Section 3.1.2 (p. 14) and Section 3.1.3 (p. 20), there are multiple IoT simulation platforms with different architectures, standards, and goals. There are two most common approaches for simulation of IoT systems that keep appearing in diverse solutions, namely: discrete event simulations (DES) and agent-based simulations (MAS). It can also be verified that most solutions are not open source or available for free download. When it comes to programming languages, there are different options, although most publications do not mention their requirements;

**SRQ2 Of these, which ones support testing and validation with hardware-in-the-loop?** Table 3.3 (p. 19) gives an overview of simulators and their properties. One can see that from the 15 solutions presented; only 6 provide hardware-in-the-loop simulation. The coexistence of virtual and physical devices offers better confidence about the system under test, as it is closer to the real environment the system will run in. Taking a look at Table 3.4 (p. 23), it is possible to verify that most tools support hardware-in-the-loop, totaling 11 out of 23;

**SRQ3 What is the scope of the tools found in SRQ1?** Different tools focus on distinct use cases and objectives, ultimately leading to the developers limiting the scope of said tools in different ways. Some tools prefer to focus on emulating networks, including their quirks and peculiarities, while others pick the simulation of smart cities as their goal. This variety in scope influences the user selecting a simulation platform, justifying the importance of having this distinction among the available simulators. From the 23 tools found from

the systematic and expanded searches, four focus on smart cities, three on network simulation, two on modeling devices, one on smart worlds and another one in smart houses. The remaining have a generic purpose, and so their scope has not been restricted;

**SRQ4 Of those found in SRQ1, which ones support the automation of tests?** Automated tests are becoming more prominent in the software industry, but the IoT field still seems like it is lagging on this matter. Out of 15 solutions found in the systematic search, only IoTTest [Bur17] and KhronoSim [CMB<sup>+</sup>18] support automated testing. Moreover, the expanded search only found OMNeT++ [VH08] to support such feature, out of 8 total tools.

### 3.1.5 Conclusions

In this work, we survey 1400 papers from IEEEXplore, Scopus and Compendex that result in 15 simulators for the Internet-of-Things. Furthermore, an expanded search is conducted using the reviews found among the publications retrieved from the databases, totaling 23 IoT simulators. The results show that automated simulation of the Internet-of-Things with hardware-in-the-loop support is still limited, with only IoTTest [Bur17], KhronoSim [CMB<sup>+</sup>18] and OMNeT++ [VH08] possessing these capabilities. Although these are good news for the field of simulation of the IoT, of these simulators, the first two are neither open source nor free to use, while the third is a network simulator and has no particular focus on the Internet-of-Things. As such, there is a clear lack of an open source hardware-in-the-loop IoT simulator capable of being automated, with OMNeT++ being the only possible solution, even though it is not ideal.

Summarizing, we noticed there is no broad range of solutions for testing IoT systems beyond the simulation-based and some testbed-based approaches. However, when analyzing the landscape of testing solutions for software-only systems, we verify that there is a lack of similar tools for IoT based systems. Especially when taking into account that IoT systems have unique characteristics and limit the use of software-only testing tools out of the box [Bei18]. Future work can focus on developing an IoT simulator that (1) is open source, (2) can be automated and (3) supports hardware-in-the-loop.

## 3.2 Related Work

Section 3.1 (p. 11) provides an in-depth overview of the current state of the art regarding the simulation of the Internet of Things. It is visible that there are multiple tools with an approach to simulate physical and virtual devices, each with its own advantages and disadvantages. Through the lens of the review, the two following tools seem the most complete of the survey.

**IoTTest** [Bur17] is based on JUnit [JUn] and supports unit, integration and system tests. The framework is divided into three major components:

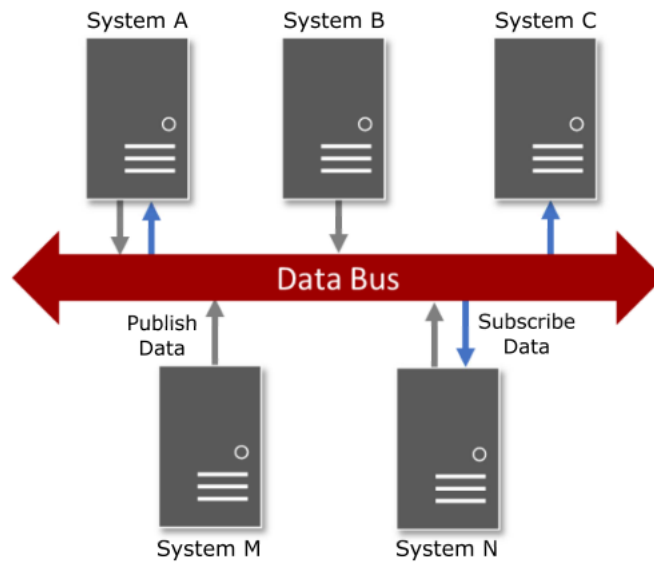
- *Test Orchestration Component* which extends the JUnit framework and controls the execution of the tests, for example, being able to cancel them if an exception is thrown;

- *Devices Simulation Platform* is the control center for all the simulated devices and contains the main data bus, through which all the information flows;
- *Devices Library* is a Java library that encapsulates emulators of various physical devices that act as providers to the main data bus.

The presence of a shared data bus abstracts the nature of the devices and allows for the introduction of physical nodes without others perceiving it.

**KhronoSim** [CMB<sup>+</sup>18] is a distributed IoT simulator, able to run on multiple computers and to emulate the physical environment. It is built on a data-oriented architecture (DOA), which consists of a main data bus and multiple systems connected to it. This provides loose coupling between the systems as they don't need to be aware of the existence of the other devices. Figure 3.6 (p. 28) explains visually the overview of a DOA. KhronoSim is divided into four main parts:

- *DDS Data Communication* that encapsulates the communication buses between the three following components;
- *Launcher* which is a daemon that controls access to the System Under Test by creating a mean of communication between the SUT and the Runner. In case there is more than one Runner trying to access the same resource, the Launcher will place it in a FIFO queue until the resource is free;
- *Runner* is the actual runner of the tests. Once it is allowed by the Launcher to execute the tests, it will ask the Executive to execute actions on the SUT on his behalf. Actions include sending messages and reading or writing electrical signals;
- *Executive* is the executor of the commands requested by Runners that controls the devices, either simulated or not. The Executive can be replicated on multiple computers to allow the concurrent execution of tests by increasing the number of simulated devices.

Figure 3.6: Data oriented architecture.[CMB<sup>+</sup>18]

### 3.3 Visual Programming Languages

A *Visual Programming Language* aims to ease the burden of developing programs by utilizing pictorial components like blocks and arrows over the traditional approach of using text [CRV15]. However, designing good visual programming languages is not trivial and the field of VPL design faces multiple obstacles that include the (1) representation of language tokens, (2) effective use of screen real estate and (3) documentation [BBB<sup>+</sup>95]. The representation of language tokens (or *static representation*) is how the language is structured. These are usually easy to create when a VPL's programs are built imperatively or in a declarative manner; however, when a VPL relies on a program-by-demonstration method (e.g., Adobe Photoshop Actions), creating the static representation becomes much more difficult. Having such representation permits a better review and analysis process for development. The use of screen real state in an effective fashion is also a problem that needs to be solved. VPLs tend to grow horizontally, while textual programming languages grow mostly vertically. In theory, this can be seen as a benefit, since most screens are wider than they are high; in practice, however, this ultimately depends on how compact VPLs can be. Textual languages are very space efficient and can usually fit multiple instructions in one line and tens of lines vertically. As such, VPLs must solve the problem of developing a space-efficient syntax before becoming more relevant in software engineering. Another point that VPLs must address is documentation, as there is a clear need for it. Although, there is no standard solution as of yet. While textual languages use textual comments to document code, visual languages have a broader range of possibilities to explore. One solution used is to employ textual documentation in the program, while some other tools [LCI<sup>+</sup>88] allow comments only on mouse hover, in order to save screen space.



### 3.3.1 VPLs for the Internet of Things

Visual Programming Languages are usually represented as nodes with connections between themselves and with multiple inputs or outputs. This modeling paradigm is closely related to an IoT system, where devices are spread across an area and send messages between each other. Due to this parallelism between both architectures, VPLs can be viewed as an accurate and practical tool to model IoT systems. After a quick search various VPLs with focus on IoT were discovered:

**NETLabTK** [net] NTK is a visual IoT system designer supporting Arduino and Raspberry Pi.

A web interface is provided to create, edit and delete devices, which are represented as nodes which are, in turn, connected to other nodes in order to communicate. NTK's GitHub repository has not been updated since September 2017;

**Ardublock** [ard] Ardublock is a VPL for Arduino-based platforms that builds on the Arduino IDE. However, development seems to have halted at the time of writing, as the last commit dates to November 2017;

**Node-RED** [nod] Initially developed by IBM and now a project of the JS Foundation, Node-RED is the most popular VPL platform and marketed as "flow-based programming for the Internet of Things" [nod]. Node-RED architecture is focused on nodes and message passing, which constitute different flows. Its ability to combine software with hardware and its extensibility create a huge array of possibilities for building IoT systems. It also provides community-made nodes through its library that fosters code reuse and reduces development time;

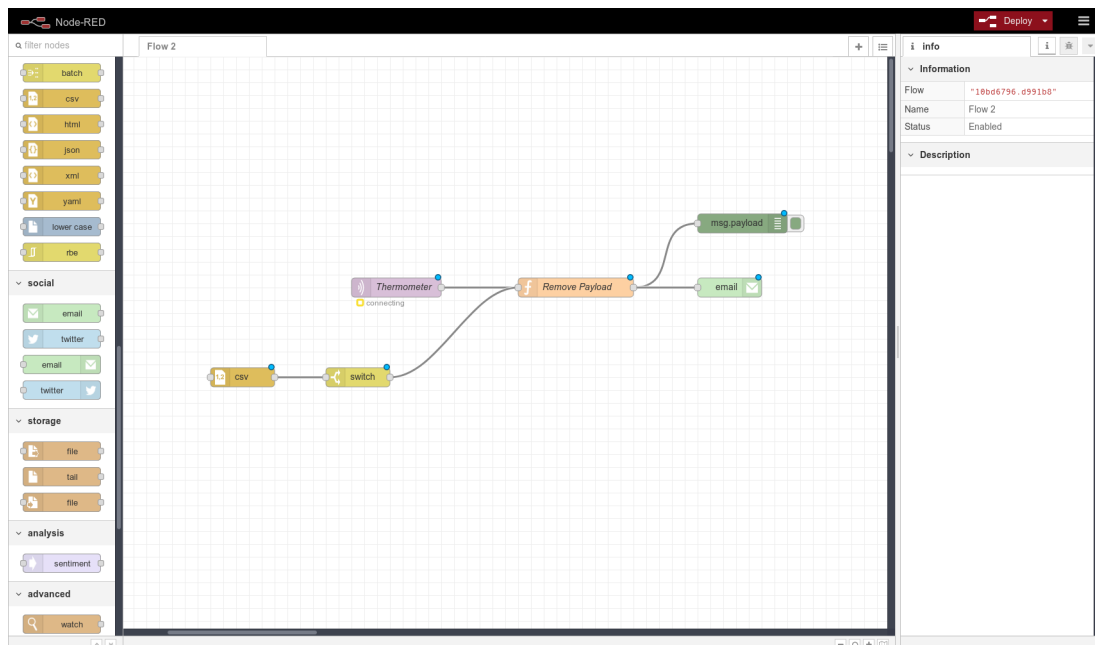


Figure 3.7: Node-RED UI.

**S4A [s4a]** S4A is a modification of the Scratch [scr] visual programming language to work with the Arduino platform. According to the website, it should work on all Arduino boards but was only tested on Arduino Diecimila, Duemilanove and Uno;

**Noodl [nool]** Noodl is a web platform for building an IoT using visual constructs. It is extendable by connecting to hardware using the MQTT protocol. Its last release is dated March 2018;

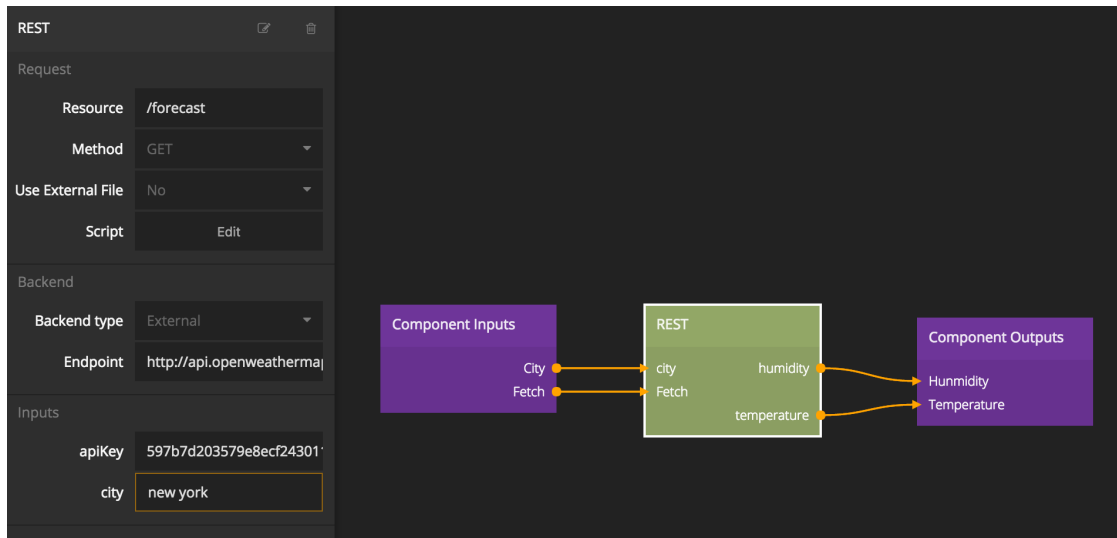


Figure 3.8: *Noodl* web interface. [nool]

**AT&T Flow [atn]** AT&T Flow is a tool built by AT&T to develop IoT systems in a flow-based structure. Nodes represent data fetching or data mutation operations such as HTTP requests, TCP connections and functions;

**Reactive Blocks [rea]** Reactive Blocks is an Eclipse-based visual programming tool suitable for any Java IoT stack. It provides an extensive array of blocks with different abilities, e.g., sending SMS or emails, making HTTP requests or obtaining GPS location;

**GraspIO [gra]** GraspIO is a visual mobile IDE for small scale IoT systems. It only works for the Raspberry Pi and can be connected to IFTTT [ift] to automate processes outside the scope of GraspIO;

**Wylodrin STUDIO [wyl]** Wylodrin STUDIO is an IDE for programming IoT systems that integrates a visual programming language, shell access and textual programming languages (e.g., JavaScript, Python and bash). It works by installing in the device the Wylodrin server, which provides remote control capabilities;

**Zenodys [zen]** Zenodys is an IoT focused platform that permits the construction of IoT system using a visual language. It can connect to multiple data storages such as Microsoft Excel, SQLServer and MySQL and provides step-by-step debugging capabilities.

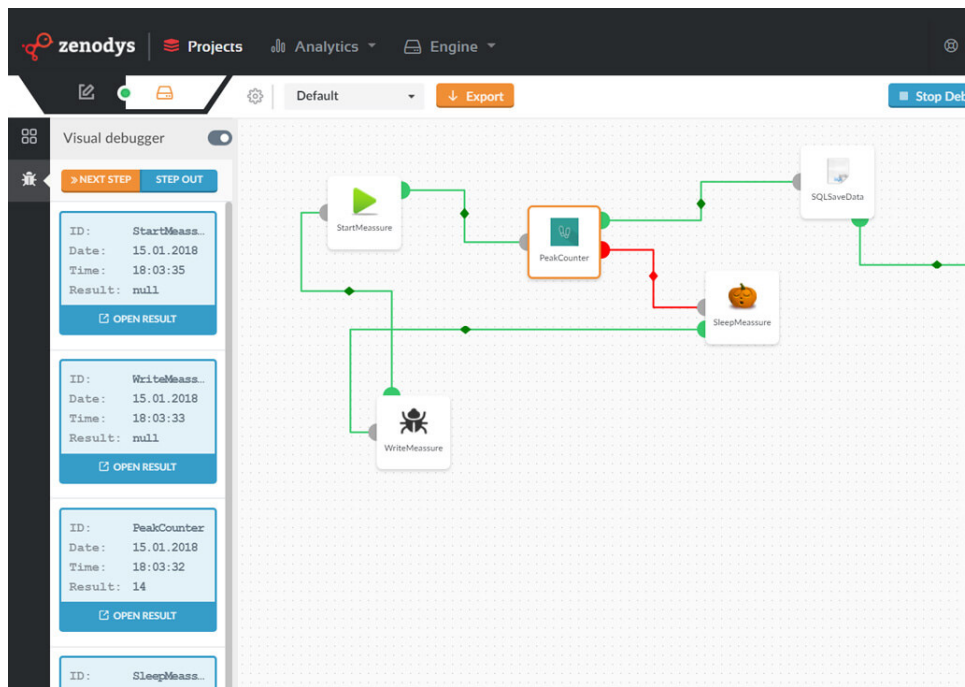


Figure 3.9: Zenodys web interface. [zen]

From the above tools, some seem to stand out. Commercial solutions such as AT&T Flow [atn] and Zenodys [zen] seem the most complete and generic visual programming tools for the Internet of Things. They even provide additional features such as git-based versioning (AT&T Flow) or step-by-step debugging (Zenodys). However, Node-RED, due to the fact that it is open source, provides more possibilities when it comes to extensibility. The other tools appear to be too restrictive since they are only compatible with specific platforms (e.g., Arduino, Raspberry Pi), thus constraining the user's ability to extend and customize their behavior. Furthermore, from all the tools found, none refer to testing automation for IoT systems, which is a common practice in software-only systems.

