FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Towards Live Development of IoT Systems

by

Nuno Guilherme Matos

December 21, 2018

CONFIDENTIAL

Mestrado em Engenharia de Software

Scientific Supervision by

Ademar Manuel Teixeira de Aguiar, PhD

Towards Live Development of IoT Systems

Nuno Guilherme Matos

Mestrado em Engenharia de Software

December 21, 2018

Abstract

Internet of Things is an upcoming area of Information Technology. As any upcoming area, it has a lot of uncertainty and it is bathed in complexity. Due to its potential, multitude of systems are already being implemented, although a substantially number of challenges are still in place waiting to be solved. Complexity is an inherent characteristic to these systems since they are integrated in heterogeneous environment, therefore depending on abstractions which lead to opaque systems ending on low comprehensibility. In the context of this thesis we try to investigate whether it is valuable to mix software disciplines such as Software Visualization and Live Programming, to achieve a solution that reduces the complexity of developing and maintaining Internet of Things systems.

Recurring to a visual game engine and live programming technologies and concepts we implemented one solution. We ended up finding that it is a new concept of programming where we can get instant feedback about the changes in a visual way, although, despite time, scope limitations and not being able to prove whether the solution, developed in the context of this research, did solve the research questions, we were able to prove, recurring to a survey, that the participants were optimistic about such solution, even though some of them did not have previous knowledge about some concepts.

Keywords: Internet of Things, Live Programming, Software Visualization, Complexity, Comprehensibility

Resumo

A Internet das Coisas é uma área, nas Tecnologias da Informação, que embora ainda em fase embrionária está em ascensão. Como qualquer área em ascensão contém uma grande grau de incerteza e é banhada em complexidade. Devido ao seu potencial, diversos sistemas já estão a ser implementados, no entanto, um número substancial de desafios ainda estão à espera de ser resolvidos. A complexidade é uma característica inerente a este tipo de sistemas já que estão integrados em ambientes heterogéneos, e, por isso, dependem de abstracções o que leva a sistemas opacos que acabam por terminar em baixa compreensão dos mesmos. No contecto desta dissertação tentamos investigar se era interessante combinar conceitos, como, Software Visualization e Live Programming, de modo a criar uma solução que reduzisse a complexidade em desenvolver e manter sistemas da Internet das Coisas.

Recorrendo a um motor de jogo visual e a tecnologias e conceitos de Live Programming, implementamos essa solução. Acabamos por descobrir que é um novo conceito de programação onde podemos obter feedback instantâneo sobre as alterações de forma visual, embora, apesar do tempo, limitações de âmbito e não sermos capazes de comprovar se a solução, desenvolvida no contexto desta pesquisa, resolve todas as questões levantadas na dissertação, pudemos comprovar, recorrendo a um questionário, com certo grau de certeza, que os participantes estavam optimistas quanto a este tipo de solução, mesmo que alguns destes não tivessem conhecimento prévio sobre alguns conceitos.

Keywords: Internet das Coisas, Programação em Tempo Real, Visualização de Software, Complexidade, Compreensibilidade

Acknowledgements

In first place, I want to express my deep gratitude to CERN to have allowed me to develop this project on site while working in the same area. I would also like to thank Manuel Martin Marquez, my supervisor at CERN, and Jose Carlos Luna Duran, my co-supervisor at CERN, for all the help they provided throughout my internship. Not less important, I want to express my thanks to the team that welcomed me at CERN for all the help on my integration and the work environment provided. That, for sure, helped me a lot doing this research.

With the same importance, related to FEUP, I would like to say a big thank you for the ones that accompanied my academic course, ranging from administrative people to professors, though, a special thank you to professor Ademar Aguiar, my supervisor at FEUP, for promptly accepting me as his researcher knowing that my internship at CERN would make things harder. For giving me all the software architecture basis, helping me be more critique and teaching me to pursue more and more, I would like to highlight Hugo Sereno, my co-supervisor at FEUP.

To my family, although specially to my mother, Anabela Matos, I want to ”shout” an ”infinite” thank you for, every time, providing me all the resources I needed to complete my objectives. To all my friends, that followed me during the course, I want to express my appreciation for all the moments that we lived together. Those are, after all, life experiences, and ones that I want to keep with me.

Special thanks to everyone, that one way or another, never let me give up and always led me to believe I was capable, specially to Raquel for being more than a friend... for being a source of support and encouragement.

Nuno Guilherme Matos

"Increasingly, people seem to misinterpret complexity as sophistication, which is baffling - the incomprehensible should cause suspicion rather than admiration."

Niklaus Wirth

Contents

List of Figures

Abbreviations

kPa	kilopascal
ºC	Degrees Celsius
ºF	Degrees Fahrenheit
3D	3 Dimensional
2D	2 Dimensional
API	Application Interface
AMQP	Advanced Message Queuing Protocol
CERN	Organisation Européenne pour la Recherche Nucléaire
CoAP	Constrained Application Protocol
ETL	Extract, Transform, Load
GDPR	General Data Protection Regulation
HTML	Hypertext Markup Language
IBM	International Business Machines
IETF	Internet Engineering Task Force
IDE	Integrated Development Environment
IoT	Internet of Things
IT	Information Technologies
JSON	JavaScript Object Notation
KA	Knowledge Area
LoRa	Long Range
MQTT	Message Queuing Telemetry Transport
MQTT-S	Message Queuing Telemetry Transport - Sensors
OMA	Open Mobile Alliance
PIC	Programmable Controller Interface
PoC	Proof of Concept
SDLC	Software Development Life Cycle
SCM	Software Configuration Management
SWEBOK	Software Engineering Book of Knowledge
UI	User Interface

Chapter 1

Introduction

In an era where Information Technology is in vogue, Internet of Things, a technical area of IT, is threatening to conquer the world (Figure 1.1) with its network of interconnected physical devices. The extensive diversity of devices existent, the scope and scale of the systems leads to highly complex and low understandable IoT Systems, either under the development or maintenance phase. This variety of devices drive the developers to build interoperable systems so they can exchange information between each one of them. Most of the time this is made recurring to abstractions resulting in opaque systems that one does not know how they work in a lower level, leading to low comprehensibility.

1.1 Context

The development of a system encompasses different activities organized in several phases, defined as Software Development Lifecycle (SDLC). It is not considered to be a methodology *per*

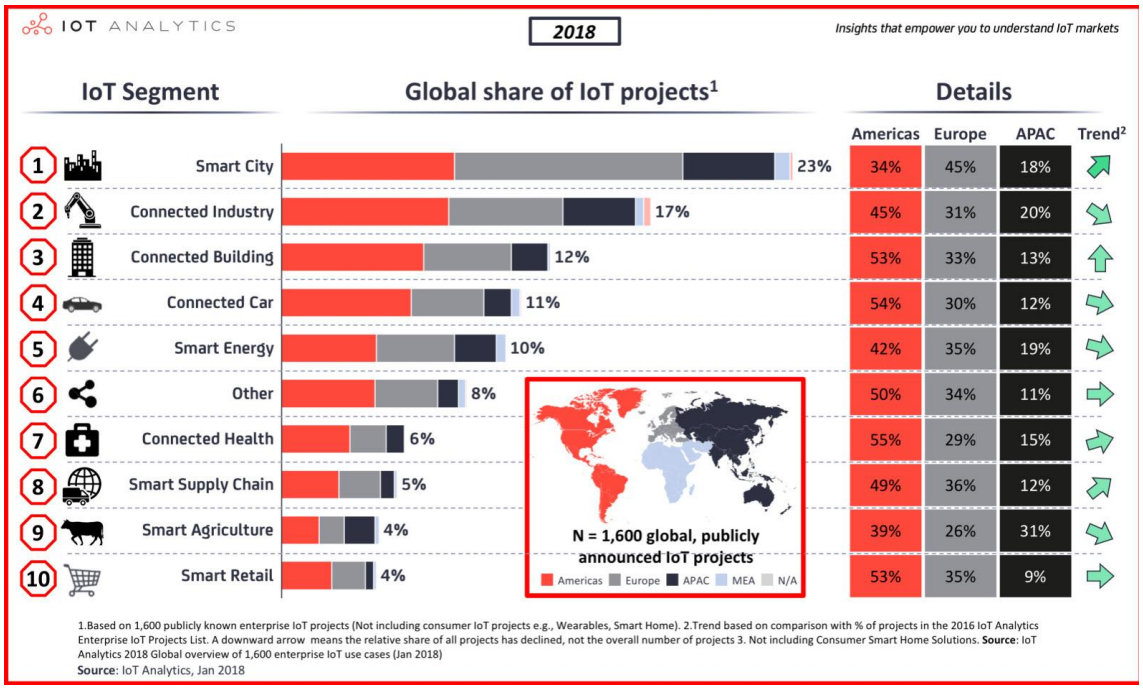

Figure 1.1: A representation of the IoT growth by segment. Adapted from [Col18]

se but rather a depiction of the phases of the lifecycle that a software goes through. Following SDLC phases in an IoT System is not smooth since some problems emerge, such as, Security Issues, Privacy Considerations, Interoperability and Standards, Legal and Regulatory Rights and Emerging Economy and Development issues [BPB12], scaling up when the devices needing to go through such revision are considered to be huge numbers in terms of diversification, resulting in a way more convoluted systems to be built.

Software follows a tendency to be incomplete and constantly evolving, as stated by the first of *Lehman's Laws of Software Evolution, Continuing Change* [Leh80], turning the development and maintenance of the system susceptible to continual modifications. Having perpetual changes in the structure will lead to its deterioration reflecting an augmentation of complexity, said *Lehman* in his second law of Software Evolution, *Increasing Complexity* [Leh80]. In addition, the time and effort one does need to absorb a change is really important and a factor to have in consideration, since it is different from person to person, and affects the system development directly. This attribute is also studied by *Lehman* affirming that a nonlinear relationship

between the magnitude of a system change and the intellectual effort and time required to absorb a change exists [Leh80]. However, the second law of Software Evolution also states that complexity increases unless something is shaped to antagonize it [Leh80].

With this work and the development of the respective solution, IoTCity we expect to reduce the complexity of understanding IoT systems and enhance the development, testing and management, given that the difficulty resides in how complex such systems are due to multiple challenge factors, such as interoperability, security, lack of standards and many others, however it has a lot of applications in multiple fields. Based on that, we try to infer if a platform that is capable of associate Software Visualization with Live Programming will suffice to fulfill the objectives as well as the expected results, described in the section below.

1.2 Motivation, Objectives and Expected Results

At the beginning, when the theme was proposed, I got instantly thrilled to see what was waiting for me in such a complex environment. Throughout my entire academic life, even though I chose software, I was always passionate about electronics however I did not have projects to work on such. Given this passion, I took the opportunity to merge together new concepts of software and, an area where I had no knowledge, electronics.

IoT systems can have multiple application in multiple fields. Some examples of such fields are: Industry, Smart agriculture and logistics, Intelligent Transportation, Smart medicine, Smart homes, and many others [Che+14; Kha+12; Zha+14]. Although the extensiveness of the fields and the large investment in these areas, there are still some challenges needing to be tackled so these systems can work in an efficient way. For instance, availability, reliability, scalability, interoperability, security and many others [AlF+15].

Keeping in mind that IoT is an interconnected network of multiple devices from different sources, one of the biggest challenges is interoperability. Not only but one of the challenges we believe that our solution IoTCity would be able to help is the described, by proving a playground to validate the design and communication between devices of the system. This might be due

to the fact that there are no common standards in this area, mainly because the manufacturers provide devices with proprietary technologies that may be not accessible. Although it would be very important to provide better interoperability [Kha+12].

Given the points above, the purpose of this work is to develop a solution that virtually map the reality of these systems in an immersive and live form, recurring to Software Visualization and Live programming techniques. This way we believe that it is possible to test the architecture of the system to better understand whether it would work in a real environment. Also, since we would have such playground, it would be possible to test changes to the system, organize the devices and trigger actions that would be executed in the real devices. In another terms, a virtually system that allows to develop and manage a real and deployed system. We also believe that such would decrease the complexity of understanding the system.

To sum up, we expect IoTCity, using concepts of Live Programming and Software Visualization, helps to reduce the burden on how IoT systems are developed and managed, to decrease the complexity of understanding the system and acts as a playground to test IoT architectures if that is the case. On top of that we also hope that it is passively to have more compatibility with a multitude of devices as well as being a tool that inspire new researches within the domain.

1.3 How to Read this Dissertation

In the following chapters we shall learn about the fundamentals of software engineering and how to apply them in a proper manner, as well as Internet of Things and their problems. The selection of topics is in no way complete but what we have found to be the most promising and relevant for this work.

In this Chapter 1 it was briefly described the context of the thesis. It gives not only an overview of what is the problem but also the goals trying to be achieved and the results. Beyond that, it is described some possible future work to be researched and implemented.

In Chapter 2 it is given the background information about Software Engineering, Internet of Things definitions and challenges, as well as an important concept of Software Habitability. In

Chapter 3 it is reviewed the State of the Art about the topics that are going to be useful throughout the research, such as Live Programming and Software Visualization.
In Chapter 4 is described the problem that the research tries to solve, specifying and narrowing the branches of it.
The Chapter 5 reveals the technology choices of the researcher and its reason and also explains the architecture of the developed solution.
In Chapter 6 one can find how the developed solution was tested in order to find out if it matches the defined criteria and solves or eases the research problem.
In the last chapter, Chapter 7, one can find the conclusion, main results and the future work planned for this research.

In the Appendix A the documents that served to help the validation of the thesis are presented.

For a comprehensive interpretation of this work, all chapters are suggested to be read in the order they are presented. Those familiar with the concepts of Software Engineering, Internet of Things and Software Habitability, who only want to have and impression of what was done in IoTCity, may skip the Chapter 2 and go directly to Chapter 3.

Chapter 2

Fundamentals

2.1 Software Engineering

Software Engineering is a computation area englobing specification, development and maintenance of software, applying techniques and practices. Its main areas are the following.

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Quality
- Software Configuration Management
- Software Engineering Management

All of these areas were thought to increase organisation, productivity and quality in the process of developing a system[SBF14].