### 3.4 Summary

Section 3.1 presents a systematic literature review of the field of IoT simulation and automated testing, summarizing the properties and capabilities of the discovered tools. Section 3.2 (p. 26) follows from the previous one by identifying the most complete solutions and providing a brief analysis of each one. Ultimately, Section 3.3 (p. 28) introduces visual programming languages and dives deeper onto VPLs for the Internet of Things and identifies a research gap in automated testing for the IoT using VPLs. Effectively, the researched tools seem lacking as they do not support visual testing, relying instead on either textual tests or no tests at all; are not capable to running tests automatically, requiring a human to run the tests manually; and do not allow continuous regression testing.

## State of the Art

## Chapter 4

# Problem Statement

This chapter provides a precise definition of the problem, as seen in Section 4.1. Afterward, Section 4.2 (p. 34) presents a list of desired features for the solution to be developed. Subsequently, Section 4.3 (p. 34) defines the scope of the project. The use cases for the solution are detailed in Section 4.4 (p. 35). Section 4.5 (p. 36) specifies the research questions to be answered by this work. A validation methodology for the found answers is delineated in Section 4.6 (p. 37). Finally, Section 4.7 (p. 37) summarizes the chapter by presenting a succinct overview of the topics mentioned throughout.

### 4.1 Current Issues

As previously verified in Chapter 3 (p. 11), there are multiple solutions that provide the ability to create IoT systems using graphical interfaces. However, none of these are capable of running automated tests. As such, the problem is defined as the union of the following issues:

1. **Visual testing:** the lack of a visual testing tool creates a paradigm difference between the programming part of the development cycle, which is done in a visual interface, and the testing part, which is either not done or developed using textual programming;
2. **Automated testing capabilities:** the lack of a tool that supports automated testing capabilities results in the failure to provide the benefits they are experienced in other software engineering fields where automated testing is available, such as faster feedback cycles and reduced manual testing time;
3. **Recurring regression testing:** the absence of a tool to allow the continuous verification of the system in order to detect inconsistencies, such as a failing physical device, is worrisome as that would lead to a more reliable system.

This defines the problems this project aims at solving by fulfilling the criteria described in Section 4.2 (p. 34).

## 4.2 Desiderata

A *desiderata*, Latin for "things desired", serves as an aggregation of needs or wants. In this case, it combines a number of requirements that a solution needs to have in order to solve all the problems identified in Section 4.1 (p. 33), which are the following:

- D1: Define tests visually** so that they are consistent with the actual program under test, providing the user with the benefits from visual programming languages;
- D2: Testing the whole system** so that the integration of different components can be validated and errors can be identified, eventually leading to greater confidence on the correctness of the system;
- D3: Testing isolated parts of a system** so that smaller units of logic can be verified before running more extensive tests. By having cheaper and shorter tests run first, test pipeline costs and duration are reduced, as extensive tests are not executed if a cheaper and shorter one has failed;
- D4: Run tests on demand** so that the behavior of the system can be tested either periodically or remotely, permitting the constant monitoring of the constituents of the system, which is more important in IoT systems, since they can suffer from both software and hardware problems;
- D5: Inject faults** to verify that the system works as expected under real hardware or software failure, hopefully reducing the number of problems in production, thus providing end users with a better user experience;
- D6: Generate report** in order to obtain a high-level view of the problems affecting a system with the intent of allowing a faster troubleshooting process.

## 4.3 Scope

In order to be able to develop the project in a realistic time frame, its scope must be correctly defined and some features may be left as secondary goals. One of these is security, which is of utmost importance, but will be relegated to a secondary objective of this project, thus allowing the dissertation to focus more on its primary ambitions.

Moreover, one of the goals of this project is to build a simple to use yet powerful platform. However, that always depends on the final user, since they may have varying degrees of familiarity and easiness with the concepts and knowledge required to handle the framework. With this in mind, the aim of the tool is to be used by (1) system integrators, such as users that are accustomed to deal with the IoT environment and provide consultancy on the matter; (2) devices developers which can use the tool to more easily and visually carry out tests on the components being developed; and (3) tech-savvy consumers that buy devices and desire to build and test the system by themselves.

## 4.4 Use Cases

This section provides all the use cases that are to be fulfilled by the solution. In Chapter 3 (p. 11) it is visible that an appropriate tool for visual testing in visual programming languages has not been found. Below, the most important use cases are detailed in order to achieve a usable and useful version of the tool, while satisfying the requirements established in Section 4.2 (p. 34):

**UC1: Fake Values from Devices** Given a scenario where a room has a thermostat, window and heater, when the window is not open and the thermostat is reading a temperature below 18°C, the heater should be turned on. It should be possible to fake the thermostat's readings. Achieving this will help test for various type of errors, e.g. when the temperature is 19°C and the window is closed, the heater should remain turned off; when the temperature is 17°C and the window is closed, the heater should turn on, etc.;

**UC2: Obtain State of Devices** Given a scenario where a room has a thermostat, window and heater, when the window is not open and the thermostat is reading a temperature below 18°C, the heater should be turned on. It should be possible to obtain the state of the heater so that tests can verify whether or not the heater has been turned on;

**UC3: Verify the Behavior of a Single Component** Given a scenario where a room has a thermostat and a window, when the thermostat is reading a temperature below 18°C, the window should be closed. It should be possible to write a test to detect a possible logic or physical errors by injecting fake values into a component and watching its response, e.g. fake the thermostat's readings to read the temperature as 17°C, then verify if the window is closed. This construct can automate the finding of logic errors such as 'temperature < 18' being confused with 'temperature > 18';

**UC4: Validate the Interaction between Components** Given a scenario where a room has a thermostat, window and heater, when the window is not open and the thermostat is reading a temperature below 18°C, the heater should be turned on. It should be possible to write a test to detect integration problems by injecting fake values into some components and spying the response from others, e.g. fake both the thermostat to read the temperature as 17°C and the window to be closed, then watch the heater to verify if it has turned on. This construct can automate the finding of integration problems such as information not being sent, physical problems with components, unexpected communication protocols, etc.;

**UC5: Generate Test Report** On a complex system, refactors are scary as they can touch many parts of the codebase and possibly create problems. As such, when a refactor is done, it is useful to run all tests and aggregate their results by generating a summary with all failing tests including the location, so that bugs that were introduced during said refactor can be found and eliminated;

**UC6: Behavior Exploration** When prototyping, it may be useful to mock and spy on certain devices to explore their behavior and get a better understanding of its functionality. Given a

scenario where some window blinds are open, it may be desirable to close them by faking a ‘close’ command, so that its speed and behavior can be assessed.

## 4.5 Research Questions

Given the current state of the Internet of Things, more specifically, the testing of IoT systems and the scope defined in Section 4.3 (p. 34), the main research question of this work is:

### **RQ: How to support automated testing in a VPL using visual elements?**

However, this question is too broad and generic to lead to a single answer. Hence, a subdivision was made, resulting in the following research questions:

**RQ1: How to test in production?** Concurrent use of resources is known to cause problems (e.g., race conditions, data inconsistency), as such it is important to understand how to test systems without altering production data;

**RQ2: How to show test results using visual metaphors?** Test results express the correctness of the system at a given point in time and its representation in a visual programming language is imperative for the end user to quickly grasp such information;

**RQ3: How to represent multiple test cases using graphical elements?** A test suite is usually comprised of multiple test cases and representing them in a compact space and in a user-friendly way may increase developer productivity while reducing strain on the user;

**RQ4: How to represent asynchronous test cases in a VPL?** The IoT is built on asynchronous events and messages, creating a scenario where most communication is done in a non-synchronous way. In such a scenario representing asynchronous tests becomes a requirement;

**RQ5: How to automate tests that depend on physical devices?** Testing edge scenarios is important to explore behaviors that push systems to an extreme. However, in some of these cases, it is not feasible to wait for certain conditions (e.g., wait for a fire alarm to fire). As such, it is essential that a way to automate tests that rely on these edge conditions is developed;

**RQ6: How to leverage tests to allow continuous regression testing?** An IoT system joins the problems of software-only systems with all the hardware issues that can happen, leading to a very large set of potential errors. In such a scenario, a way to test for regressions (e.g., check if a lamp stopped working) can be important for the long-term monitoring of a system.



## 4.6 Validation

In order to validate whether or not the solution implemented fulfills the *desiderata* and use cases defined, two test scenarios will be developed and each one will be verified against each *desideratum* and research question. This validation will show to what extent the research questions were satisfactorily answered.

## 4.7 Summary

Section 4.1 (p. 33) raises the issues of the current state of the testing of the Internet of Things. Subsequently, Section 4.2 (p. 34) presents a *desiderata* for a visual programming tool supporting automated testing, while Section 4.3 (p. 34) details the assumptions that have been made to restrict the scope of the project so that it can be developed in a reasonable time frame. Afterward, several use cases are also defined in Section 4.4 (p. 35). The research questions this dissertation aims to answer are introduced in Section 4.5 (p. 36), which will be validated as explained in Section 4.6.

## Problem Statement

# Chapter 5

## Solution

This chapter explains how the solutions to the problems presented in Chapter 4 (p. 33) are designed, structured and achieved, while also detailing the reason why some design decisions were taken.

First, the standalone IoT simulator is described, along with its architecture, benefits, disadvantages and trade-offs. Afterward, the implementation of an extension for Node-RED based on the simulator is explained. Subsequently, the architecture and behavior of a testing framework for Node-RED are described. Finally, a brief summary of the information revealed in the chapter is presented.

### 5.1 Simulator for the Internet of Things

The first goal of this dissertation was to build a generic simulator for the Internet of Things that allowed hardware in the loop and was programming language agnostic. The idea behind such simulator is to allow the introduction of software-based logic instead of actual hardware devices to mimic the functionality of a system. As a secondary goal, the tool should permit the programming of devices using almost any language available. This section describes the reasons behind its architectural decisions, characterizes its components and summarizes the knowledge obtained from its implementation.

#### 5.1.1 Architectural Overview

In order to solve the problems outlined in Chapter 4 (p. 33), an attempt was made to build an architecture that would be as generic as possible, meaning that anyone should be able to apply it to almost every IoT system.

The system is built on top of **message queues** using the MQTT [[mqta](#)] protocol and its constituent devices are assumed to follow the **Web Thing API**<sup>1</sup>. However, the architecture is protocol

---

<sup>1</sup><https://iot.mozilla.org/wot/>

agnostic, so it could have been AMQP<sup>2</sup> or any other, but MQTT was chosen due to being "extremely lightweight" [mqta], thus making it good to use in an IoT scenario.

#### 5.1.1.1 One-Queue Architecture

Initially, a simple architecture using a *Message Proxy*, as seen in Figure 5.1, was developed. It relies on a *Message Queue*, such as RabbitMQ<sup>3</sup>, to act as a message broker between devices.

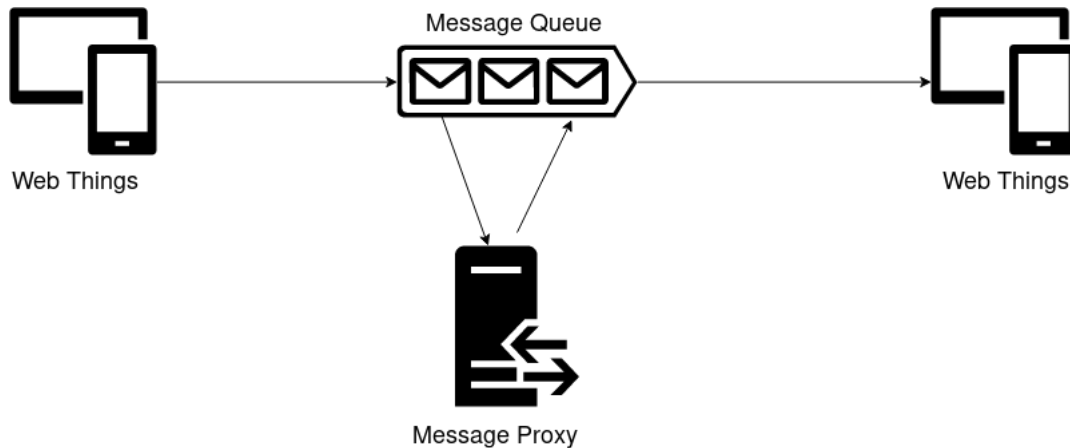


Figure 5.1: One Queue Architecture.

This approach works very well in most cases and is the simplest to understand. However, in specific scenarios, some problems may surface, namely:

1. **Ensure Proxy Priority:** when there is a message, the proxy should be able to read it before the end devices are able to. This means there must be some kind of higher priority for the proxy (or lower priority for the devices). MQTT does not have the concept of priority, so to ensure the correct behavior of the system, we would need to depend on the different brokers that implement some kind of priority system or that allow custom routing logic;
2. **Message Looping:** the particular solution has the problem of possibly creating infinite loops of messages. Let's say we have a rule:

*Whenever a message from /things/thermometer is received with temperature > 25 , then set the temperature to 30.*

Using the proposed architecture, the system would enter an infinite loop of receiving and sending a message with *temperature = 30*. This state is undesirable as it causes unnecessary strain on the system and corrupts its logic from the point of view of the user.

In order to solve the first problem, the solution would need to restrict itself to working only on brokers with a priority system. This is undesirable as restricting the message queues this tool works with basically diminishes the possible impact it can have.

<sup>2</sup><https://www.amqp.org/>

<sup>3</sup><https://www.rabbitmq.com>

Dealing with the second issue requires a more intricate solution, such as **storing ID of sent messages**. By allowing the message proxy to store the ID of sent messages, it can assess whether or not messages have already been sent. In the case that they had been sent, the proxy would put them back in the queue. If they had not been sent, they would be processed according to the predefined configuration. Since packet IDs only work on MQTT packets with QoS > 0 [mqtb], it is unclear how QoS = 0 would be handled in this case.

This approach carries the advantage of requiring only one message queue. However, besides the lack of knowledge of how to handle of QoS = 0 messages, it also brings the disadvantage of needing to store the ID of messages sent, which raises even more questions, such as (1) how it would impact memory consumption and (2) how much IDs to store before deleting.

Due to these restrictions and disadvantages, a new two-queue architecture was devised and implemented, as seen in Section 5.1.1.2.

#### 5.1.1.2 Two-Queue Architecture

A *Two-Queue Architecture* (cf. Figure 5.2) splits a message queue into two: one for reading; another one for writing. The *Write Message Queue* is the message queue where the **devices write to**, while the *Read Message Queue* is where **devices read from**. The process is reversed for the *Message Proxy*.

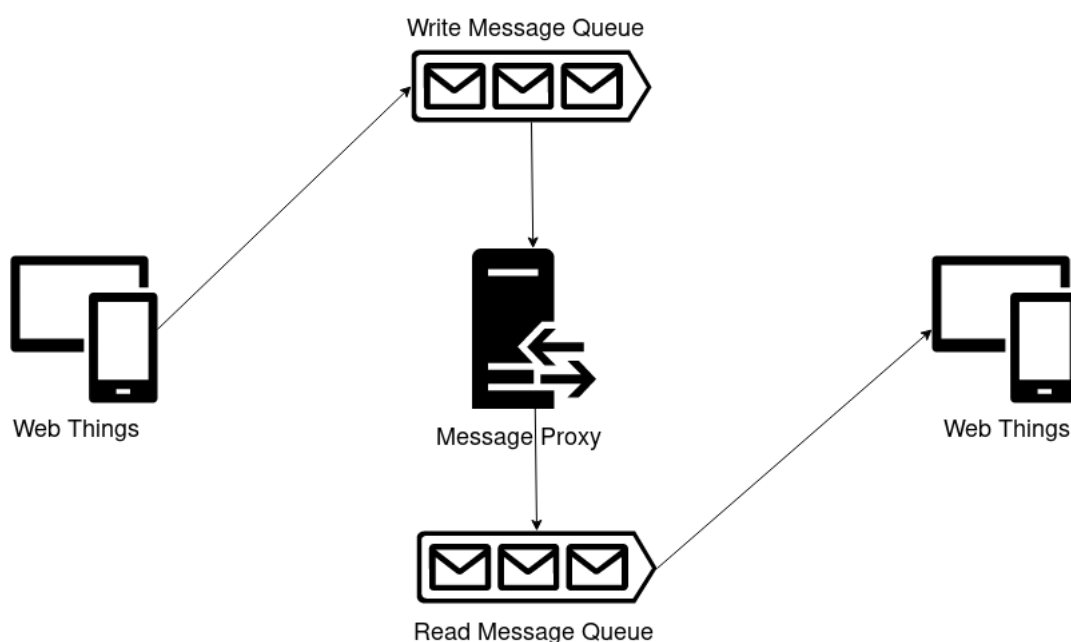


Figure 5.2: Two Queue Architecture.

The workflow of this system is as follows:

1. Devices send messages to the *Write Message Queue*;
2. The *Message Proxy* reads from the *Write Message Queue* and applies its configuration, e.g. replace the message, suppress it, etc.;

3. The *Message Proxy* sends non-suppressed messages to the *Read Message Queue* after altering the message according to the configuration defined by the user;
4. Devices read from the *Read Message Queue* without knowing if the received messages were modified or not.

This specific architecture solves the problems outlined in Section 5.1.1.1 (p. 40):

1. **Proxy Priority** is ensured by only allowing the connection of *Message Proxy* between the message queues, which solves the problem effectively. A disadvantage that may surface from this approach is that the proxy must be able to cope with the throughput of the **whole** system, not only the messages it is interested in. This may create a bottleneck. Fortunately, having multiple instances of the *Message Proxy* can help, as long as they all have the same configuration at every point in time;
2. **Message Looping** is effectively resolved when moving messages from the write to the read queue, which prevents the message from being processed again by the proxy, as it cannot read from the output queue.

Although previously raised questions were answered assertively, some new problems may surface, such as (1) the *RETAIN*<sup>4</sup> property of the MQTT protocol, which is rendered useless, since the *Message Proxy* cannot know whether the message sent to the *Write Message Queue* was sent with *RETAIN* set to 1 or not; (2) how to synchronize the configuration across multiple proxies to enable the system to scale with the number of messages.

## 5.1.2 Component Characterization

The possible architectures were specified in Section 5.1.1 (p. 39) and their pros and cons outlined. For this specific implementation, the *Two-Queue Architecture* was chosen since the disadvantages that it carries are regarded as less severe, when compared to the *One-Queue Architecture*'s.

### 5.1.2.1 Message Queues

The *Message Queues* act as a messenger between end devices and the *Proxy*, as seen in Figure 5.2 (p. 41). One of them receives messages from devices and only the *Proxy* should read from it. This is called the *Write Message Queue*. The other one — the *Read Message Queue* — relays messages published by the *Proxy* to the end devices.

Although the tool was tested using RabbitMQ with the MQTT Plugin, in theory, any message queue supporting MQTT should work, since there is nothing being used that is exclusive to RabbitMQ.

---

<sup>4</sup>[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718038](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718038)

### 5.1.2.2 Proxy Configuration

The *Proxy Configuration* defines the runtime behavior of the *Message Proxy* (cf. Section 5.1.2.3, p. 44). Its purpose is to define how to handle the messages the proxy receives. From delaying or generating a message to suppressing or modifying it, the configuration effectively has the capability of defining or altering the behavior of the system while it is running. This proves useful when experimenting with parts of a system where switching between simulated and real devices is beneficial.

The configuration is split between two types of agents:

**Replacers** are in charge of replacing the input message with any number of output messages.

An output message can be a static value or a dynamic value calculated using an expression and the input value. Every output message can also be delayed by an arbitrary amount of milliseconds, making it useful for simulating timeouts or delays in the network. The input message can also be suppressed, useful when the intention is to replace the message entirely, or can be let through;

**Generators** have the responsibility of generating messages according to a *cron* expression. Generators can also have multiple outputs, which can be used to delay and calculate values depending on the timestamp the generation function was triggered.

The configuration is internally defined as a JavaScript object and so every format that is able to be converted into a JavaScript object is supported. For this work, we used the TOML <sup>5</sup> format.

### Example Configurations

In order to provide concrete examples with the goal of helping understand how configurations are defined, two examples are presented. The first one represents a replacer, while the second one configures a generator. In this specific instance, they are separate to simplify the explanation, but they can be composed together to form complex configurations.

The configuration below defines a replacer:

```

1  [[replacers]]
2  input = {
3      href = "/things/thermometer",
4      property = "temperature",
5      suppress = true
6  }
7
8  [[replacers.outputs]]
9  expr = "value > 1 ? 1 : value"
10 href = "/things/thermometer"
11 property = "temperature"
```

<sup>5</sup><https://github.com/toml-lang/toml>

## Solution

```
12 delay = 2
```

Listing 5.1: Example configuration using a replacer.

The configuration above defines a replacer with one input and one output. The input is the Web Thing defined at `/things/thermometer` and the property `temperature` will be used as the base value. Furthermore, the message will be suppressed and, thus, not sent to the receiver. The output is the same Web Thing (`/things/thermometer`) and the same property (`temperature`), but the message will be delayed by 2 seconds and the new value will be calculated based on the expression `value > 1 ? 1 : value`. In this case, `value` is the value of the property of the input message, so the actual behavior of the expression is to clamp the input value to a maximum value of 1.

The following configuration represents a generator:

```
1 [[generators]]
2 input = { cron = "10/5 * * * * *" }
3
4 [[generators.outputs]]
5 value = 45
6 href = "/things/humidity-sensor"
7 property = "humidity"
```

Listing 5.2: Example configuration using a generator.

This configuration defines a generator that runs according to the `10/5 * * * * *` cron expression, which in this case means every 5 seconds starting at the 10th second of every minute. The output is sent to `/things/humidity-sensor` with the property `humidity` set as 45.

### 5.1.2.3 Message Proxy

The *Message Proxy* is the most crucial component of the system as it implements the most important functionality. It is responsible for proxying messages from the *Write Message Queue* to the *Read Message Queue*, applying the rules defined in the configuration. Said configuration is injected at runtime and is defined in finer detail in Section 5.1.2.2 (p. 43). Whenever a new configuration is injected, the list of message handlers is updated with the newly generated handler functions.

A replacer handler function implements the design pattern *Chain of Responsibility* [Gam95], in the sense that a message is passed from handler to handler, with each being responsible for executing its task independently from the others.

The handler function is passed, as parameters: (1) an object, which we will call *message context*, with information about the message received, including the MQTT topic, message payload, and whether or not to suppress the input message; and (2) a publish function so that the handler can publish messages to the *Read Message Queue*. For replacers, it follows the algorithm below:



## Solution

1. If the message topic does not match the replacer configuration's input *href*, then skip the handler by returning the message context;
2. Otherwise, run the replacer handler, comprised of the following steps:
  - (a) Obtain the new value, which is done either by setting a static value or computing the expression given;
  - (b) Create the message with the new *href* and *property* fields (if not set, use the input's);
  - (c) Publish the message with the specified delay.

When it comes to generators, the algorithm is as follows:

1. Use the static value or compute it from the tick;
2. Create the message with the *href* and *property* fields;
3. Publish the message with the specified delay.

## 5.2 Node-RED IoT Simulation

Node-RED is a web user interface for flow-based programming for IoT. It allows the creation and manipulation of nodes (which can be IoT devices, transformation functions, etc.) as well as wiring them together to develop complex systems. Groups of interconnected nodes constitute *flows*, which represent higher units of abstraction when compared to nodes and can encapsulate arbitrarily complex logic.

In order to provide a simple user interface for the IoT simulator described in Section 5.1 (p. 39) and reduce the development time and effort, we chose Node-RED as the GUI to replace the TOML configuration file, hoping to relieve the burden of defining the configuration by hand and giving better feedback to the user by means of visual elements.

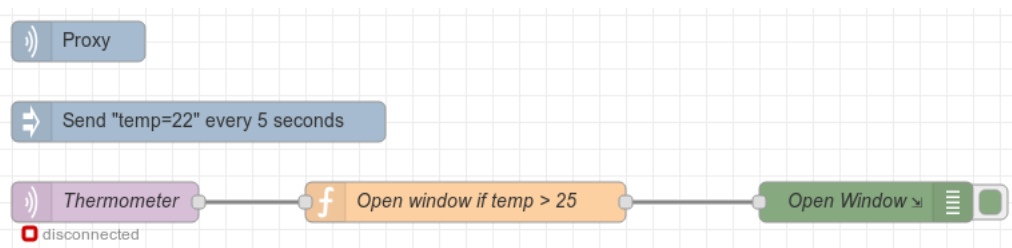


Figure 5.3: Node-RED IoT Simulation Extension.

Figure 5.3 captures a simple simulation scenario where the temperature of a thermometer is sent every 5 seconds.

### 5.2.1 Component Characterization

With the goal of integrating the IoT simulator with Node-RED, an extension was developed that contains several custom nodes. Its goal is to entirely replace the TOML configuration file with a graphical environment that achieves the same goals, but with less strain and with more visual appeal to the end user. The newly created nodes are the following:

**Proxy Config** *Config* node that selects the message queues to read from/write to. Uses Node-RED's MQTT node to set both the *Read Queue* and *Write Queue*;

**Proxy** Node that depends on *Proxy Config* and is shared among all *Generators* and *Replacers*. Represents one instance of a *Message Proxy*;

**Generator Input** *Config* node that configures the generation function to execute on a periodic basis. It uses *cron*<sup>6</sup> based scheduling, with accuracy to the second;

**Generator Output** *Config* node that configures the value generation and validates the settings for a generator. Sets the Web Thing's `href` and `property` as well as the message delay and how to calculate its values — whether it's static or calculated from an expression;

**Generator** Node that generates values on a periodic basis. Depends on *Generator Input* and *Generator Output* to set the configuration;

**Replacer Input** *Config* node that configures how to replace input values. It receives the Web Thing's `href` and `property` to replace and a flag indicating whether or not the original message should be suppressed;

**Replacer Output** *Config* node that configures the value generation and validates the configuration for a replacer. Sets the Web Thing's `href` and `property` as well as the message delay and how to calculate its values — whether it's static or calculated from an expression;

**Replacer** Node responsible for replacing incoming messages' values with new ones or calculating new values based on old ones. It is also capable of suppressing receiving messages and can also output either static values or the result of an expression. Depends on *Replacer Input* and *Replacer Output* to function.

Both *Replacer* and *Generator* generate either static or dynamic values. The first are set once and do not ever change, e.g. 42, "on". The second, however, rely on a simple expression parser — *mathjs* [mat] — that is capable of executing simple JavaScript mathematical expressions (e.g., `value * 2`, `value > 10 ? 5 : 10`) and can compute result based on the input value, such as the value in the input message.

---

<sup>6</sup><https://linux.die.net/man/5/crontab>

### 5.3 Node-RED Testing Framework

Having implemented the simulator and added the Node-RED integration, new opportunities arose. One of them is the possibility of testing Node-RED flows using Node-RED itself, something which is not available at the time of this writing. Even though there is a proposal for *Flow Testing*<sup>7</sup>, it is still awaiting approval and the workflow differs greatly from the one implemented in this project. One of the major differences between the proposal and this implementation is the fact that the first focuses on having tests in the **same** flow as the nodes under test, while the latter utilizes a separate flow to harbor the tests. Figure 5.4 (p. 48) illustrates a test suite running.

With this extension, it is expected that tests in Node-RED flows will transform the platform for the better by permitting more critical systems to developed and tested. Thus, unlocking the faster development time and visual benefits that Node-RED already offers, while providing a well-tested and reliable IoT system for the end user.

#### 5.3.1 Solution Overview

The solution is based on the fact that Node-RED allows the access of nodes in different flows. This is how nodes like *Mock* or *Spy* can access the properties of nodes under test. Node-RED also has a capability named *Context*<sup>8</sup> which allows to set variables in different contexts, such as global, flow or node. By leveraging the flow context, the various testing nodes are able to communicate with a central node to inform it of tests' failures or successes. Taking advantage of these features comes with some tradeoffs that are detailed in Section 5.3.3 (p. 50).

#### 5.3.2 Custom Nodes

The framework is implemented by providing the user with custom nodes, giving the user all the flexibility from the composability of Node-RED components. In order to allow the user to benefit from said composability, nodes were created with the goal of being simple and modular, as commanded by Unix philosophy. As such, the following nodes were developed:

**Test Runner** is responsible for acting as a switch and selecting which test can run at once. It is activated by receiving a message with the payload `"start_tests"`. Once this message has been received, it sends a message to reset the tests' state followed by another one to effectively run the tests. By requiring a message to be received, the test runner can be composed with other nodes, which allows tests to be run every 15 seconds - using an *Inject* node - or be triggered remotely by an MQTT message;

**Test Start** works mostly as an informative node by informing the user whether the test has already been started or is still pending authorization from the *Test Runner*. Its name should explain what the test is validating so that tests are easily recognizable;

<sup>7</sup><https://github.com/node-red/designs/tree/c6c51a910020a55fdbf60d685f86a254cfc5f9f6/designs/flow-testing>

<sup>8</sup><https://nodered.org/docs/user-guide/context>

Solution

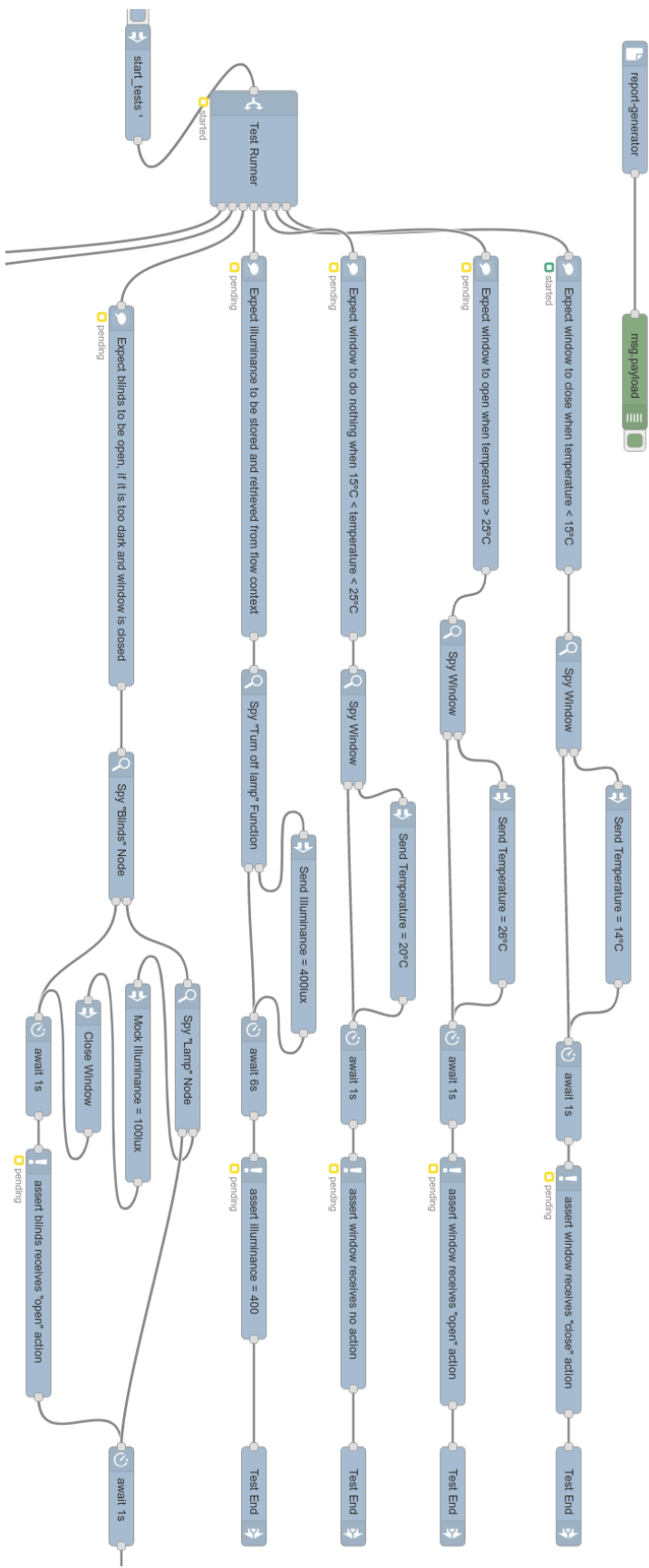


Figure 5.4: Node-RED Testing Framework running tests.

## Solution

**Spy** is a node that focuses on spying what other nodes are receiving. It is very useful to use with an *Assert* node to verify if the flow under test is behaving as expected. Under the hood, the *Spy* node installs an *input* event listener in the target node and sends every message the target node receives;

**Mock** fakes the sending of a message by a node. This is done by calling the *send* method of the target node with the specified message. Its configuration is very similar to Node-RED's *Inject* node but does not allow for time-based repetition. The configuration interface can be seen in Figure 5.5;

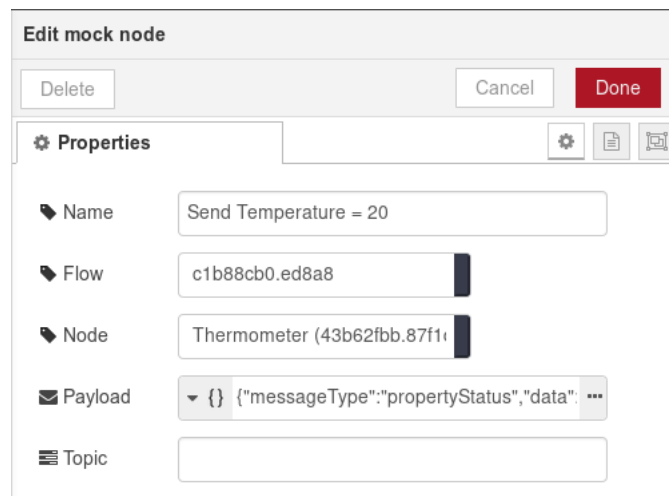


Figure 5.5: *Mock* node configuration interface.

**Await** is a node that waits for a message to arrive, forwarding it if it does arrive. Otherwise, the test continues without the message. It is useful for asynchronous operations such as spying and then mocking, but can be used for other use cases as well. The timeout can be specified in milliseconds and is editable through Node-RED's web interface;

**Assert** validates whether or not a received message is in accord with some expectation. If the assertion is not successful, then not only will the whole test fail, but the *Test Runner* will be informed and the failure added to the collection of test failures, which can afterward be gathered using a *Report Generator*.