Software Requirements

The Software Requirements knowledge area is concerned with the elicitation, analysis, specification and validation of software requirements as well as the management of requirements during the whole life cycle of the software product. They represent the needs and constraints placed on a software product that contribute to the solution of some real-world problem and without being applied correctly software projects are critically vulnerable [SBF14].

It is tightly related to Software Design, Software Testing, Software Maintenance, Software Configuration Management, Software Engineering Management, Software Engineering Process, Software Engineering Models and Methods, and Software Quality KAs [SBF14].

Software Design

Design is defined as both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of that process" [SBF14]. It is the software engineering life cycle activity in which software requirements are analyzed in order to describe how software is decomposed and organized into components and the interfaces between those components [SBF14].

During software design, software engineers produce various models that form a kind of blueprint of the solution and then analyze and evaluate these models to determine whether or not they will allow them to fulfill the various requirements. On top of that it is also possible to examine and evaluate alternative solutions and trade-offs. Finally, the resulting models allow to plan subsequent development activities, such as system verification and validation, in addition to using them as inputs and as the starting point of construction and testing [SBF14].

It consists of two activities that fit between software requirements analysis and software construction [SBF14]:

1. Software architectural design (sometimes called high-level design): develops top-level structure and organization of the software and identifies the various components [SBF14].

2. Software detailed design: specifies each component in sufficient detail to facilitate its construction [SBF14].

Software Construction

Refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging. Linked to all the other KAs, bust mostly linked to Software Design and Testing, since it involves significant design and testing, it uses the design output and provides an input to testing.

Throughout construction, software engineers both unit test and integration test their work. Typically produces the highest number of configuration items, thus it is tightly linked with Software Configuration Management. While quality is important in all KAs, code is the ultimate deliverable of a software project, thus it is linked with Software Quality. It is also related to project management, insofar as the management of construction can present considerable challenges [SBF14].

Software Testing

Software testing consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain [SBF14].

1. Dynamic - Means that testing always implies executing the program on selected inputs [SBF14].

2. Finite - Even in simple programs, so many test cases are theoretically possible that exhaustive testing could require months or years to execute. Determined by risk and prioritization criteria, testing is conducted on a subset of all possible tests [SBF14].

3. Selected - Many proposed test techniques differ essentially in how the test set is selected and different selection criteria may yield vastly different degrees of effectiveness. Identifying the most suitable selection criterion under given conditions is a complex problem; in practice, risk analysis techniques and software engineering expertise are applied [SBF14].

4. Expected - It must be possible, although not always easy, to decide whether the observed outcomes of program testing are acceptable or not [SBF14].

Software testing is, or should be, pervasive throughout the entire development and maintenance life cycle. It is also related with software construction, in particular, unit and integration testing are intimately related to software construction, if not part of it [SBF14].

Software Maintenance

Software maintenance is defined as the totality of activities required to provide cost-effective support to software. Software development efforts result in the delivery of a software product that satisfies user requirements. Once in operation, defects are uncovered, operating environments change, and new user requirements surface. Therefore, it must change or evolve [SBF14].

Software maintenance is an integral part of a software life cycle. However, it has not received the same degree of attention that the other phases have. This is now changing, as organizations strive to squeeze the most out of their software development investment by keeping software operating as long as possible [SBF14].

The Software Maintenance KA is related to all other aspects of software engineering [SBF14].

Software Quality

Software quality may refer: to desirable characteristics of software products, to the extent to which a particular software product possesses those characteristics, and to processes, tools, and techniques used to achieve those characteristics. Although, more recently, software quality is defined as the "capability of software product to satisfy stated and implied needs under specified conditions" and as "the degree to which a software product meets established requirements;

however, quality depends upon the degree to which those established requirements accurately represent stakeholder needs, wants, and expectations" [SBF14].

Software quality is achieved by conformance to all requirements regardless of what characteristic is specified or how requirements are grouped or named. Software quality is also considered in many of KAs in SWEBOK because it is a basic parameter of a software engineering effort. For all engineered products, the primary goal is delivering maximum stakeholder value, while balancing the constraints of development cost and schedule [SBF14].

Software Configuration Management

The configuration of a system is the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product. Configuration management is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle [SBF14].

Software configuration management (SCM) is a supporting-software life cycle process that benefits project management, development and maintenance activities, quality assurance activities, as well as the customers and users of the end product. Its activities are management and planning of the SCM process, software configuration identification, software configuration control, software configuration status accounting, software configuration auditing, and software release management and delivery, therefore it has relations to all the other KAs [SBF14].

Software Engineering Management

Software engineering management can be defined as the application of management activities planning, coordinating, measuring, monitoring, controlling, and reporting to ensure that software products and software engineering services are delivered efficiently, effectively, and to the benefit of stakeholders [SBF14].

Software engineering management activities occur at three levels: organizational and infrastructure management, project management, and management of the measurement program [SBF14].

2.1.1 Software Engineering Methodologies

Multiple software engineering methods exist. They provide an organized and systematic approach of developing software. Since the decision of using one method may have impact on the system being developed one has to choose carefully which methodology to use [SBF14]. The selected methods will be briefly described below.

Heuristic Methods

Fairly widely used in the software industry and experience based the Heuristic Methods breaks down into three main topics. *Structured Analysis and Design Methods* refers to the development of a high level view from a functional or behavioral point. *Data Modeling Methods*, as the name says, is the act of construct data tables and relationships that define, from a data viewpoint, the data models. *Object-Oriented Analysis and Design Methods* leads to the representation of a collection of objects that encapsulate the data and relationships previously made and define the interaction between objects over methods [SBF14].

Formal Methods

Applying rigorous mathematical based notation and language formal methods can be used to specify, develop and verify the software. Checking for consistency, completeness and correctness in an automated or semi-automated way can be achieved by the usage of a *specification language*. These languages, used during the software specification, requirements analysis and design stages, have the purpose of describing specific input/output behavior. *Program Refinement and Derivation* using a series of successive transformations is the mechanism that refines, or in other words, creates a higher detail specification. In order to verify if the models follow

the specified conditions a form of *Formal Verification* has to be used. One of these forms can be *Model Checking*. It is an analysis that verifies correct program behavior under all possible interleaving of event or message arrivals, and it should suffice to perform a Formal Verification. The last part on this type of methodology refers to the specification of pre and postconditions for each decisive block of code recurring to mathematical logic, meaning proof that defined conditions hold under all types of input [SBF14].

Prototyping Methods

Prototyping is considered to be an activity that most of the times generates incomplete or minimal software applications. Generally used to try out new features, getting feedback on user requirements or any other kind of assets and gaining useful insight into the software. At the level of *Prototyping styles* one can find multiple, though, an example of this would be paper products or throw-away chunks of code. The choice of the styles to follow is dependent on the type of project being developed. The *Prototyping Target* is the object that is being targeted to be prototype (e.g. algorithm). In order to define if the prototype is corresponding to the objectives it has to undergo an evaluation. This evaluation can occur in a number of ways. One of them is to test the prototype against the actual software or against software requirements (requirements prototype) [SBF14].