The *Assert* node reuses the *Function* node functionality of accepting and running JavaScript code. The `expect` function from the JavaScript testing library Chai [cha] is injected into the scope so it is possible to write asserts in a behavior driven style.

If the `expect` fails, then the whole test is deemed as a failure and the `testDone` function, present in the flow's context, is called with the assertion error, the ID of the assertion node and the message and function body that caused the failure;

**Test End** signals that a test case has ended successfully. When a `run` message arrives at a *Test End*, the context flow `testDone` function is called to inform the *Test Runner* that the test is finished and the next one can now begin;

**Report Generator** aggregates the failures of the current test run and passes it as a message to the next node(s). It contains the function body and the message that triggered each failure. It is up to the user to format the output as desired.

### 5.3.3 Limitations

As described in Section 5.3.1 (p. 47), some trade-offs were made to accommodate the testing framework. These are defined below:

**Test tab must be after flow tab** Since the test tab depends on the flow under test and Node-RED loads the nodes in the order of the appearance of their tabs, the flow under test tab must precede the test tab, so that when the latter runs, the first is already loaded and operations are executed correctly;

**One Test Runner per flow** Due to the fact that the *Test Runner* leverages the flow context to communicate with other testing nodes, only one instance can be present in a flow at once. Failing to do so may create inconsistencies as multiple *Test Runners* may alter the data at the same time;

**One Report Generator per flow** Since the *Report Generator* also uses the flow context to be informed of failures, only one can be present in a flow. Failure to comply will cause the same problems as having multiple *Test Runners* in a flow;

**No separate environments** The extension executes the tests by injecting and spying messages from nodes in the flow under test. What this means in practical terms is that the tests are running in the production system, i.e., it is running the actual flow. While this is acceptable in some cases as it provides more confidence on the correctness of the system, in others it may create unintended consequences, such as altering production data;

**Tests cannot be run in parallel** The underlying behavior of running tests is to inject the messages directly into the flow under tests, meaning that there is no separate instance or sandbox for testing. The lack of a proper container to isolate the flow makes the tests not safe to be run in parallel as multiple executions can interfere with one another;

**One Test Start per Test Runner output** The outputs of the *Test Runner* node represent one test, so hooking up multiple tests to one output will make them run concurrently, which can cause concurrency issues such as race conditions and data inconsistencies.

## 5.4 Summary

The solutions presented throughout this chapter solve the problems identified and described in Chapter 4 (p. 33). Section 5.1 (p. 39) explains the developed prototype of a simulator for the Internet of Things providing hardware-in-the-loop support as well as being programming language agnostic. Afterward, Section 5.2 (p. 45) illustrates the integration of the simulator into Node-RED and characterizes its constituent parts. Section 5.3 (p. 47) reports the behavior of the testing framework created for Node-RED.

Since no solution is perfect, it is possible to find future work to be done in order to further improve the field of IoT simulation and testing. Section 5.3.3 (p. 50) thoroughly details the caveats and limitations of the framework, thus defining possible paths of progress in this area.

Solution



## Chapter 6

# Evaluation

This chapter focuses on the evaluation of the developed solutions. Section 6.1 establishes the scenarios used to verify the success of the implementation and to which extent it realizes the desired features described in Section 4.2 (p. 34) and fulfills the use cases outlined in Section 4.4 (p. 35). Subsequently, Section 6.2 (p. 55) evaluates how the solutions fared when put against the test scenario. Section 6.3 (p. 56) answers the research questions defined in Section 4.5 (p. 36). Ultimately, Section 6.4 (p. 57) closes the chapter by summarizing the conclusions obtained from previous sections.

### 6.1 Test Scenarios

Two tests scenarios were designed to assess how the tool fulfills the requirements and answers the research questions. These scenarios are virtual and do not map to physical devices by leveraging testing framework data injection capabilities.

#### 6.1.1 Scenario A

Figure 6.1 illustrates how the Scenario A is represented in Node-RED. It mimics a simple system where a window will open if the temperature in a room becomes too high.

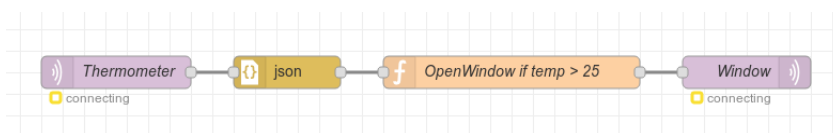


Figure 6.1: Test Scenario A.

The test suite for this scenario is portrayed in Figure 6.2 (p. 54). It contains three test cases, complemented by a node that generates a report of test failures.

## Evaluation

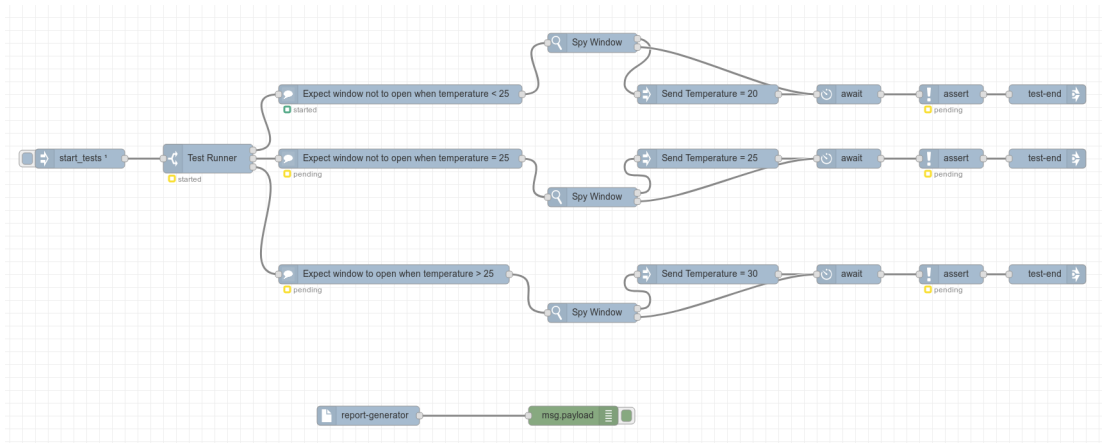


Figure 6.2: Test suite for Scenario A.

The test flow incorporates an *Inject* node that periodically starts the *Test Runner*, three test cases and a *Report Generator*. The test cases assert that the window does not open when the temperature is less than or equal to 25°C and that it receives an `open` command whenever the temperature exceeds 25°C. Furthermore, a report is generated in case of any failure.

### 6.1.2 Scenario B

Scenario B (cf. Figure 6.3) represents an automatic room light adjustment system. The goal is to keep the room light in a range of values, taking into account energy expenditure (e.g., prioritizing opening blinds over turning on a lamp) and room temperature.

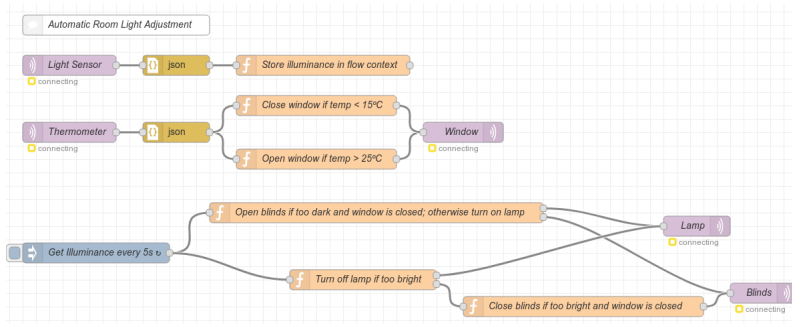


Figure 6.3: Test Scenario B.

The scenario is comprised of two sensors and three actuators:

**Light Sensor** Measures the room light;

**Thermometer** Obtains the room temperature;

**Window** Can be opened or closed remotely and influences the room temperature and brightness;

**Lamp** Can be turned on or off and affects the room temperature, bright and energy expenditure;

**Blinds** Can be opened or closed, impacting the room light.

Figure 6.4 illustrates a subset of a test suite for this scenario.

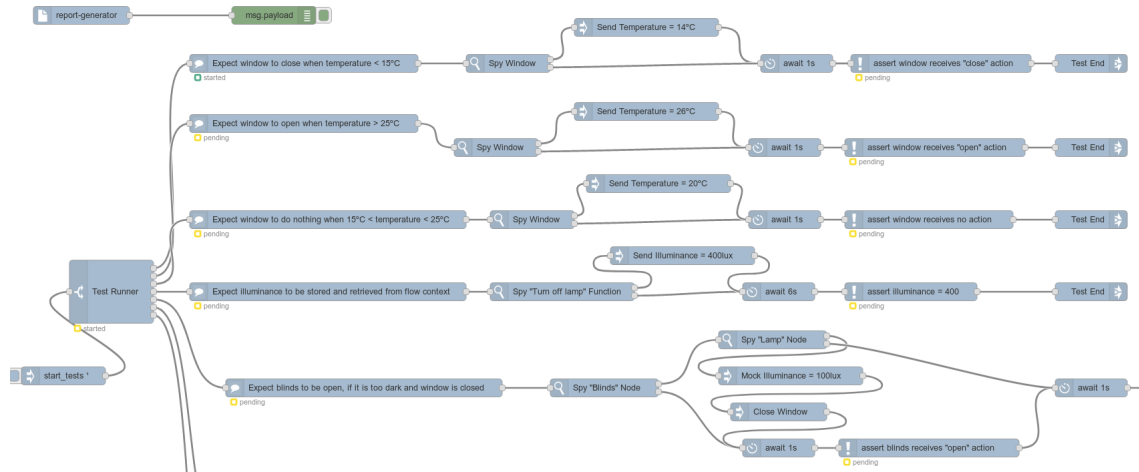


Figure 6.4: Subset of the test suite for Scenario B.

It consists of five visible test cases, as denoted by the nodes depicted by a comment balloon. These nodes represent the start of a test and only one will run at once. They are connected to a *Spy* node, identified by the magnifying glass icon, that are used to obtain the messages sent by nodes in the flow under test. Using the top test case as an example and starting at the *Spy Window* node, a message setting the temperature to be 14°C is sent by the *Send Temperature = 14°C* node. Afterward, a *Await* node waits one second for a message to be sent by the *Spy*. The result is then forwarded to the node represented by the exclamation point, whose function is to verify that certain conditions are met. If they are not, the test is considered a failure. If the assertions pass, the flow continues executing until a message meets the *Test End* node, signaling the end of the test and allowing the *Test Runner* to run the following test.

After all tests have been executed, the *report-generator* will send a payload with the test failures to the next node(s) in the flow, which can be tailored to the specific use case, e.g., send an e-mail to the person responsible for the system.

## 6.2 Assessment

The evaluation of the developed tools is executed through the fulfillment of all the requirements established in Chapter 4 (p. 33). Utilizing the scenarios defined in Section 6.1 (p. 53), the following desideratum were achieved:

**D1: Define tests visually** Both scenarios illustrate that the tests were defined in a visually appealing and effective way;

**D2: Testing the whole system** Scenario B demonstrates the validation of behavior of a complex system;

**D3: Testing isolated parts of a system** Scenario B exhibits the testing of a subset of the whole system;

**D4: Run tests on demand** Both scenarios show the possibility of running tests on demand, e.g., by using the *Inject* node to start the *Test Runner* periodically;

**D5: Inject faults** Both scenarios prove that it is possible to insert faults into the system, e.g., by injecting invalid messages;

**D6: Generate report** Both scenarios contain a node that generates a report based on the failed tests.

Concluding, it is visible that the solution meets every point of the *desiderata* without sacrificing the visual aspect of Node-RED.

### 6.3 Research Questions

This dissertation obtained the answers to the research questions established in Section 4.5 (p. 36) through research and the implementation of the solutions described in Chapter 5 (p. 39).

**RQ: How to support automated testing in a VPL using visual elements?** In order to achieve automated testing in a VPL by maintaining the visual components, the introduction of more graphical elements with specific behavior, such as components for asserting conditions, mocking elements or listening to messages, is necessary.

While the main research question provides an overview of the procedure to implement automated testing in a VPL, the following questions concretely lay out more specific scenarios.

**RQ1: How to test in production?** Having a separate environment for testing and production is a common practice in the software world to avoid the mix between test data and real users' information. However, for the Internet of Things, such separation is complex, as copies of physical devices cannot be made. Thus, testing in production should restrict itself to some guidelines to avoid affecting the real data;

**RQ2: How to show test results using visual metaphors?** Extrapolating from the solution found in the specific implementation explained in Section 5.3 (p. 47), test results can be shown by attaching stateful informational graphical elements to nodes. This way nodes themselves are unaffected, but there still is visual feedback on the results of tests;

**RQ3: How to represent multiple test cases using graphical elements?** It is normal in the software world to have test suites that encapsulate either test suites or test cases. In the VPL field, such structure can be mirrored by having a graphical element (test suite) that branches into multiple other elements that are laid out alongside each other;

**RQ4: How to represent asynchronous test cases in a VPL?** IoT is mainly asynchronous by nature, as most of its architectures are based in events and message passing. Hence, representing asynchronous tests in an effective manner is crucial to the good working of the framework. From our experience, asynchronous test cases can be represented using some type of await graphical element that waits for a value, message, etc. and concurrently sets up a timeout. If the timeout is reached without receiving anything, then the test case has failed; otherwise, the element should forward the received message;

**RQ5: How to automate tests that depend on physical devices?** The IoT merges software with hardware, thus a need for testing physical devices arises. Automating the tests of such devices can provide automatic fault detection, reducing the time to repair an issue. The automation of tests for physical devices can be achieved by spying the behavior of the device, e.g., when a lamp is turned on, the brightness of a room should increase;

**RQ6: How to leverage tests to allow continuous regression testing?** An IoT system is considerably more prone to errors when compared to a normal software system, especially due to its high heterogeneity between devices and the fact that they may fail more frequently. Continuous regression testing eases this complication by constantly running tests that monitor if the system is behaving correctly. This can be executed by having a test suite that runs periodically or on-demand, allowing greater confidence in the behavior of the whole system.

## 6.4 Conclusions

This chapter illustrates the results of the validation of this particular implementation. Section 6.1 (p. 53) defines the test scenarios utilized to assess the tool's conformance to the requirements. Moreover, Section 6.2 (p. 55) analyzes how the solution fulfills requirements of the *desiderata*. Finally, Section 6.3 (p. 56) contains the answers to the research questions established before.

## Evaluation

## Chapter 7

# Conclusions

This chapter is comprised of various sections. Section 7.1 details the difficulties faced during the development of the solutions. Section 7.2 outlines the research contributions made to the field of IoT simulation and testing. Afterward, an overview of the achieved results is described in Section 7.3 (p. 60). Section 7.4 (p. 60) delineates directions to explore as they were not fully covered in this work and are seen to have some potential to create value and fill gaps in the research.

### 7.1 Difficulties

During the development of this project, numerous difficulties rose up and were tackled. While some were successfully solved, others were not and are presented as future work in Section 7.4 (p. 60). Regarding the IoT simulator, problems such as dealing with the `RETAIN` of the MQTT protocol and scaling up the proxy to multiple instances were left as open questions. However, the problems inherent to the one-queue architecture were dealt with by pivoting into a two-queue architecture. The testing framework also posed interesting issues, namely the communication between nodes and passing information about test results. These were eventually solved using regular Node-RED messages with special properties and utilizing the flow context, respectively. Nonetheless, a few limitations were identified that can be viewed as future improvements, as Section 5.3.3 illustrates.

### 7.2 Contributions

The implementation of a solution to solve the problems identified in Chapter 4 (p. 33) provides convenient tools as well as useful knowledge on the advantages and disadvantages of the decision taken during development. As such, the contributions of this work are as follows:

## Conclusions

**Systematic Literature Review of Hardware-in-the-loop Simulators for the IoT** A systematic literature review was carried out to understand the state of the art with regards to IoT simulators that allow the inclusion of hardware in the simulation process. The paper [BDRSF19] was submitted to the 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems;

**Hardware-in-the-loop IoT Simulator** The simulator, as described in Section 5.1 (p. 39), which supports hardware-in-the-loop and is programming language agnostic;

**Node-RED IoT Simulator Extension** The Node-RED extension that allows the use of the IoT Simulator in a visual programming environment;

**Node-RED Testing Framework** The framework for Node-RED that permits the testing of flows;

**Continuous Regression Testing** The Node-RED testing framework provides continuous regression testing, a technique which can be generalized into other environments;

**Visual Test Results** The testing framework is equipped with a real-time visual representation of the test results that give the user instant feedback on how the system compares to the expected behavior.

### 7.3 Conclusions

Chapter 3 (p. 11) mentions the results of a systematic literature review about the state of the art of simulators for the Internet of Things and identifies a gap in this specific field. The review of visual programming languages for the Internet of Things also returned several results, but once again discovered a promising research gap. Based on this void, Chapter 4 (p. 33) clearly defines the what issues currently exist and, furthermore, lists the desired features in a solution to such problems, while also identifying use cases for the tool. A set of research questions to be answered by this thesis is also established.

Chapter 5 (p. 39) builds on the two previous chapters to create solutions that fulfill the imposed requirements, filling the gap for an IoT simulator with hardware-in-the-loop support and a graphical testing framework for the IoT. Chapter 6 (p. 53) explicitly answers the research questions defined before and evaluates the implementation.

Finally, Chapter 7 (p. 59) identifies the difficulties faced, the contributions made as well as the future work that can be explored to further advance the field of IoT testing using VPLs.