Agile Methods

Considered to be lightweight methods and being characterized by its short-term iterative development lifecycle, self-organizing teams, simple designs, and, among other things, with an emphasis on creating demonstrable working product, it was created to reduce the large overhead associated with the heavyweight of plan-based methods used in large scale software projects. A conglomerate of Agile methods is at disposition, although, for the sake of simplicity, the most common will be described. *Scrum*, considered to be the most used one, is more project management friendly than its competitors. A sprint, lasting no more than 30 days, is composed of tasks

which are previously identified, prioritized and estimated. At the end of each sprint there's a release of software. Choosing one of the methods also varies on the type of project being developed [SBF14].

2.1.2 Software Development Life Cycle

The Software Development Life Cycle describes the main phases and activities commonly used for developing and maintaining software [Ben12]. Depending on the type of software one is developing the SDLC can shapeshift into many styles. Some of these styles are described above though, as an example, one would have traditional approaches (e.g. waterfall process) vs agile approaches (e.g. Scrum methodology). These two are considered to be the most used by systems developers [Lea+12]. One would ask why such a systematic development process is usually followed. The reason behind is because it helps on having a clear vision of the project's scope and also on reducing the complexity of the entire development process. This leads to an increment on the success rate of a software application.

Figure 2.1: A representation of the SDLC. Adapted from [Sam12]

2.2 Internet of Things

This section intends to describe the core concepts, some of the history and also part of the challenges of the exponentially growing, Internet of Things.

2.2.1 Definition

The simplest Internet of Things (IoT) definition would be to explain that it is a set of devices, usually possessing some kind of intelligence, connected to the network sharing data between each other. While it may be true, it is much more than that. It is a global Internet-based architecture that eases the process of exchanging goods and services, at the cost of having impact on security and privacy [SS17]. This could also be seen as the new Internet extended with more interconnectivity, with a better perception of the information and more easily and intelligible services, such as education, health, security services, among others. One of the strongest advantages of the IoT is the allowance of the possibility of exploring new markets with different needs that may be solved recurring to this emerging technology.

2.2.2 Challenges

There are multiple challenges in this area, especially because it's an upcoming technology with various riddles to figure out and solve. Below, some of them are briefly described to give an overview of what one may find when confronted with IoT.

Reliability

Tightly coupled with availability, reliability leads to the rate of success of the service. Becoming even more demanding in case of critical systems, the development of all the system layers has to be resilient to failures. In order to not make wrong decisions, one has to build reliable information, distributed reliably. This becomes a challenge because this can only be achieved if all the layers throughout the entire IoT system can be trusted.

Performance

As any other type of service, IoT systems need to continuously be developed to meet user requirements. This means a big effort of testing multiple devices and many components that are most of the time under improvement and development. A challenge emerges when the performance versus price has to be accountable in order to have an affordable price for the customers.

Interoperability

Interoperability is one of the biggest challenges in IoT, due to heterogeneous devices, working on different platforms, that must communicate between each other. Since all sorts of devices come out everyday with new protocols one has to account on new communication protocols without losing functionality, and maintain integration with different technologies.

Security and Privacy

Concerns about security and privacy in such systems are critical. Essentially exchange of information is the core business of IoT. When done in heterogeneous networks and between millions of devices, it becomes even more difficult to address such concerns. With the new European General Data Protection Regulation (GDPR) new problems arise to define which type of data the devices should be allowed or denied collecting, and if they are able to handle such cases or not.

Management

As previously said, the multitude of heterogeneous devices and networks make the management of these smart systems a nightmare, in terms of Configuration, Security, Performance, and many others. Some efforts have been made, mainly by Open Mobile Alliance (OMA) and Internet Engineering Task Force (IETF) to come up with a communication protocol that allows the devices to be abstracted from the application level in order to achieve remote management capabilities.

2.3 Software Habitability

> *Habitability is one of the most important characteristics of software. It enables developers to live comfortably in and repair or modify code and design* [Gab96].

Gabriel, on [Gab96], merged the habitability concept of architecture with software. He defined the concept of software habitability as being the source code characteristic that allowed programmers, coders, bug-fixers, a people later come across the code, to understand its construction and intentions. Adding to those the comfortability and assurance on changing the code base. The purpose of this is to make the interaction with the code by humans not to forced, meaning that they genuinely think about it like they were at home.

Gabriel goes further and state that software needs to be habitable:

> *Software needs to be habitable because it always has to change. Software is subject to unpredictable events: Requirements change because the marketplace changes, competitors change, parts of the design are shown wrong by experience, people learn to use the software in ways not anticipated. Notice that frequently the unpredictable event is about people and society rather than about technical issues. Such unpredictable events lead to the needs of the parts which must be comfortably understood so they can be comfortably changed [Gab96].*

Summary

With this chapter we pretend to introduce some core concepts that intersect with what the research tries to tackle. We started by having a broader vision of software engineering narrowing it down to the methodologies, finalizing in the SDLC. It is possible, by then, to say that developing systems is not an easy task, let alone if it follows all the process described by SWEBOK. Not only such served as inspiration to this work, focusing on areas like, design, implementation and testing, but it also allowed us to have a background supporting the meaning of development in the title; it is described in Chapter 4.

Afterwards is presented an overview of what is and what challenges exist in IoT. We believe it was important to be added since in Chapter 3 we do a description on some of the technologies used in IoT. This allows us to better understand that this is a new upcoming area, that has multiple challenges, multiple applications and brings something useful to the development of certain fields. For this reason, multiple countries have been investing on it [Che+14; SS17; Kha+12].

Lastly a brief presentation on Software Habitability is given, since it is a concept that is used in Chapter 3 and is of importance to understand what is meant by that description.

Chapter 3

State of the Art

Searching for new ways of understanding and manage complex systems, many researchers tried to investigate different techniques to reach that objective. The techniques described throughout this chapter were considered to be the ones that had more relation with our work objectives. Two core areas for our work are Software Visualization and Live Programming.

Nonetheless, other technologies, mainly related to the IoT field, to better understand what is the path they're following, to extract key ideas that might be useful and in the direction of this work, as well as discovering similarities between tools.

3.1 Live Programming

In programming, working live need not be dangerous, and the opportunity it offers for immediate feedback can be very valuable. Live programming is taking place in more and more contexts, including web-server scripting, learning environments, and professional tools. This trend is likely to continue [Tan13].

We believe that this concept will help to minimize the latency between a programming action and seeing its effect on program execution as well as supporting learning, as described by Tanimoto, in [Tan13].

While live programming is likely to become ubiquitous, with its increasing incorporation into IDEs, scripting environments, and tools for learning, there are some qualitatively different possibilities, with much of the character of liveness, on the horizon for programming environments [Tan13].

3.1.1 Liveness Concept

Four separate phases were recognized as a traditional cycle of program development, them being: edit, compile, build and run. This did not allow a program to be subject of changes while running, except in some rare cases where debugging was engaged, which is where the liveness concepts appear, the ability to modify and reflect changes in a running software. With this ability Live Programming, that is one concrete implementation of the concept, emerged. Ideally it involves only one phase, instead of the four previously seen, constantly running while assorted events occur [Tan13]. Tanimoto divided liveness into six levels.

3.1.2 Levels of Liveness

As seen in *A Perspective on the Evolution of Live Programming*, by *Steven Tanimoto* [Tan13] the levels are:

1. Informative - The first and the most basic level of liveness. It does not provide semantic feedback at all. An example of this is simply a descriptive flowchart.

2. Informative and Significant - In this level, the developer would have to ask for feedback after doing changes in the program. Later on, the computer would respond based on the modifications applied. An example of this level would be an executable flowchart.
3. Informative, Significant and Responsive - Level 3 works like a push event system. A while after changes are made, the computer would provide a reaction to it. An instance to it is edit-triggered updates.
4. Informative, Significant, Responsive and Live - The difference between level 3 and level 4, is that, the latter, need not wait to trigger an update since it is always running and modifying the behavior as soon as changes are specified and made. This would be verified in a system that supports stream-driven updates.
5. Tactically Predictive - As an upgrade to level 4, this level also foresee the next action, conceivably with multiple branches, likely to occur when a change happens. Then, it executes one or multiple actions predicted, ideally in different sandboxes or virtual machines. An example of this would be a system that after presenting such predictions allows the developer to choose one of them and implement it without human intervention.
6. Strategically Predictive - Refers to an utopian level. Adds the possibility to infer gross functionality hinged on goals or aspirations of the programmer's thought. Able to favorably perform strategic predictions instead of simple tactical predictions (Level 5). The system would then be able to combine a program, with such background, acting accordingly, from the consolidation of the current program and a large knowledge base.

3.1.3 Smalltalk

An example of a language that supports live programming is Smalltalk. Smalltalk allows developers to dynamically change the program at runtime [Cal+13].It supports hot-swapping, meaning the code is updated without restarting [McD07]. To achieve such an IDE is shipped using features to create, remove, alter, methods and classes, while the system is running [Cal+13]. Even though it is not considered to be a "true" live programming language [McD07], it emulates

the concept since its kernel (e.g. classes, methods) and development tools (e.g. compiler, code browser) extensively use dynamic features to implement live behavior [Cal+13].

3.1.4 Light Table

An open source integrated development environment that arises from the obsession of just working with static text [Gra12]. An idea just flowered. The idea of creating an IDE providing instant feedback and debugging, based on the premises of needing more than editor and project explorer. The need for moving things around, keeping clutter at down levels and bringing information into foreground in the correct places was very present [Gra12]. So, Light Table surges. One of the founders, Chris Granger, has his vision as following:

> *Light Table is a new environment for creating software. The main problem of building software is way harder than it should be and it's really inefficient. A good analogy is like, being a developer today is being a painter with a blindfold. You make a stroke on the canvas, you don't see the result. The time to be able to see a change when coding could be anywhere from 30 seconds to 8 hours. That disconnect is hugely impactful, it means what we end up doing most of the time, instead of doing little things and trying them, we make huge changes and hope to God that we got it right. -in [Lyn12]*

There are more features that Light Table provides [Tab], though, in the context of this thesis, the liveness features that it provides was considered to be the most important concept.

3.1.5 Summary

The tools presented that followed the concept of live programming and support it on its core, focus on giving immediate feedback to the user. A more thorough search reveals more tools that implement the same concept, such as Lisp and SuperGlue [McD07]. Although, given the time constraints, none of it was target of experimentation. One of the goals of the research is

to explore this concept and try to define whether it goes towards the same direction of the main goal.

3.2 Software Visualisation

As a result of becoming an important work instrument capable of excelling the comprehension of a complex phenomenon, Visualisation became a field of study in Computer Science, which, in fact, is rich in metaphors suitable to memorize concepts and exploit analogies, leading to an improvement of perception in structures and functions [Die07]. Gershon [Ros+94] described visualisation, in 1994, as following:

> *Visualization is more than a method of computing. Visualization is the process of transforming information into a visual form, enabling users to observe the information. The resulting visual display enables the scientist or engineer to perceive visually features which are hidden in the data but nevertheless are needed for data exploration and analysis.*

The goal of this area of investigation is to produce the best computer images so mental images are evoked in one's mind so to embellish software understanding, rather than generating the best good looking ones [Die07]. Using the same approach as Diehl [Die07], in between the two major areas of visualisation, *Scientific Visualisation* and *Information Visualisation*, in this thesis context, the latter will be the only considered, since it refers abstract data instead of physical data. Such considered, the following definition on *Information Visualisation*, arises, by Gershon et al. [GEC98]:

> *Visualization provides an interface between two powerful information processing systems — the human mind and the modern computer. Visualization is the process of transforming data, information, and knowledge into visual form making use of humans' natural visual capabilities. With effective visual interfaces we can interact*

> *with large volumes of data rapidly and effectively to discover hidden characteristics, patterns, and trends.*

Comparing both definitions their intent is almost the same, although the latter is oriented towards a specific area of visualisation (Information Visualisation), causing it to be written in an extra simplistic way and further objectively.
Investigators in this area are researching towards the augmentation of comprehensibility on the software systems recurring to the development and study of computer graphical representations of software aspects, concerning the structure, behavior and evolution of the software [Die07].

- Structure - Static parts and relations that can be computer or inferred without executing the program.
- Behavior - Refers to the execution of the program with real and abstract data. Can be seen as a sequence of program states, containing both the current code and the data.
- Evolution - Emphasizes code is changed over time so to add functionality or fix bugs.

The end objective of *Software Visualisation* is to improve understandability on software systems as well as the productivity in the development process [Die07]. In the following subsections, different techniques of Software Visualisation will be explored.

3.2.1 The City Metaphor

Wettel et al. [WLR11] state that civil architecture and software engineering have several similarities. Based on that premise, they conceptualized a city metaphor visualisation approach using the urban domain as the core analogy. In this visualisation technique, the system is depicted as the whole city, the packages as districts and the classes as buildings [WL07]. In addition, Wettel and Lanza decided to consider 3D so it was possible to use the third dimension, once not taken into consideration since it used to be dismissed claiming its small benefit, as an additional mean to encode quantitative values, and also because they claimed it could greatly improve software habitability [WLR11].