### 7.4 Future Work

The solutions developed for this dissertation were satisfactory in the sense that they solved the urging problems, as identified in Section 4.1 (p. 33). However, the implementations are still a proof of concept and do not cover every possible use case, giving space for future improvements.



## Conclusions

The IoT simulator comes with some limitations, mostly due to the architecture of Node-RED, that were overlooked and should be addressed. Namely, the evident disconnection between normal nodes and the simulator nodes, which are mostly alone and do not have any inputs or outputs; and the lack of clear visual elements that illustrate the effect that simulator nodes have on normal nodes. The testing framework is a bit more polished but still has some rough edges. Specifically, the restrictions of the number of tests that can be run in parallel, the lack of separate environments for testing, etc.

It is clear these tools still have space for improvement and, in conjunction with the research questions that were not answered conclusively, there is potential work to be done.

## Conclusions

# References

- [AAZ09] Michele Amoretti, Matteo Agosti, and Francesco Zanichelli. Deus: A discrete event universal simulator. In *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 58:1–58:9, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [adv] Advisor advanced vehicle simulator. [Online; accessed 14-January-2019].
- [ard] Ardublock | a graphical programming language for arduino. [Accessed 4-Jun-2019].
- [atn] At&t flow - design and build solutions for the internet of things. [Accessed 5-Jun-2019].
- [aws] Iot device simulator | aws solutions. [Online; accessed 13-May-2019].
- [Bac05] M Bacic. On hardware-in-the-loop simulation. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 3194–3198, 2005.
- [BB94] Margaret M Burnett and Marla J Baker. A classification system for visual programming languages. *Journal of Visual Languages and Computing*, 5(3):287–300, 1994.
- [BBB<sup>+</sup>95] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, and P. Van Zee. Scaling up visual programming languages. *Computer*, 28(3):45–54, March 1995.
- [BBB<sup>+</sup>11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.
- [BCC<sup>+</sup>18] Ahcene Bounceur, Laurent Clavier, Pierre Combeau, Olivier Marc, Rodolphe Vauzelle, Arnaud Masserann, Julien Soler, Reinhardt Euler, Taha Alwajeeh, Vyas Devendra, Umber Noreen, Emilie Soret, and Massinissa Lounis. CupCarbon: A new platform for the design, simulation and 2D/3D visualization of radio propagation and interferences in IoT networks. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, volume 2018-Janua, pages 1–4. IEEE, 1 2018.
- [BDRSF19] Bernardo Belchior, João Pedro Dias, André Restivo, and Hugo Sereno Ferreira. A systematic review of internet-of-things simulators using virtual and physical devices. In *27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2019.

## REFERENCES

- [Bei18] Noud Beijer. Continuous Delivery in IoT Environments. In Agile Alliance, editor, *International Conference on Agile Software Development*, pages 1–7. Agile Alliance, 2018.
- [BML<sup>+</sup>18] Ahcene Bounceur, Olivier Marc, Massinissa Lounis, Julien Soler, Laurent Clavier, Pierre Combeau, Rodolphe Vauzelle, Loic Lagadec, Reinhardt Euler, Madani Bezoui, and Pietro Manzoni. CupCarbon-Lab: An IoT emulator. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, volume 2018-Janua, pages 1–2. IEEE, 1 2018.
- [BPC<sup>+</sup>14] Giacomo Brambilla, Marco Picone, Simone Cirani, Michele Amoretti, and Francesco Zanichelli. A Simulation Platform for Large-scale Internet of Things Scenarios in Urban Environments. In *Proceedings of the First International Conference on IoT in Urban Space, URB-IOT '14*, pages 50–55, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [BTN<sup>+</sup>18] Andreas Brokalakis, Nikolaos Tampouratzis, Antonios Nikitakis, Ioannis Papaefstathiou, Stamatis Andrianakis, Danilo Pau, Emanuele Plebani, Marco Paracchini, Marco Marcon, Ioannis Sourdis, Prajith Ramakrishnan Geethakumari, Maria Carmen Palacios, Miguel Angel Anton, and Attila Szasz. COSSIM: An open-source integrated solution to address the simulator gap for systems of systems. In *Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018*, pages 115–120, 2018.
- [Bur17] Miroslav Bures. Framework for Integration Testing of IoT Solutions. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1838–1839. IEEE, 12 2017.
- [CBBZ18] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. Internet of Things (IoT): Research, Simulators, and Testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 6 2018.
- [cha] Chai. [ Accessed 28-Jun-2019 ].
- [Cha05] R. N. Charette. Why software fails [software failure]. *IEEE Spectrum*, 42(9):42–49, Sep. 2005.
- [CMB<sup>+</sup>18] Alcidney Chaves, Ricardo Maia, Carlos Belchio, Rui Araujo, and Goncalo Gouveia. KhronoSim: A Platform for Complex Systems Simulation and Testing. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 2018-Sept, pages 131–138. IEEE, 9 2018.
- [Coh09] Mike Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 1st edition, 2009.
- [CRB<sup>+</sup>11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César A F De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

## REFERENCES

- [CRV15] I Culic, A Radovici, and L M Vasilescu. Auto-generating Google Blockly visual programming elements for peripheral hardware. In *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, pages 94–98, 2015.
- [CXL<sup>+</sup>14] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang. A vision of iot: Applications, challenges, and opportunities with china perspective. *IEEE Internet of Things Journal*, 1(4):349–359, Aug 2014.
- [DB97] Michael Downes and Marat Boshernitsan. Visual Programming Languages: A Survey. *Control*, (December):1–29, 1997.
- [DFG17a] Gabriele D’Angelo, Stefano Ferretti, and Vittorio Ghini. Modeling the Internet of Things : a simulation perspective. *arXiv*, 7 2017.
- [DFG17b] Gabriele D’Angelo, Stefano Ferretti, and Vittorio Ghini. Multi-level simulation of Internet of Things on smart territories. *Simulation Modelling Practice and Theory*, 73:3–21, 11 2017.
- [DFG18] Gabriele D’Angelo, Stefano Ferretti, and Vittorio Ghini. Distributed hybrid simulation of the Internet of things and smart territories. *Concurrency and Computation: Practice and Experience*, 30(9):e4370, 5 2018.
- [DGV04] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *29th annual IEEE international conference on local computer networks*, pages 455–462. IEEE, 2004.
- [DNP15] M. Dyk, A. Najgebauer, and D. Pierzchala. SenseSim: An Agent-Based and Discrete Event Simulator for Wireless Sensor Networks and the Internet of Things. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015.
- [DPB13] Dominique Dhoutaut, Benoit Piranda, and Julien Bourgeois. Efficient Simulation of Distributed Sensing and Control Environments. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 452–459, Beijing, 8 2013. IEEE.
- [dpw09] Devices profile for web services (dpws), 2009. [Online; accessed 15-January-2019].
- [FDGM17] Stefano Ferretti, Gabriele D’Angelo, Vittorio Ghini, and Moreno Marzolla. The quest for scalability and accuracy: Multi-level simulation of the Internet of Things. In *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, volume 2017-Janua, pages 1–8. IEEE, 10 2017.
- [FGR12] G. Fortino, A. Guerrieri, and W. Russo. Agent-oriented smart objects development. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 907–912, 5 2012.
- [FGRS17] Giancarlo Fortino, Raffaele Gravina, Wilma Russo, and Claudio Savaglio. Modeling and Simulating Internet-of-Things Systems: A Hybrid Agent-Oriented Approach. *Computing in Science & Engineering*, 19(5):68–76, 2017.
- [Gam95] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

## REFERENCES

- [GDGB16] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. iFogSim : A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things , Edge and Fog. *arXiv*, pages 1–22, jun 2016.
- [GMPE13] Pablo Gimenez, Benjamin Molina, Carlos E Palau, and Manuel Esteve. SWE Simulation and Testing for the IoT. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 356–361, Manchester, 10 2013. IEEE.
- [gra] Graspio cloudio. [Accessed 5-Jun-2019].
- [HLC<sup>+</sup>15] Son N. Han, Gyu Myoung Lee, Noel Crespi, Nguyen Van Luong, Kyoungwoo Heo, Mihaela Brut, and Patrick Gatellier. DPWSim: A devices profile for web services (DPWS) Simulator. *IEEE Internet of Things Journal*, 2(3):221–229, 2015.
- [HLR<sup>+</sup>08] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.
- [ibm] Rational rhapsody. [Online; accessed 14-January-2019].
- [ift] Ifttt helps your apps and devices work together. [Accessed 5-Jun-2019].
- [ine] Inet framework. [Online; accessed 16-January-2019].
- [iot] Iotify- develop full stack iot application with virtual device simulation. [Online; accessed 13-May-2019].
- [JJW17] Tobias Jung, Nasser Jazdi, and Michael Weyrich. A survey on dynamic simulation of automation systems and components in the Internet of Things. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 9 2017.
- [JKBL14] Beate Jost, Markus Ketterl, Reinhard Budde, and Thorsten Leimbach. Graphical programming environments for educational robots: Open roberta-yet another one? In *2014 IEEE International Symposium on Multimedia*, pages 381–386. IEEE, 2014.
- [JSW18] Tobias Jung, Payal Shah, and Michael Weyrich. Dynamic Co-Simulation of Internet-of-Things-Components using a Multi-Agent-System. *Procedia CIRP*, 72:874–879, 2018.
- [JUn] Junit 5. [Online; accessed 30-January-2019].
- [KC07] B. Kitchenham and S Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.
- [KPB<sup>+</sup>10] Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. Systematic literature reviews in software engineering-A tertiary study, 2010.
- [KT08] Stamatis Karnouskos and Mian Mohammad Junaid Tariq. An agent-based simulation of SOA-ready devices. *Proceedings - UKSim 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, pages 330–335, 2008.

## REFERENCES

- [LCI<sup>+</sup>88] F Ludolph, Y . Chow, D Ingalls, S Wallace, and K Doyle. The Fabrik programming environment. In *[Proceedings] 1988 IEEE Workshop on Visual Languages*, pages 222–230, 1988.
- [Llo] Robin Lloyd. Metric mishap caused loss of nasa orbiter. [Online; accessed 28-January-2019].
- [LODYJ13] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Yla-Jaaski. MAMMOTH: A massive-scale emulation platform for Internet of Things. *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, 3:1235–1239, 2013.
- [LWZ<sup>+</sup>13] Xiaoyu Li, Yuying Wang, Xingshe Zhou, Dongfang Liang, and Chenglie Du. An implementation towards integrated simulation of cyber-physical systems. In *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCoM 2013*, pages 789–796, 2013.
- [LXZ15] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, apr 2015.
- [mat] math.js | an extensive math library for javascript and node.js. [ Accessed 28-Jun-2019 ].
- [MBR15] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (iot). *IEEE Internet Initiative*, 1:1–86, 2015.
- [MCFL14] Cristyan Manta-Caro and Juan M. Fernandez-Luna. A discrete-event simulator for the web of things from an information retrieval perspective. In Velasquez-Villada C.E., editor, *2014 IEEE Latin-America Conference on Communications, IEEE LAT-INCOM 2014*. Institute of Electrical and Electronics Engineers Inc., 2014.
- [mcp] Hp labs: Mcpat. [Online; accessed 14-January-2019].
- [min] minibloq. [Accessed 4-Jun-2019].
- [MLBK14] Kamal Mehdi, Massinissa Lounis, Ahcene Bounceur, and Tahar Kechadi. CupCarbon: A Multi-Agent and Discrete Event Wireless Sensor Network Design and Simulation Tool, 2014.
- [mqta] Mqtt. [ Accessed 10-Jun-2019 ].
- [mqtb] Mqtt version 3.1.1. [Online; accessed 22-April-2019].
- [MRR<sup>+</sup>10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.
- [net] Ntk | netlabtk: Tools for tangible design. [Accessed 4-Jun-2019].
- [nih] What is hardware-in-the-loop? [Online; accessed 3-February-2019].
- [nod] Node-red. [Accessed 4-Jun-2019].
- [noo] Noodl. [Accessed 5-Jun-2019].

## REFERENCES

- [Nor16] Amy Nordrum. Popular internet of things forecast of 50 billion devices by 2020 is outdated, 2016.
- [ns2] nsnam. [Online; accessed 16-January-2019].
- [ns3] ns-3 | a discrete-event network simulator for internet systems. [ Accessed 14-May-2019 ].
- [ÖDE<sup>+</sup>06] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In :, page 8, 2006.
- [OMN] Omnet++ discrete event simulator. [Online; accessed 30-January-2019].
- [ope] Openmodelica. [Online; accessed 15-January-2019].
- [osm] brambilla/osmobility: Discrete event simulator for vehicular mobility. [ Accessed 14-May-2019 ].
- [psa] Actions and the actions panel in photoshop. [ Accessed 28-Jun-2019 ].
- [PVK15] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.
- [qua] Qualnet for network simulation | scalable networks. [ Accessed 17-May-2019 ].
- [Ray18] P.P. Ray. A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences*, 30(3):291–319, jul 2018.
- [rea] Reactive blocks | bitreactive. [Accessed 5-Jun-2019].
- [RH10] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.
- [rha] Working with the object execution framework (oxf). [Online; accessed 14-January-2019].
- [s4a] S4a. [Accessed 5-Jun-2019].
- [scr] Scratch - imagine, program, share. [Accessed 5-Jun-2019].
- [sen] Senscript (cupcarbon). [Online; accessed 14-January-2019].
- [sim] Simulink for system modeling and simulation. [Online; accessed 14-January-2019].
- [SKH18] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Internet of things: A comprehensive review of enabling technologies, architecture, and challenges. *IETE Technical Review*, 35(2):205–220, 2018.
- [swe] Why is the ogc involved in sensor webs? [Online; accessed 16-January-2019].
- [TLL<sup>+</sup>14] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. Bug characteristics in open source software. *Empirical Software Engineering*, 19(6):1665–1705, 2014.



## REFERENCES

- [VH08] András Varga and Rudolf Hornig. AN OVERVIEW OF THE OMNeT++ SIMULATION ENVIRONMENT. In *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*. ICST, 2008.
- [wyl] Wyliodrin - iot solutions for industry and education. [Accessed 5-Jun-2019].
- [zen] Zenodys | internet of things development platform. [Accessed 5-Jun-2019].
- [ZKHS17] Meftah Zouai, Okba Kazar, Belgacem Haba, and Hamza Saouli. Smart house simulation based multi-agent system and internet of things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, volume 2018-Janua, pages 201–203. IEEE, 12 2017.

## REFERENCES

## **Appendix A**

# **Systematic Literature Review**

This appendix contains the Systematic Literature Review submitted to the 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems.

# A Systematic Review of Internet-of-Things Simulators using Virtual and Physical Devices

Bernardo Belchior  
DEI  
Faculty of Engineering  
University of Porto  
Porto, Portugal  
up201405381@fe.up.pt

João Pedro Dias  
INESC TEC and DEI  
Faculty of Engineering  
University of Porto  
Porto, Portugal  
jpmdias@fe.up.pt

André Restivo  
LIACC and DEI  
Faculty of Engineering  
University of Porto  
Porto, Portugal  
arestivo@fe.up.pt

Hugo Sereno Ferreira  
INESC TEC and DEI  
Faculty of Engineering  
University of Porto  
Porto, Portugal  
hugosf@fe.up.pt

**Abstract**—The Internet-of-Things is comprised of billions of connected devices. IoT systems can have many components, creating a huge array of possible errors. The process of detecting those errors before they move into a production environment for IoT system is usually done using testbeds, simulators or emulators. This paper is a systematic review of simulators for testing Internet-of-Things systems focusing on hardware-in-the-loop simulation tools with support for automated testing. The papers were retrieved from the IEEEExplore, Scopus and Compendex publication databases. Of these, 15 met all the defined criteria. The analysis was complemented by two additional surveys. We concluded that, while these types of tools exist, a free or open-source tool with hardware-in-the-loop capabilities that can be automated was not found.

**Index Terms**—Internet-of-Things, simulation

## I. INTRODUCTION

The Internet-of-Things, the result of the inter-connectivity of physical and virtual entities which can sense and actuate in the real-world, has been growing massively. In 2016 there were between 6.4 billion and 17.6 billion devices (excluding mobile phones, tablets, and computers) connected to the Internet and it is predicted to reach numbers between 20 and 30 billion of connected devices [1].

Such scale makes IoT simulations a field to be explored due to its potential benefits regarding cost savings and time-to-prototype. Numerous tools for the job exist, although each focuses on very specific use cases, be it performance measurement, power cost monitoring, network modeling, etc. Due to the diversity of tools used to perform the same task, this paper compares them following defined criteria and then presents the results obtained.

The structure of this review is as follows: Section II provides an overview of some concepts regarded as important for the understanding of our work; Section III details the methodology followed to achieve the results presented afterward; Section IV explains and compares the results obtained from the conducted search; Section VII concludes the review by presenting an analysis of the retrieved results.

## II. BACKGROUND

This section introduces several pieces of knowledge deemed crucial to understand the rest of our work. Ranging from

an introduction to the Internet-of-Things to software testing, while providing a succinct explanation of systems and IoT simulation, we summarize the construction blocks for the rest of the survey.

### A. Internet-of-Things

The Internet-of-Things (IoT) is a paradigm where computational devices, typically known as *things*, are connected to the Internet and are uniquely identifiable and globally accessible [2]. IoT opens doors for using information from both the real and virtual realms to actuate in the real world. Within the nature of IoT systems, there are several particularities that, although not new or unique, come together at an unprecedented scale in terms of interconnected devices, people, systems, and information resources. These results in an ever-increasing systems' complexity that must be addressed by the IoT systems developers due to characteristics such as heterogeneity, logical and geographical distribution, interoperability requirements, real-time needs and scalability [3].