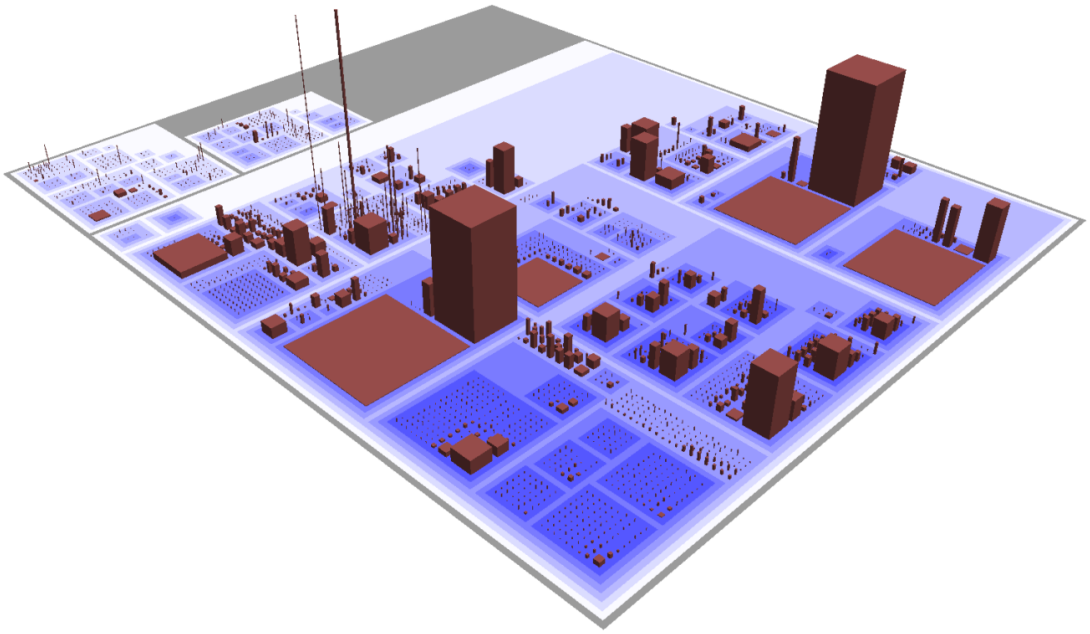

Figure 3.1: A representation of ArgoUML in CodeCity. Adapted from [WL07]

From that research, Wettel et al. produced CodeCity. It concluded that, in terms of program comprehension and design quality, using the city metaphor was efficient and versatile as a visualisation tool. To achieve its validation, an empirical evaluation was run spanned across over six months. After that they concluded that the tool provided some benefits in terms of software visualisation and that people were actually enthusiastic about the tool, despite the industry being the main drawback of adherence.

Yet another applicant of the concept, Vizz3D, a reusable framework [LP05], act as an information visualization system, promoting system structure and quality information in a comprehensible way, easing the perception of the system [Pan05]. Re-using the same concept of mapping software entities to relatable city objects, City Metaphor [Pan05], in order to enhance comprehensibility while decreasing the effort of the humans involved in maintenance tasks [LP05].

The height of the buildings represent the amount of code within classes/interfaces, the flame texture indicates a low/lack level of documentation, and the boxes are mapped as classes while the cylinders as interfaces [Pan05]. In the next figure (3.3) it is possible to see how the classes relate to each other by graphically represent the same color to each member function of

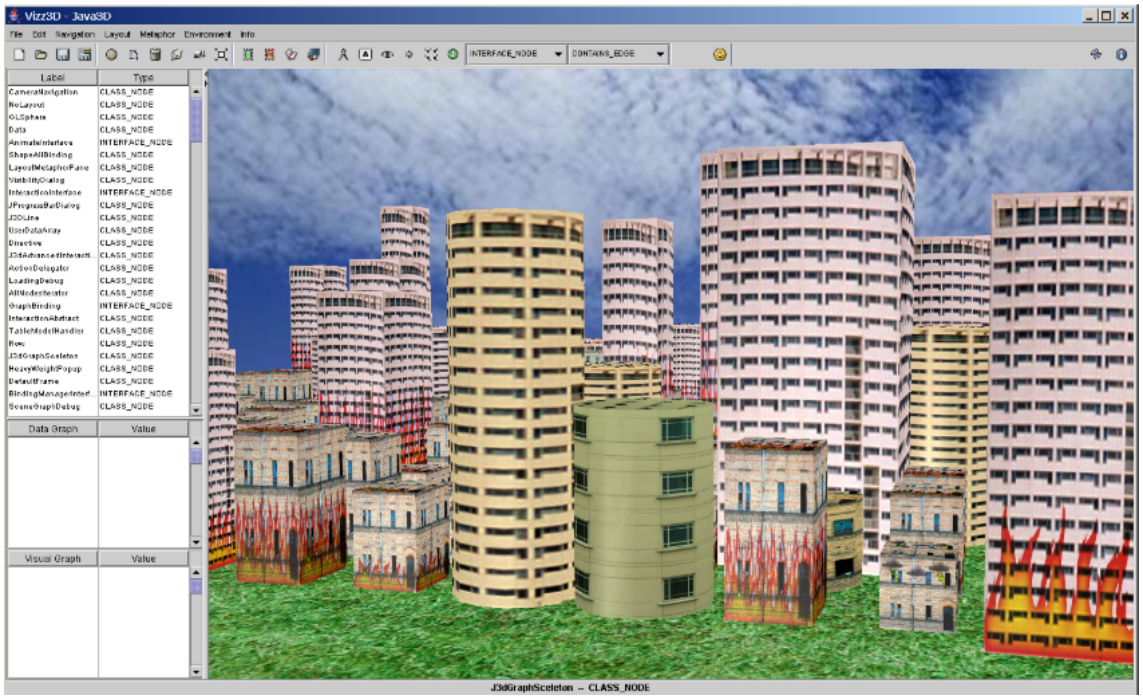

Figure 3.2: A representation of class hierarchy using City Metaphor concept. Adapted from [Pan05]

a determined class [Pan+07].

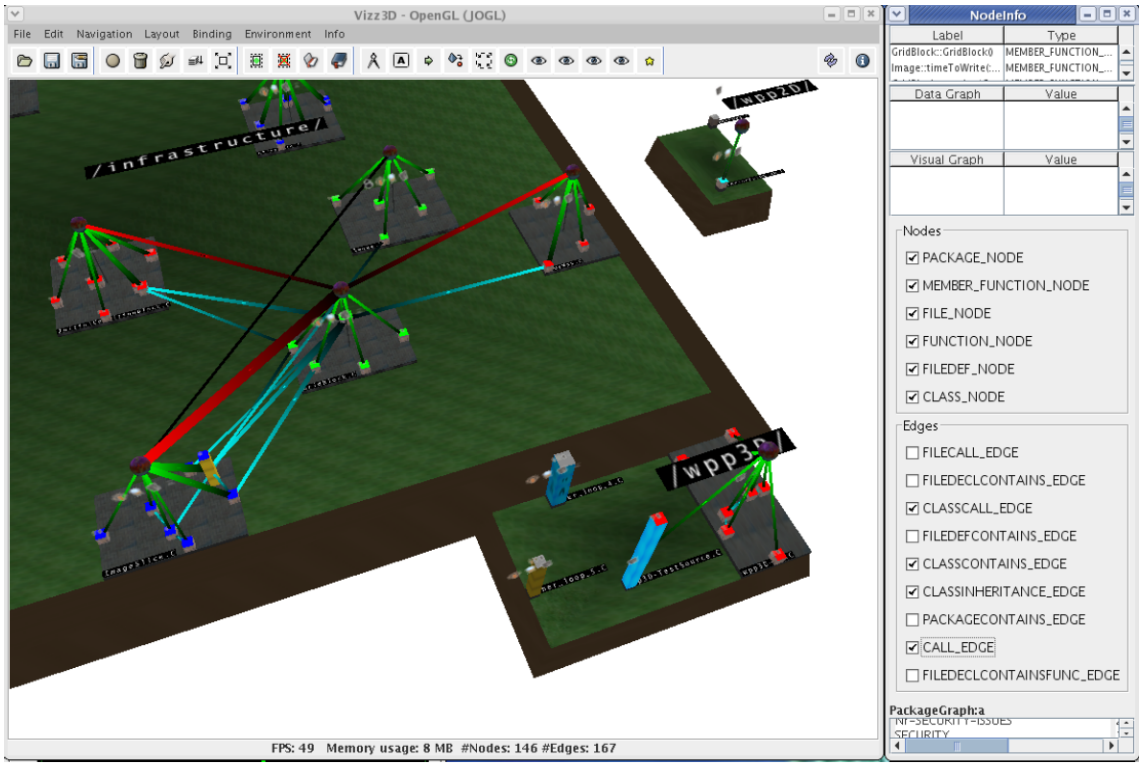

Figure 3.3: A representation of class membership using City Metaphor concept. Adapted from [Pan+07]