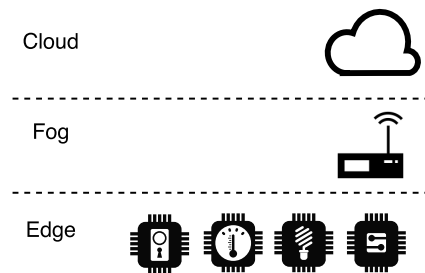


Fig. 1. Typical tier architecture of Internet-of-Things systems.

From an high abstraction standpoint, IoT systems typically follow a three-tier base architecture, depicted in 1. The bottom tier is constituted by the *things* – objects with computing and connectivity abilities – that run software that allows them to sense and actuate with the containing environment. The higher-tier is the cloud, constituted by the servers that store, process, and offer services of lower tiers. An intermediate tier, the fog tier, guarantees the connection between edge devices and the cloud while making some optimization's, e.g., in terms of the number of connections and data pre-processing [4].

The complexity of IoT systems affects both the design and development of such systems, but it also implies a greater complexity on their verification and validation. As an example, each of the IoT tiers has different responsibilities, constraints, and features, but all need to be considered when verifying and validating IoT systems. Most of the time, traditional approaches for testing software-only systems are mostly limited and insufficient, not covering, for example, the necessity of testing software as well as the hardware counterpart. Recent works point that both simulation and the use of testbeds became a go-to solution to verify and validate hardware-dependent and human-in-the-loop systems, such as IoT [5].

## B. Systems Simulation

Systems are defined both by their structure (i.e., components, relationships and attributes) and by their behaviour (i.e., functions, inputs and outputs and control operators). Systems have a *structures*, a *state* – a collection of variables necessary to describe a system at a particular time – and can have both continuous – the state variables change continuously with respect to time – and discrete – *the state variables change instantaneously at separated points in time* – behaviors [6]. Both aspects of the systems can be modeled into, respectively, structure models and behavior models [7]. These models can embrace the notion of time, as per Huang et al.: *the semantics of models are directly related to the passage of time and causality of outputs with respect to states and inputs* [8].

The use of models as abstractions of the systems structure and behaviour (that include both software as well as hardware concerns) have been long used to carry out the so called systems simulation. Simulation *per se* is the use of computers to imitate, *viz.* simulate, the operations of various kinds of system components or, even, sub-systems. Different kinds of simulations can be distinguished by the models they employ, that can be classified along three different dimensions [6]:

- *Static vs. Dynamic Simulation Models*: A static simulation model represents the system at a particular time while a dynamic simulation model represents a system while it evolves over time.
- *Deterministic vs. Stochastic Simulation Models*: Stochastic simulation models have probabilistic or random properties, at least from their inputs, while deterministic simulations have no randomness.
- *Continuous vs. Discrete Simulation Models*: Simulation models that follow the above-mentioned definition of discrete and continuous systems.

Simulations have long been used due to their advantages such as (1) providing practical feedback when designing real-world systems without the need of actually constructing them, (2) allowing systems to be studied at different levels of abstraction, (3) aiding in finding unexpected behavior without real consequences, (4) supporting the test of *What-If* situations [9]. All of these advantages attempt to improve the liveness of the testing tools [10].

## C. IoT Simulation

The Internet-of-Things is generally comprised of heterogeneous embedded devices running on different hardware, distinct operating systems and with various peripherals. This diversity in conditions, along with the scale of IoT systems create some challenges that are left to be answered, especially regarding standardization and security.

As a result of the growth of IoT, simulation is a promising field and currently being explored by many researchers. Its potential benefits, such as cost savings and time-to-prototype, are seen as desirable.

By simulating the Internet-of-Things, it is possible to test systems without actually purchasing the hardware, while also opening doors for a faster feedback loop when compared to using real hardware. However, the simulation of IoT networks, allowing the validation of models, protocols and algorithms before the actual deployment of the network infrastructure, is an important but complex task since the number of the simulated devices may vary from dozens (e.g., smart homes) to thousands (e.g. smart cities), with varying degrees of density and different communication paradigms. Moreover, factors unrelated to the applications but specifically associated to the networking (e.g., traffic congestion, wireless signal attenuation and coverage, etc.) influence the interactions between nodes [11].

1) *Common Architectures*: There are several architectures in common among IoT simulators. In this field, two seem to be more prominent than others:

- A **Multi Agent System (MAS)** is an environment where single, autonomous entities, called agents, act and interact with each other. Benefits of this architecture include autonomy for each agent and decentralization of the system. This model fits well with the IoT principle, where each device is uniquely identifiable.
- A **Discrete Event Architecture** is driven by events, and the state of the system is only known by deriving it from a series of events. It is often used in conjunction with state machines, where a new state is the result of a function taking the current state and an event. This maps well to the IoT concepts as an Internet-of-Things system can be seen as a stream of data (events) being collected sequentially.

2) *Key Concepts*: Terms and expressions such as *hardware-in-the-loop simulation* and *multi-level simulation* are explained in the following section:

- **Hardware-in-the-loop simulation** is a type of simulation is used to test physical devices. It tricks the hardware into thinking that it is part of a system, either by connecting the hardware to the simulator and sending electrical signals [12] or by simulating how the physical system would communicate with the device under test [13].
- **Multi-level simulation** is a paradigm where different devices can be simulated at different levels of detail. One device could have its application layer simulated, but another could have its application as well as its network

## Systematic Literature Review

and/or physical layers simulated. This idea is useful in large-scale simulations where it is not feasible to simulate every aspect of every node with the highest level of detail. Instead one could opt to simulate a smaller part of the system with finer detail. A multi-level simulation tool provides the user with the option to scale between simulation speed and simulation correctness [14].

IoT simulation builds upon the three different dimensions pointed out in Section II-B and defines a fourth one. A simulation model in an IoT system can be comprised of physical or simulated devices, but also by a mixture of them, i.e., hardware-in-the-loop simulation. Unlike the other dimensions, which are binary (e.g., a simulation model is either static or dynamic), the fourth dimension can range from an all-hardware to a software-only simulation, and everything in between, viz. mixed-simulation.

### D. Testing

Testing, specifically software testing, is the process of ensuring that said software does not behave in manners not expected by its developer or owner. The importance of testing comes from the possible catastrophic accidents that can happen, such as a plane crash due to integer overflows or the loss of a \$125 million satellite due to unit conversion errors [15]. Testing can help prevent these disasters by reducing the number of error scenarios that were not accounted for. In order to achieve this, there are various types of test that can be performed:

- The purpose of **Unit tests** is to verify small units of code and their correct functionality. They are usually fast, which is why, according to the test pyramid [16], they should be the most numerous in a software test suite.
- **Integration tests** focus on the interaction between components. They usually take longer than unit tests. On the other hand, they provide more confidence to the tester that a specific set of components are working. These should appear in less quantity than unit tests due to their slowness.
- **End-to-end tests** provide the most confidence to the tester as they test the whole system. These are more often than not slow and costly. Therefore, their usage should be limited to critical paths of execution in a program.

While most of these tests can be executed manually, automated testing is becoming more widespread between the software development community, mainly due to its efficiency. Using automated tests, regression testing becomes easier, since the developer does not need to keep testing the system after every change, instead relying on an external program to repeat the steps the developer would have taken. Besides, by having automated tests, more use cases can be tested after each change, possibly discovering faults the developer could not have thought about.

## III. METHODOLOGY

In this systematic review we follow a specific and objective methodology to reduce bias [17]. The first step taken was to define the research questions to answer and the data sources

to utilize. Afterwards, the search query was constructed, and the inclusion and exclusion criteria laid out.

### A. Research Questions

In this work, we intend to address the following questions:

- RQ1. What relevant IoT **simulation platforms** exist?
- RQ2. Of these, which ones support testing and validation with **hardware-in-the-loop**?
- RQ3. What is the **scope** of the tools found in RQ1?
- RQ4. Of those found in RQ1, which ones support the **automation** of tests?

In this context, *relevant* means publications not excluded by the methodology explained in the next sections. By *scope*, we mean the specific use case, if any, that each tool focuses on. For abbreviation purposes, *simulation* refers to any kind of simulation, emulation, or another type of approach that allows the test and validation of hardware/software IoT systems.

### B. Publication Databases

For this research, publications were retrieved from the following databases, which are regarded as good sources for software engineering [18]:

- IEEEExplore Digital Library
- Scopus
- Compendex

### C. Search Process

A search query was created to narrow down the results. It was designed to include the field of research, Internet-of-Things, combined with the more specific part the paper will review: simulation. The search expression used was:

```
(internet-of-things OR IoT OR
"Internet-of-Things") AND (simulator
OR (simulation AND (tool OR library OR
framework)))
```

The search was made in December 2018 and revealed the results available in I. The downloaded results were the ones deemed "most relevant" by each of the publication databases.

TABLE I  
SEARCH RESULTS PER DATABASE

Databases	Filters	Total Results	Downloaded Results
Compendex	Subject/Title/Abstract, English only	3962	500
Scopus	Title, Abstract, Keywords	1510	400
IEEEExplore	All	1422	500

### D. Inclusion Criteria

Publications are put through the inclusion criteria to define whether or not they should be included in the results. If a publication does not meet *all* of them, it should not be included. The inclusion criteria are the following:

## Systematic Literature Review

- 1) Original research study (including patents and grey literature for completeness);
- 2) Review papers;
- 3) Publications of code testing simulation platforms for the Internet-of-Things;
- 4) The study content must be in English.

It should be noted that, although review papers are included, they are processed differently. After reviewing the simulation tools retrieved from this systematic review process, the surveys will be studied to include other simulators that were not covered by the methodology. This *expanded search* allows a critical analysis of the selection process.

### E. Exclusion Criteria

Papers are filtered by understanding whether or not they violate any of the exclusion criteria. If they do fail to comply with one criterion, they will be excluded. The exclusion criteria are defined as follows:

- 1) Secondary research and other non-relevant publications;
- 2) Publications presenting just ideas, magazine publications, interviews, and discussion papers;
- 3) Duplicate publications;
- 4) Publications on the topic of Internet-of-Things testing whose only focus is security, privacy, power efficiency or performance.

### F. Selection Process

The publication selection process follows three steps to obtain the final result:

- 1) Check if the publication's title meets the inclusion and exclusion criteria;
- 2) Verify if the publication's abstract meets the inclusion and exclusion criteria;
- 3) Read the whole content of the paper and analyze whether or not it meets the inclusion and exclusion criteria.

The results of each phase can be seen in Table II.

TABLE II  
PUBLICATIONS PER STEP

Step	No. of Publications	Excluded
Search	1400	N/A
No Duplicates	1206	194
Inclusion/Exclusion Criteria (Title)	62	1144
Inclusion/Exclusion Criteria (Abstract)	32	30
Specificity	20	12

## IV. RESULTS

Of the 20 selected publications, [19] and [20] were surveys on simulators. Jung et al. [19] surveys various IoT simulation frameworks in search of dynamic (i.e., possibility to integrate new entities during the simulation) and decentralized tools that allow for heterogeneous simulation (i.e., simulation using

different simulators) of devices. After finding none, a conceptual agent-based solution is proposed with hardware-in-the-loop capabilities. Chernyshev et al. [20] perform an in-depth review of simulators and testbeds for the Internet-of-Things with each tool being assessed following several criteria such as the evaluated scale of the simulated system, last update to the tool, built-in IoT standards, etc. Since simulators are generally specialized in one aspect of IoT simulation, the authors conclude that a combination of the distinct IoT simulation tools may provide simulation with higher fidelity [20]. However, this review takes on the challenge of identifying the different types of tests and programming languages that each simulator supports, while also analyzing if they can run automated testing and to integrate hardware-in-the-loop.

The remaining 18 publications were simulation frameworks, 4 of which refer to the same tool, producing 15 unique results. These are:

- 1) The cyber-physical system (CPS) simulator proposed by **Li et al.** [21] uses the concepts of physical, computation, and interaction entities to achieve simulation of a CPS. A *physical entity* represents the physical aspect of a device, with spatial and temporal awareness, and is usually based on laws of physics. It is modeled using Simulink [22]. Computation entities are modeled as finite state machines using UML in IBM Rational Rhapsody [23]. There is also an *interaction entity* that works as the interface of the two other entities, effectively allowing them to communicate with each other. Every component is simulated separately, allowing for model heterogeneity, but there is no mention of support for the integration of real and simulated devices. IBM Rational Rhapsody supports OXF [24], allowing C++ code not generated by Rhapsody to be used in the simulation. Because of this, C++ can be considered as a supported language.
- 2) **COSSIM** [25] is an IoT emulator capable of handling networks, processors and peripherals, while also providing power consumption information and allowing security audits by integrating gem5 [26], OMNeT++ [27] and McPAT [28] tools under a single interface. The integration allows the "simulation of an actual system of systems, including its complete software stack, network dynamics and energy aspects" [25]. As such, it is regarded as an emulator. The framework takes care of the synchronization between the different simulators under the hood. In [25], the tool's performance, and correctness are assessed using two different case studies.
- 3) **CupCarbon** [29] is a tool to "design and simulate Wireless Sensor Networks dedicated to Smart-city and IoT applications" [29]. It is comprised of a radio channel, which integrates two radio propagation models, an interference module, randomly generating communication interference in networks and a 2D/3D environment, useful for designing smart territories, e.g., smart cities. The framework supports the visualization of the sim-

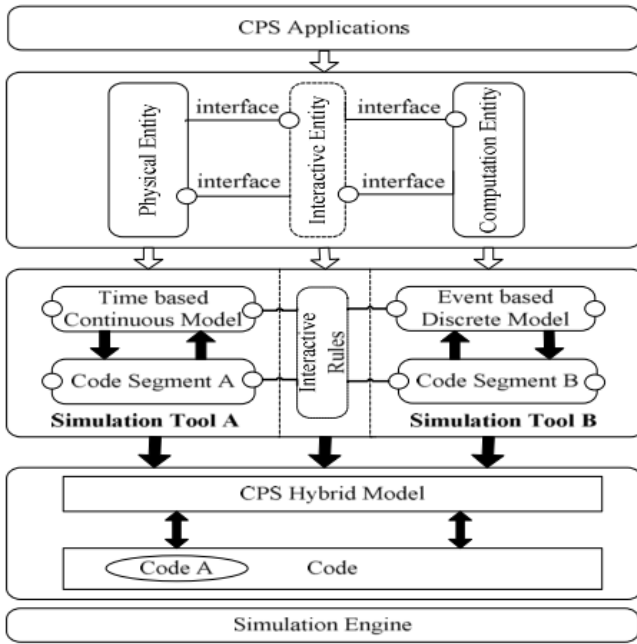


Fig. 2. CPS simulator architecture by Li et al. [21]

ulation results while modeling radio propagation and interference.

- 4) **CupCarbon-Lab** [30] is an extension of the CupCarbon simulator and uses less powerful devices, such as the Raspberry Pi or Android smartphones, to act as a network of physical nodes, providing hardware-in-the-loop simulation. It uses the host computer as a server where the system can be monitored and visualized. This host functions as the remote control of the simulation environment and is also responsible for injecting code into the physical devices.
- 5) A multi-level simulator proposed by **D'Angelo, Ferretti et al.** [14] [31] [32] [33] makes use of different established simulators to simulate a whole system with varying levels of detail. It uses an agent-based simulator (Smart Shire Simulator [32]) for the high-level, low-detail simulation and a transportation system simulator based on ADVISOR [34] and a network simulator (OMNeT++ [27]) when higher levels of granularity are needed. Despite using three distinct simulation tools, the framework has only two levels of simulation, as the transportation system and network simulators simulate mutually exclusive devices, i.e., a device is either simulated by one or the other.
- 6) **DPWSim** [35] is a simulator using the standard Devices Profile for Web Services (DPWS) [36]. DPWSim is bundled with (1) DPWS Explorer, a tool for analyzing and exploring DPWS service and (2) DPWSim Web, a web interface for controlling DPWSim devices on different devices, e.g., mobile phone. DPWSim follows a discrete event architecture where devices have *events*

that they fire or *operations* that can be executed. The only way to customize device functionality is either through its GUI or a .dpws file.

- 7) The agent-based domain-agnostic solution presented by **Jung et al.** [37] applies a different architecture allowing runtime changes and integration of distinct simulation tools simultaneously. The simulator requires a communication and synchronization interface between the varying simulation frameworks so that they can communicate and behave as expected. In [37] a communication interface was built for Simulink [22] and OpenModelica [38] to permit agents simulated in both tools to communicate with each other. Due to this integration, the tool allows for hardware-in-the-loop testing, given that the underlying simulators in use have support for it. The publication has no mention about whether or not the tool can be automated.
- 8) **VisibleSim** [39] is a discrete event simulator aimed with a 3D environment that can display objects (exported at .obj files) or simple blocks. It can simulate forces applied by objects on others using a physics engine. According to [39], one of its main advantages is the ability to simulate 2 million nodes at a rate of 650,000 events per second on a Xeon-based processor (base-frequency of 2.56GHz) with 12GB of RAM.
- 9) **IoTTest**, proposed by Bures [40], is an approach to device simulation that focuses on integration and end-to-end (e2e) testing by borrowing technical concepts from JUnit [41]. It is capable of emulating some IoT devices while integrating real physical hardware into the system, effectively providing hard-in-the-loop simulation.
- 10) **KhronoSim** [42] is a distributed data-oriented simulator for testing the IoT. It provides "progressive development" [42] by allowing the user to convert simulated nodes into physical devices progressively. It's divided into three main components: (1) Launcher, acting as a lock on the system under test; (2) Runners, that are responsible for executing a test suite and must wait on Launcher's permission to run; and (3) Executive, which, in the end, executes the commands (such as sending messages and reading electrical signal) requested by a Runner. KhronoSim [42] can be used in conjunction with Simulink [22], effectively giving it spatial and temporal context awareness.
- 11) **MAMMoTH** [43] is a massive-scale Internet-of-Things emulator, with the aim of emulating 20 million devices. It uses a distributed architecture based on virtual machines (VMs), which can emulate 10,000 nodes per VM and uses ns-2 [44] models for emulating network protocols.
- 12) The simulator proposed by **Fortino et al.** [45] makes use of a multi-agent system to model nodes in a network. Using the INET Framework [46] for OMNeT++ [27], the simulator can model Wireless Sensor Networks (WSN), while the integration of ACOSO [47] provides the platform to develop and run the multi-agent system.



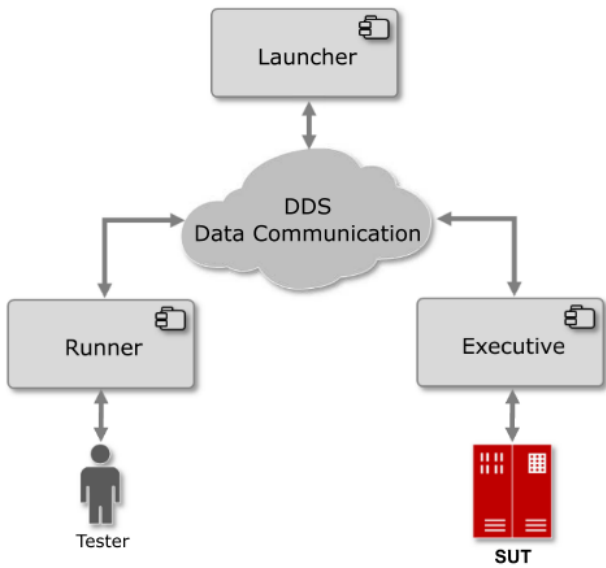


Fig. 3. KhronoSim high-level architecture [42]

- 13) The tool presented by **Zouai et al.** [48] is an agent-based simulation tool for smart houses, capable of controlling smart devices from an external network by using an IoT Gateway that connects the house to the Internet.
- 14) The **SWE Simulator** proposed by **Gimenez et al.** [49] uses a Sensor Web Enabled [50] (SWE) architecture component, Sensor Observation Service (SOS), that serves as a sensor information database, providing standard data insertion and retrieval methods, via HTTP, for all sensors. The tool also features a Control Center, where actions are handled as responses to events and where a web platform, used for monitoring the system, is served from.
- 15) **WoTSIM** [51] is a Web of Things (WOT) simulator allowing static, moving, and offline devices. Things are described as XML configuration documents at the start of the simulation, and more XML documents are created during runtime and stored in "Real-Time IR Test Collection" [51]. Running a simulation with 6000 sensors for 19 hours resulted in a 41.1 MB collection size [51].

We divided the testing platforms into the following categories, according to their characteristics:

**Type** The type of tool: simulator or emulator. Possible values: *Simulator* or *Emulator*

**Scope** Some tools are built for a specific use case (e.g., smart cities, smart grids, etc.). Therefore, knowledge of the scope of each platform is useful to assess whether or not they may solve a problem one may have. Example values: *smart city*, *smart grid*, *network*, *device*.

**Architecture** Simulators for the Internet-of-Things tend to gravitate towards Multi-Agent Systems or Discrete Event architectures, with each one having its benefits and dis-

advantages.

**License** The license of a software product may impede its utilization or be a strong point in favor of its adoption. For example, an open source simulator allows for high degrees of customizability by giving users the power to change and alter the software. Possible values: the license's name or *N/A* if the software license is not known.

**Scalability** Defines the how the platform scales. Due to the different performance metrics used by authors, this characteristic is only an approximation. Possible values are *very low*, *low*, *medium*, *high* or *very high*; or *N/A* if there is no information regarding the scalability of the tool.

**Tier** IoT systems usually follow the three tier architecture, as explained in Section II. Aggregating the simulation of all tiers under a single simulator is generally out of scope for most tools, so the project usually selects one tier to focus on, which is what this parameter describes. Its values can be the permutations of *Cloud*, *Fog* and *Edge*.

**Context Awareness** The context of a simulator represents whether or not the simulated devices know about their time and/or spatial position. Possible non-mutually exclusive values: *Spatial*, *Temporal*.

**Hardware-in-the-loop** A simulator is said to be a hardware-in-the-loop simulator if it can support physical devices [13]. Such a tool has the advantage of testing a system closer to the real environment, which provides more confidence in its correctness. Possible values: *yes*, if the software supports physical and virtual devices simultaneously; *no* otherwise;

**Can be Automated?** Automated testing reduces the time spent by developers testing software and, therefore, platforms with this capability are advantageous for their users. Possible values: *yes*, *no* or *N/A* if there is no information available.

**Programming Language** Programming language(s) that are required for the usage of the simulator.

## V. EXPANDED SEARCH

The systematic review protocol results are detailed in Section IV. However, when comparing these tools to non-systematic surveys [19] [20], these display some additional information that was not found by the search terms established. Multiple reasons may apply for this divergence in results, such as:

- 1) the tools not having an associated academic publication (e.g. AWS IoT Device Simulator [53], IoTIFY [54], obtained from Google by searching *IoT Simulator*), causing them not to be returned by the publication databases outlined in Section III.
- 2) the publications may not use the keywords *IoT* or *Simulator* — including variants —, but rather employ *Wireless Sensor Networks* or *Emulation*, e.g., CupCarbon [55], resulting in them not being included in the search results.

## Systematic Literature Review

TABLE III  
SIMULATORS AND THEIR PROPERTIES. N/A STANDS FOR *Information Not Available*.

Solution	Type	Scope	Architecture	License	Scalability	Tier	Context Awareness	Hardware-in-the-loop	Can be Automated?	Programming Languages
Li et al. [21]	Simulator	Device	Event-based	N/A	N/A	Edge	Spatial, Temporal	N/A	N/A	C++ <sup>a</sup>
COSSIM [25]	Emulator	N/A	Event-based	BSD 2-Clause	N/A	Edge	N/A	N/A	N/A	Any
CupCarbon [29]	Emulator	Smart City	Event-based	No	N/A	Fog	Spatial	N/A	N/A	Sen-Script [52] or Python
CupCarbon-Lab [30]	Emulator	Smart City	Event-based	N/A	N/A	Fog	Spatial, Temporal	Yes	N/A	Sen-Script [52] or Python
D'Angelo, Ferretti et al. [14]	Emulator	Smart City	Agent-based	N/A	High	Fog	Spatial, Temporal	N/A	N/A	N/A
DP-WSim [35]	Simulator	N/A	Event-based	No	Low	Edge	Spatial	N/A	N/A	None <sup>b</sup>
Jung et al. [37]	Simulator	Device	Agent-based	N/A	N/A	Edge	Spatial, Temporal	Yes	N/A	Any <sup>c</sup>
VisibleSim [39]	Simulator	Virtual World	Event-based	Apache 2.0	Very High	Edge	Spatial, Temporal	N/A	N/A	C++
IoTTest [40]	Emulator	N/A	N/A	N/A	N/A	Edge	N/A	Yes	Yes	N/A
KhronoSim [42]	Simulator	N/A	N/A	N/A	N/A	Edge	Spatial, Temporal	Yes	Yes	Any
MAM-MotH [43]	Emulator	N/A	N/A	N/A	Very High	Edge, Fog	N/A	N/A	N/A	N/A
Fortino et al. [45]	Emulator	N/A	Agent-based	N/A	High	Fog	N/A	Yes	N/A	N/A
Zouai et al. [48]	Simulator	Smart House	Agent-based	N/A	N/A	Fog	Spatial, Temporal	N/A	N/A	N/A
SWE Simulator [49]	Emulator	N/A	Event-based	N/A	N/A	Edge, Fog	Spatial, Temporal	Yes	N/A	Any <sup>d</sup>
WoTSIM [51]	Simulator	N/A	Event-based	N/A <sup>e</sup>	High	Edge	Spatial, Temporal	N/A	N/A	N/A

<sup>a</sup>Although C++ does not have first class support, the paper states that "C++ code generated in different environments can communicate through interface under Rhapsody/OXF support, thus implement integrated co-simulation of physical and computation processes." [21]

<sup>b</sup>Sensors are programmed using the DPWSim GUI [35].

<sup>c</sup>Supports the programming languages used by the underlying simulation systems.

<sup>d</sup>Since data is inserted and retrieved using HTTP virtually any language can be used

<sup>e</sup>Reference [51] states the simulator would be open-sourced, but its code could not be found at the time of this writing.

### A. Expanded Results

To complete this survey, the unique results from the found reviews [19] [20] were selected. From this selection, the tools were assessed against the selection process defined in Section III-F and reviewed according to the parameters established in Section IV. Using the aforementioned methodology, the following tools remain:

- 1) The Java-based urban IoT simulator presented by **Brambilla et al.** [56] unites the COOJA [57] and ns-3 [58] network simulators using DEUS [59], a discrete event simulator, to aggregate information, such as transmission delays, energy consumption and schedule communication events [56]. The OSMobility [60] simulator is included to allow the simulation of vehicles and pedestrian

within the urban environment.

- 2) **Karnouskos et al.** [61] proposes a DPWS-based [36] multi-agent enterprise-grade simulator with the goal of providing transparent simulation, mobility support, and dynamic environments capabilities, while also supporting micro and macro simulation and *self-X* (e.g., self-configuration, self-healing, etc.) behavior [61]. The tool uses agents that represent individual simulated devices as well as physical devices, while also providing a communication bus so that the transmission of information can be made regardless of the type of device, i.e., if the device is simulated or not [61].
- 3) **SenseSim** [62] is a simulator for the Internet-of-Things based on agents and discrete events that, using its

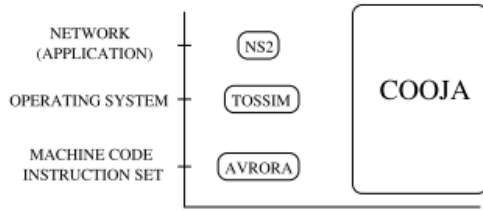


Fig. 4. COOJA can run at multiple simulation levels [57]

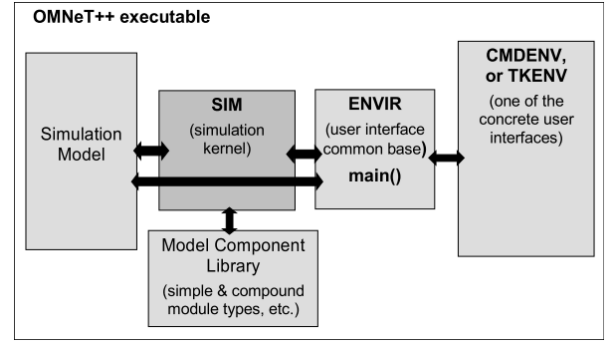


Fig. 5. OMNeT++ Internal Architecture [66]

layered architecture, intends to be capable of running the same application regardless of whether the devices executing it are simulated or not. To achieve such a goal, each simulated sensor is composed of four different components handling distinct parts of the logic: from changing the sensor's internal state or providing an API to manipulate to the sensor to a middleware that runs an application on the sensor [62].

- 4) **iFogSim** [63] is an extension of CloudSim [64], and so inherits its functionalities. However, unlike CloudSim, which is only focused on the cloud, iFogSim adds a novel *edge-ward placement* [63] strategy. This strategy favors the use of edge and fog devices over the cloud. Its GUI allows the creation of a network with edge, fog and cloud devices and its connections. A special class represents not only a communication packet but also defines the processing requirements and the length of the data it encapsulates. This characteristic is what allows the edge-ward placement strategy to decide whether or not to route the tuple to a device closer to the edge.
- 5) **COOJA** [57] is a sensor network simulator for the Contiki [65] operating system. It emulates devices running the code without needing modifications so that transferring to a physical device can be made painlessly. COOJA also emulates radio models, allowing different types of radio wave propagation. COOJA makes use of specific features of the Contiki kernel (e.g., its event-driven nature) to emulate devices, which makes it incompatible with other operating systems.
- 6) **OMNeT++** [27] is a discrete event simulator for communication networks and distributed systems containing not only a GUI but also a domain-specific C++-like programming language named *NED* for defining the network topology. The behavior of models is written in C++ in a separate file from the topology. While the simulator is not tailor-made for the Internet-of-Things, the fact that it can model the topology of a network and is event-based makes it suitable for this use case.
- 7) **ns-3** [67] is a network simulator with the aim of succeeding the widely used ns-2 [44], but with added realism and ability to plug real devices into the simulation. ns-3 also provides C++ and Python API to plug into, while also supporting a distributed simulation setting for large scale simulations [68]. Although being lower-level

when compared to other simulators, when performance and network emulation is needed, ns-3 seems to be a promising solution.

- 8) **QualNet** [69] is a commercial network emulator with spatial awareness that allows the building of large wired and wireless networks and visualization by plugging into other tools. It is also possible to connect QualNet to real networks so that real hardware can be tested.

## VI. ANALYSIS & DISCUSSION

It is essential to take note that more tools were not included because they did not meet the criteria defined in Section III. Specifically, tools that focus solely on simulating the cloud part of an IoT system were left out, while platforms that simulate the fog and the edge were included.

### A. Result Analysis

Having summarized the results from not only the systematic review but also from the surveys found, there are a total of 23 simulators for the Internet-of-Things available. It is possible to verify that they vary significantly in terms of characteristics, from emulator to event-based, from highly scalable to supporting hardware-in-the-loop. The tools found were collected and analyzed. The results from such analysis are laid out below:

**Type** Regarding whether the tools are simulated or emulate systems, ten simulators were found contrasting with the 13 emulators discovered.

**Scope** From the 23 platforms identified, the most common is scope is smart cities, followed closely by network simulators. The others either have little representation or have no particular scope and are deemed as generic purpose.

**Architecture** When it comes to architecture, event-based tools are the most common ones, while the agent-based simulators, as well as the ones without mention of their architecture, appear only six times. There is also a case where the framework follows both an event- and an agent-based architecture.

**License** Most of the tools found do not mention a license, the few that do are split between an open source one (e.g.,

## Systematic Literature Review

TABLE IV  
SIMULATORS AND THEIR PROPERTIES. N/A STANDS FOR *Information Not Available*.

Solution	Type	Scope	Architecture	License	Scalability	Tier	Context Awareness	Hardware-in-the-loop	Can be Automated?	Programming Languages
Brambilla et al. [56]	Emulator	Smart City	Event-based	N/A	High	Edge, Fog	Spatial, Temporal	N/A	N/A	Java
Karnouskos et al. [61]	Simulator	N/A	Agent-based	N/A	High	Edge, Fog	Spatial	Yes	N/A	Any
Sens-eSim [62]	Simulator	N/A	Agent-based & Event-based	N/A	N/A	Edge	Spatial, Temporal	Yes	N/A	N/A
iFogSim [63]	Simulator	N/A	Event-based	No	N/A	Cloud, Fog	N/A	N/A	N/A	Java
COOJA [57]	Emulator	N/A	N/A	Contiki <sup>a</sup>	Medium	Edge	Spatial	N/A	N/A	C
OM-NeT++ [66]	Emulator	Network	Event-based	Academic <sup>b</sup>	N/A	Edge	N/A	Yes	Yes	C++/NED
NS-3 [67]	Emulator	Network	N/A	GNU GPLv2	Very High	Edge	N/A	Yes	N/A	C++/Python
QualNet [69]	Emulator	Network	N/A	N/A	High	Edge	Spatial, Temporal	Yes	N/A	N/A

<sup>a</sup><http://www.contiki-os.org/license.html>

<sup>b</sup>Academic Public License: <https://omnetpp.org/intro/license>

GNU GPLv2, BSD 2-Clause, etc.) and no license. There is a clear gap in the open source simulation tools with permissive licenses.

**Scalability** The majority of solutions identified, do not have their scalability metric evaluated from performance benchmarks. The few that include such measurement are usually the ones high or very high and present it as an advantage of the tool. This may explain why there are many platforms with high scalability, while there are only a few that do not scale very well.

**Tier** It is possible to verify that most focus on the Edge tier of an IoT system. A substantial number of tools simulate only the fog tier, and there are a few that can support both tiers. Only one solution can simulate the cloud: the reason for that is that publications with tools that *only* have the ability to simulate the cloud tier have been excluded, as explained in Section VI.

**Context Awareness** Regarding context awareness, i.e., whether device are aware of time and their position, most tools support spatial awareness, with some also providing temporal awareness. However, there are also a few platforms that do not explicitly mention to which extent they provide simulated devices with context awareness.

**Hardware-in-the-loop** Out of the 23 solutions analyzed, 11 provide some way of connecting the simulation to real hardware, while 12 do not mention whether or not it is supported.

**Can be Automated?** The automation of simulations tools can be useful for integrating autonomous processes such as test pipelines or batch operations. However, only IoTTest [40], KhronoSim [42] and OMNeT++ [66] are

capable of such automated procedure.