3.2.2 Spirals

Spirals technique can be used to provide time based information in a compact and an efficient manner [Rei01]. This method can be used, among other things, to better depict the behavior of software entities (e.g. Attribute allocation quantity, Compiler execution behavior) [Cru+16].

A concrete application using this type of technique is Bloom. In this case, Reiss and Renieris [RR02] decided to provide a set of visualisation controls. The first set of controls is to define what the spiral will be presenting and how to handle the data, while the second set is related to the objects being analyzed and how they're going to behave when drawn in the spiral. To do that, Reiss and Renieris [RR02] used properties such as hue, saturation, height and width, and on top of that also added texture behavior.

Making usage of such spirals would allow us to visualize our system metrics in real time, since they're time based, as well as providing the possibility of visualizing the system and respective metrics in a different timeline. Although passive to other, above are described two examples of two useful cases in our own solution. Figure 3.4 contains information relative to the variables being analyzed throughout the time of execution is shown. We can see that throughout the execution time the memory stack graphic changes the height, width, saturation and hue based on properties defined in the second set of visualization controls provided by BLOOM.

3.2.3 3D Hierarchical Edge Bundle

This technique intends to represent the hierarchical structure of software entities and their relationship. Based on graphs and cities context the entities are shown as city components while connections are characterized as edges connecting the urban objects [Cru+16]. Splitting this method into two main layouts, *Nested* (Figure 3.5) and *Street* (Figure 3.6), it is possible to have different type of control and comprehensibility [CZB11]. In the nested layout each element is graphically represented according to its size and all the components are drawn on top of their parent, while in the street layout packages are represented as streets, sub-packages placed perpendicularly to their parent and classes drawn as buildings surrounding the "streets" (packages)

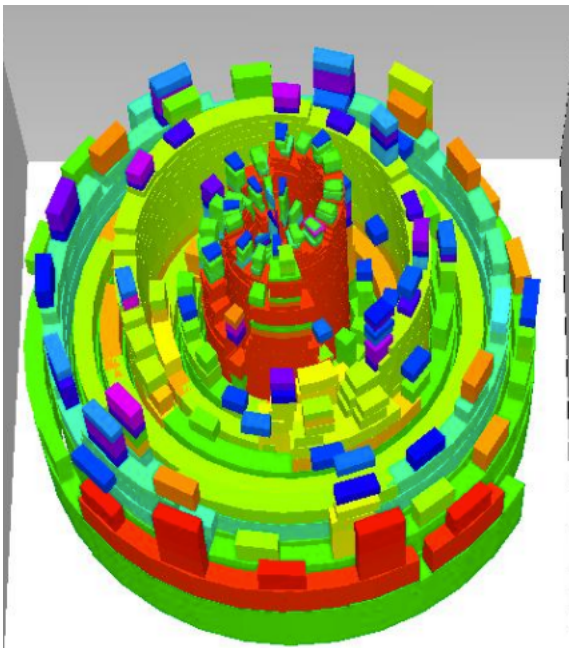

Figure 3.4: Bloom spiral view of the memory stack during execution. Adapted from [RR02]

[CZB11]

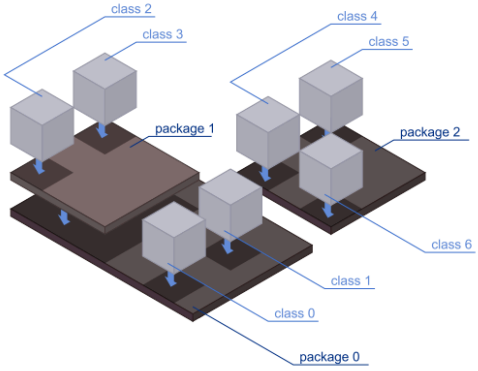

Figure 3.5: Nested software city layout. Adapted from [CZB11]

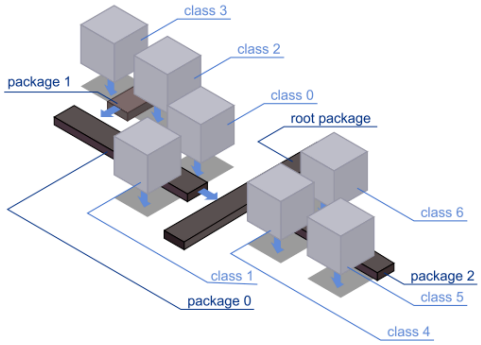

Figure 3.6: Street software city layout. Adapted from [CZB11]

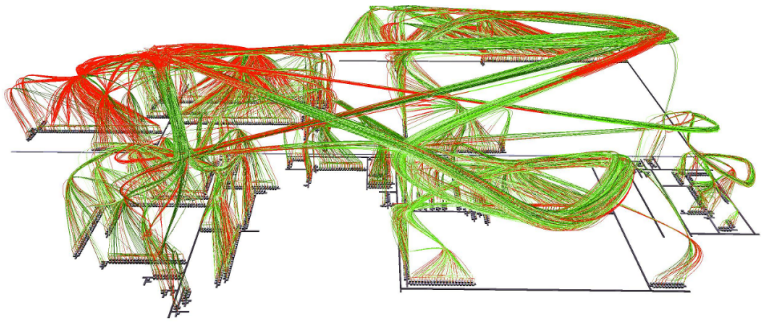

Figure 3.7: Component relations on top of street layout. Adapted from [CZB11]

3.2.4 Summary

The previously described techniques of software visualisation were considered to be the most important in the context of the thesis, however does not neglect that other techniques exist and should be used in the correct context. Some examples of non-reviewed, in depth, techniques would be *Animation System*, *Box Tree* and *Point Maps* [Cru+16].

3.3 Graphical Engines

Building a realistic virtual environment is a tough task. It is complex, expensive and time consuming. Although toolkits are available, many only provide a subset of tools needed to create a complete virtual world. Fortunately, one has the alternative to use current generation game engines. They afford realistic virtual worlds with user friendly interaction and the simulation of the real world. Given the fact that this new generation game engines provide a multitude of features, such as, 3D rendering, 2D drawing, physics, dynamics, tools to edit the environment, among others, one could say that they are suitable for prototyping virtual environments, since they allow the developers to customize experimental requirements in a short amount of time [TS08].