**Programming Language** Tools regularly introduce restrictions such as the programming language to code the devices with so that higher performance gains can be achieved. While performance is regarded as good, the restriction can also be too limiting and prohibit users from choosing a certain platform. From the programming languages that the discovered simulators support, C++ repeatedly emerges, as well as Java. There are also some frameworks that are programming language agnostic and some others that require programs to be written in a language created just for the simulator.

### B. Evolution of Publications

To understand how the field of simulation for the Internet-of-Things, it is important to perceive how it is developing. One metric to be used is the number of publications of the field over the years. Starting in 2006, the year the first IoT simulation tool present in the survey was published, and until 2019, publications were aggregated and their frequency measured.

### C. Research Questions

The research questions posed in III serve as guidance for the development of this publication, and its answers are crucial to assess the state of the art of the field of IoT simulators. Such answers are provided below.

**RQ1 What relevant IoT simulation platforms exist?** As seen in Section IV and V, there are multiple IoT simulation platforms with different architectures, standards, and goals. There are two most common approaches for simulation of IoT systems that keep appearing in diverse solutions, namely: discrete event simulations (DES) and

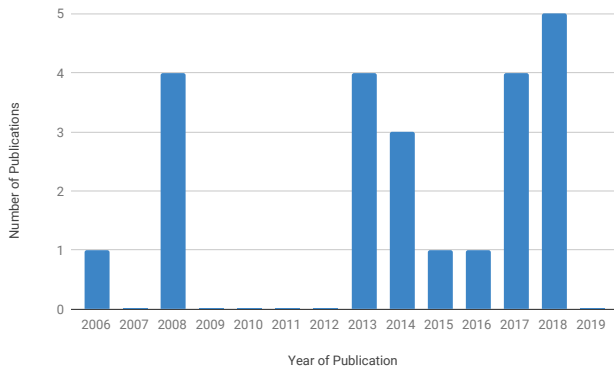


Fig. 6. Publications per year of IoT simulators

agent-based simulations (MAS). It can also be verified that most solutions are not open source or available for free download. When it comes to programming languages, there are different options, although most publications do not mention their requirements.

**RQ2 Of these, which ones support testing and validation with hardware-in-the-loop?** Table IV gives an overview of simulators and their properties. One can see that from the 15 solutions presented; only 6 provide hardware-in-the-loop simulation. The coexistence of virtual and physical devices offers better confidence about the system under test, as it is closer to the real environment the system will run in. Taking a look at Table V-A, it is possible to verify that most tools support hardware-in-the-loop, totaling 11 out of 23.

**RQ3 What is the scope of the tools found in RQ1?** Different tools focus on distinct use cases and objectives, ultimately leading to the developers limiting the scope of said tools in different ways. Some tools prefer to focus on emulating networks, including their quirks and peculiarities, while others pick the simulation of smart cities as their goal. This variety in scope influences the user selecting a simulation platform, justifying the importance of having this distinction among the available simulators. From the 23 tools found from the systematic and expanded searches, four focus on smart cities, three on network simulation, two on modeling devices, one on smart worlds and another one in smart houses. The remaining have a generic purpose, and so their scope has not been restricted.

**RQ4 Of those found in RQ1, which ones support the automation of tests?** Automated tests are becoming more prominent in the software industry, but the IoT field still seems like it is lagging on this matter. Out of 15 solutions found in the systematic search, only IoTTest [40] and KhronoSim [42] support automated testing. Moreover, the expanded search only found OMNeT++ [66] to support such feature, out of 8 total tools.

## VII. CONCLUSIONS

In this work, we survey 1400 papers from IEEEExplore, Scopus and Compendex that result in 15 simulators for the Internet-of-Things. Furthermore, an expanded search is conducted using the reviews found among the publications retrieved from the databases, totaling 23 IoT simulators. The results show that automated simulation of the Internet-of-Things with hardware-in-the-loop support is still limited, with only IoTTest [40], KhronoSim [42] and OMNeT++ [66] possessing these capabilities. Although these are good news for the field of simulation of the IoT, of these simulators, the first two are neither open source nor free to use, while the third is a network simulator and has no particular focus on the Internet-of-Things. As such, there is a clear lack of an open source hardware-in-the-loop IoT simulator capable of being automated, with OMNeT++ being the only possible solution, even though it is not ideal.

Summarizing, we noticed there is no broad range of solutions for testing IoT systems beyond the simulation-based and some testbed-based approaches. However, when analyzing the landscape of testing solutions for software-only systems, we verify that there is a lack of similar tools for IoT based systems. Especially when taking into account that IoT systems have unique characteristics and limit the use of software-only testing tools out of the box [70]. Future work can focus on developing an IoT simulator that (1) is open source, (2) can be automated and (3) supports hardware-in-the-loop.

## REFERENCES

- [1] A. Nordrum, "Popular internet of things forecast of 50 billion devices by 2020 is outdated," 2016. [Online]. Available: <https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated>
- [2] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the internet of things (iot)," *IEEE Internet Initiative*, vol. 1, pp. 1–86, 2015.
- [3] R. Buyya and A. V. Dastjerdi, *Internet of Things: Principles and Paradigms*. Elsevier, 2016. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/C20150041351>
- [4] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.
- [5] J. P. Dias, F. Couto, A. C. R. Paiva, and H. S. Ferreira, "A brief overview of existing tools for testing the internet-of-things," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, April 2018, pp. 104–109.
- [6] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis*. McGraw-Hill New York, 2000, vol. 3.
- [7] M. Austin, "Modeling system structure and system behavior," 2012.
- [8] Dongping Huang and H. Sarjoughian, "Software and simulation modeling for real-time software-intensive systems," in *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, Oct 2004, pp. 196–203.
- [9] M. Xiannong, "Simulation courseware," 2002.
- [10] H. S. Ferreira, J. P. Dias, A. Aguiar, A. Restivo, and F. F. Correia, "Live Software Development: Tightening the feedback loops," *5th Programming Experience Workshop*, pp. 1–5, 2019.
- [11] G. Fortino, W. Russo, and C. Savaglio, "Agent-oriented modeling and simulation of iot networks," in *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2016, pp. 1449–1452.
- [12] "What is hardware-in-the-loop?" [Online; accessed 3-February-2019]. [Online]. Available: <http://www.ni.com/white-paper/53958/en/>
- [13] M. Bacic, "On hardware-in-the-loop simulation," in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 3194–3198.

## Systematic Literature Review

- [14] S. Ferretti, G. D'Angelo, V. Ghini, and M. Marzolla, "The quest for scalability and accuracy: Multi-level simulation of the Internet of Things," in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, vol. 2017-Janua. IEEE, 10 2017, pp. 1–8. [Online]. Available: [https://www.engineeringvillage.com/share/document.url?mid=inspec\\_M18b6c86f160c2afb704M3d2d10178163176&database=inshttp://ieeexplore.ieee.org/document/8167672/](https://www.engineeringvillage.com/share/document.url?mid=inspec_M18b6c86f160c2afb704M3d2d10178163176&database=inshttp://ieeexplore.ieee.org/document/8167672/)
- [15] "Metric mishap caused loss of nasa orbiter," [Online; accessed 17-January-2019]. [Online]. Available: <http://edition.cnn.com/TECH/space/9909/30/mars.metric.02/>
- [16] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*, 1st ed. Addison-Wesley Professional, 2009.
- [17] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.
- [18] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.
- [19] T. Jung, N. Jazdi, and M. Weyrich, "A survey on dynamic simulation of automation systems and components in the Internet of Things," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 9 2017, pp. 1–4.
- [20] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of Things (IoT): Research, Simulators, and Testbeds," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1637–1647, 6 2018.
- [21] X. Li, Y. Wang, X. Zhou, D. Liang, and C. Du, "An implementation towards integrated simulation of cyber-physical systems," in *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCOM 2013*, 2013, pp. 789–796.
- [22] "Simulink for system modeling and simulation," [Online; accessed 14-January-2019]. [Online]. Available: <https://www.mathworks.com/solutions/system-design-simulation.html>
- [23] "Rational rhapsody," [Online; accessed 14-January-2019]. [Online]. Available: <https://www.ibm.com/us-en/marketplace/rational-rhapsody/>
- [24] "Working with the object execution framework (oxf)," [Online; accessed 14-January-2019]. [Online]. Available: [https://www.ibm.com/support/knowledgecenter/SSB2MU\\_8.2.0/com.ibm.rhp.frameworks.doc/topics/rhp\\_t\\_fw\\_working\\_object\\_execution\\_framework.html/](https://www.ibm.com/support/knowledgecenter/SSB2MU_8.2.0/com.ibm.rhp.frameworks.doc/topics/rhp_t_fw_working_object_execution_framework.html/)
- [25] A. Brokalakis, N. Tampouratzis, A. Nikitakis, I. Papaefstathiou, S. Andrianakis, D. Pau, E. Plebani, M. Paracchini, M. Marcon, I. Sourdis, P. R. Geethakumari, M. C. Palacios, M. A. Anton, and A. Szasz, "COSSIM: An open-source integrated solution to address the simulator gap for systems of systems," in *Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018*, 2018, pp. 115–120.
- [26] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoab, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: <http://doi.acm.org/10.1145/2024716.2024718>
- [27] Omnet++ discrete event simulator. Accessed 14-January-2019. [Online]. Available: <https://omnetpp.org/>
- [28] Hp labs: Mcpat. Accessed 14-January-2019. [Online]. Available: <http://www.hpl.hp.com/research/mcpat/>
- [29] A. Bounceur, L. Clavier, P. Combeau, O. Marc, R. Vauzelle, A. Masserann, J. Soler, R. Euler, T. Alwajeeh, V. Devendra, U. Noreen, E. Soret, and M. Lounis, "CupCarbon: A new platform for the design, simulation and 2D/3D visualization of radio propagation and interferences in IoT networks," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, vol. 2018-Janua. IEEE, 1 2018, pp. 1–4.
- [30] A. Bounceur, O. Marc, M. Lounis, J. Soler, L. Clavier, P. Combeau, R. Vauzelle, L. Lagadec, R. Euler, M. Bezoui, and P. Manzoni, "CupCarbon-Lab: An IoT emulator," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, vol. 2018-Janua. IEEE, 1 2018, pp. 1–2.
- [31] G. D'Angelo, S. Ferretti, and V. Ghini, "Modeling the Internet of Things : a simulation perspective," *arXiv*, 7 2017.
- [32] —, "Multi-level simulation of Internet of Things on smart territories," *Simulation Modelling Practice and Theory*, vol. 73, pp. 3–21, 11 2017.
- [33] —, "Distributed hybrid simulation of the internet of things and smart territories," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 9, p. e4370, 5 2018.
- [34] Advisor advanced vehicle simulator. Accessed 14-January-2019. [Online]. Available: <http://adv-vehicle-sim.sourceforge.net/>
- [35] S. N. Han, G. M. Lee, N. Crespi, N. Van Luong, K. Heo, M. Brut, and P. Gatellier, "DPWSim: A devices profile for web services (DPWS) Simulator," *IEEE Internet of Things Journal*, vol. 2, no. 3, pp. 221–229, 2015.
- [36] (2009) Devices profile for web services (dpws). Accessed 14-May-2019. [Online]. Available: <http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01>
- [37] T. Jung, P. Shah, and M. Weyrich, "Dynamic Co-Simulation of Internet-of-Things-Components using a Multi-Agent-System," *Procedia CIRP*, vol. 72, pp. 874–879, 2018.
- [38] Openmodelica. Accessed 15-January-2019. [Online]. Available: <https://www.openmodelica.org/>
- [39] D. Dhoutaut, B. Piranda, and J. Bourgeois, "Efficient Simulation of Distributed Sensing and Control Environments," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. Beijing: IEEE, 8 2013, pp. 452–459.
- [40] M. Bures, "Framework for Integration Testing of IoT Solutions," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 12 2017, pp. 1838–1839.
- [41] Junit 5. Accessed 15-January-2019. [Online]. Available: <https://junit.org/junit5/>
- [42] A. Chaves, R. Maia, C. Belchio, R. Araujo, and G. Gouveia, "KhronoSim: A Platform for Complex Systems Simulation and Testing," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 2018-Sept. IEEE, 9 2018, pp. 131–138.
- [43] V. Looga, Z. Ou, Y. Deng, and A. Yla-Jaaski, "MAMMOTH: A massive-scale emulation platform for Internet of Things," *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, vol. 3, pp. 1235–1239, 2013.
- [44] nsnam. Accessed 16-January-2019. [Online]. Available: [http://nsnam.sourceforge.net/wiki/index.php/Main\\_Page/](http://nsnam.sourceforge.net/wiki/index.php/Main_Page/)
- [45] G. Fortino, R. Gravina, W. Russo, and C. Savaglio, "Modeling and Simulating Internet-of-Things Systems: A Hybrid Agent-Oriented Approach," *Computing in Science & Engineering*, vol. 19, no. 5, pp. 68–76, 2017.
- [46] Inet framework. Accessed 16-January-2019. [Online]. Available: <https://inet.omnetpp.org/>
- [47] G. Fortino, A. Guerrieri, and W. Russo, "Agent-oriented smart objects development," in *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 5 2012, pp. 907–912.
- [48] M. Zouai, O. Kazar, B. Haba, and H. Saouli, "Smart house simulation based multi-agent system and internet of things," in *2017 International Conference on Mathematics and Information Technology (ICMIT)*, vol. 2018-Janua. IEEE, 12 2017, pp. 201–203.
- [49] P. Gimenez, B. Molina, C. E. Palau, and M. Esteve, "SWE Simulation and Testing for the IoT," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*. Manchester: IEEE, 10 2013, pp. 356–361.
- [50] "Why is the ogc involved in sensor webs?" [Online; accessed 16-January-2019]. [Online]. Available: <http://www.opengeospatial.org/domain/swe>
- [51] C. Manta-Caro and J. M. Fernandez-Luna, "A discrete-event simulator for the web of things from an information retrieval perspective," in *2014 IEEE Latin-America Conference on Communications, IEEE LATINCOM 2014*, V.-V. C.E., Ed. Institute of Electrical and Electronics Engineers Inc., 2014. [Online]. Available: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84946689633&doi=10.1109/%2FLATINCOM.2014.7041851&partnerID=40&md5=292f8ed92c6233b834548d33a497298e>
- [52] Sencscript (cupcarbon). Accessed 14-January-2019. [Online]. Available: <http://www.cupcarbon.com/sencscript.html>
- [53] "Iot device simulator — aws solutions," [Online; accessed 13-May-2019]. [Online]. Available: <https://aws.amazon.com/solutions/iot-device-simulator/>
- [54] "Iotify- develop full stack iot application with virtual device simulation," [Online; accessed 13-May-2019]. [Online]. Available: <https://iotify.io/>
- [55] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi, "CupCarbon: A Multi-Agent and Discrete Event Wireless Sensor Network Design and Simulation Tool," 2014.
- [56] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, and F. Zanichelli, "A Simulation Platform for Large-scale Internet of Things Scenarios

- in Urban Environments,” in *Proceedings of the First International Conference on IoT in Urban Space*, ser. URB-IOT '14. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 50–55. [Online]. Available: <http://dx.doi.org/10.4108/icst.urb-iot.2014.257268>
- [57] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-level sensor network simulation with cooja,” in ., 2006, p. 8.
- [58] ns-3 — a discrete-event network simulator for internet systems. Accessed 14-May-2019. [Online]. Available: <https://www.nsnam.org/>
- [59] M. Amoretti, M. Agosti, and F. Zanichelli, “Deus: A discrete event universal simulator,” in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 58:1–58:9. [Online]. Available: <https://doi.org/10.4108/ICST.SIMUTOOLS2009.5754>
- [60] brambilla/osmobility: Discrete event simulator for vehicular mobility. Accessed 14-May-2019. [Online]. Available: <https://github.com/brambilla/osmobility>
- [61] S. Karnouskos and M. M. J. Tariq, “An agent-based simulation of SOA-ready devices,” *Proceedings - UKSim 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, pp. 330–335, 2008.
- [62] M. Dyk, A. Najgebauer, and D. Pierzchala, “SenseSim: An Agent-Based and Discrete Event Simulator for Wireless Sensor Networks and the Internet of Things,” *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015. [Online]. Available: <https://www.engineeringvillage.com/share/document.url?mid=inspec{ }61e71935152cced8ad2M6fad10178163171{&}database=ins>
- [63] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, “iFogSim : A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things , Edge and Fog,” *arXiv*, pp. 1–22, jun 2016. [Online]. Available: <https://www.engineeringvillage.com/share/document.url?mid=inspec{ }77172d7715699c5b216M5b5a10178163171{&}database=ins>
- [64] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.995>
- [65] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki-a lightweight and flexible operating system for tiny networked sensors,” in *29th annual IEEE international conference on local computer networks*. IEEE, 2004, pp. 455–462.
- [66] A. Varga and R. Hornig, “AN OVERVIEW OF THE OMNeT++ SIMULATION ENVIRONMENT,” in *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*. ICST, 2008. [Online]. Available: <http://eudl.eu/doi/10.4108/ICST.SIMUTOOLS2008.3027>
- [67] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, “Network simulations with the ns-3 simulator,” *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.
- [68] G. F. Riley and T. R. Henderson, “The ns-3 network simulator,” in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.
- [69] Qualnet for network simulation — scalable networks. Accessed 17-May-2019. [Online]. Available: <https://www.scalable-networks.com/qualnet-network-simulation>
- [70] N. Beijer, “Continuous Delivery in IoT Environments,” in *International Conference on Agile Software Development*, A. Alliance, Ed. Agile Alliance, 2018, pp. 1–7.