There are several game engines available, as of this moment. Some of them newer, some of them older. A quick search demonstrates the most recent ones. Some of them are Unity3D, CryEngine, Unreal Engine, Godot Engine, among others. A more thorough inspection reveals dozens of game engines, nevertheless, given the scope and time to release this study, one had to

confine himself to the most popular ones, that met the requirements of the application. Also, it is not in the scope of this thesis to do a benchmark comparison in order to show which one will be considered the best. Yet, in the implementation section, technical decisions will be thoroughly explained.

3.4 Internet of Things

Multiple IoT scenarios require integration with online services and real time sensing and actuation. One simple case of this might be industrial automation. Although possible to achieve such use-cases using traditional programming tools, it can become very difficult, leading to the apprenticeship of new protocols and APIs, creation of new data processing components and the linkage between them [BL14]. In the following points it will be discussed some of the most recent technologies and techniques one has at disposition to develop an IoT system. It will not be described all of the technologies and techniques but the ones that the researcher thought could make his research easier and focus on the statement he wants to measure.

3.4.1 Cloud Overview

As of today, the solutions for cloud IoT management tools are diversified. After trying some of them, such as Node-RED, Google IoT, ThingsBoard, they tend to provide the same functionality of setting up the devices, minimal configuration and the possibility of having dashboard with data provided by the devices [BL14; Thi; Goo]. Given the time provided only this applications were tried out, however there are more in the market such as AWS IoT, IBM IoT, Salesforce IoT Cloud, SAP Hana, Azure IoT.
A small description of the technologies tested will be given in the following subsections.

Node-RED

To provide more flexibility while keeping the easiness of usage, Node-RED provides a data flow programming paradigm where computer programs are modeled as directed graphs connecting

networks of black box nodes that exchange data along connected arcs. This paradigm lies at the heart of visual programming languages [BL14]. Definitely an application of the previously described, Node-RED is a dashboard-like application that runs locally, on a device or on the cloud. It is a flow-graph based web application that allows the users to wire hardware devices, APIs, and services together with drag and drop mode [Nod], as passively seen in Figure 3.8. Implemented using Javascript using Node.js framework, taking advantage of Node's built in event model, it allows an easier management of the system [BL14].

In Node-RED, once a flow is created or after a change, the user needs to deploy the flow, saving it to the server and restarting its execution [BL14]. This flow suggests that it would be considered a software with liveness level 2. It is also supported by IBM and users' community contributing to a repository of new nodes and flows [BL14; Lib].

As stated by Blackstock and Lea [BL14], new nodes can be simply added to the application just by adding the JavaScript implementation plus the HTML file. This was an inspiration for our research to create an architecture that was capable of accepting new integrations with easiness. The fact that is supports the deployment of nodes templates just by dragging and drop was also an inspiration to us and why we decided to implement such, as well as allowing users to plug components into sensors.

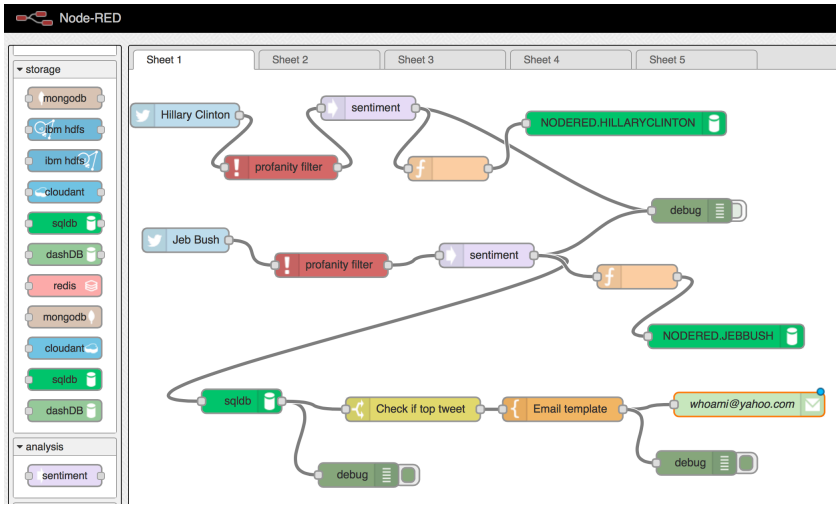

Figure 3.8: Node-RED drag and drop dashboard. Adapted from [IBM15]

Google IoT

Referring to Google Cloud Services, Google IoT is yet another service provided by Google that pretend to ease the hassle of managing multiple devices and its configuration. It does not have a drag and drop interface, but it allows to integrate with other services provided by the cloud platform. Thus meaning that one could have, for example, the data coming from the sensors being stored in a database and dashboard of metrics.

ThingsBoard

ThingsBoard is also a dashboard-like application that allows a minimal configuration and management of IoT devices (Figure 3.10). Aside from other possibilities, they also allow the users to have a real time dashboard with the data coming from the configured devices (Figure 3.9).

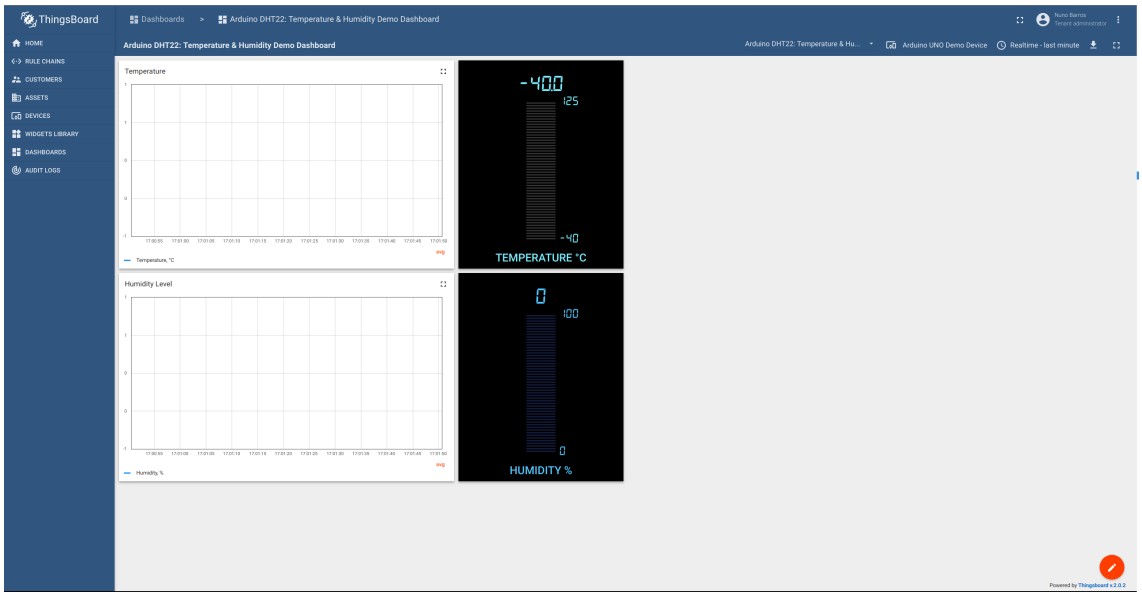

Figure 3.9: Dashboard of real time data from ThingsBoard

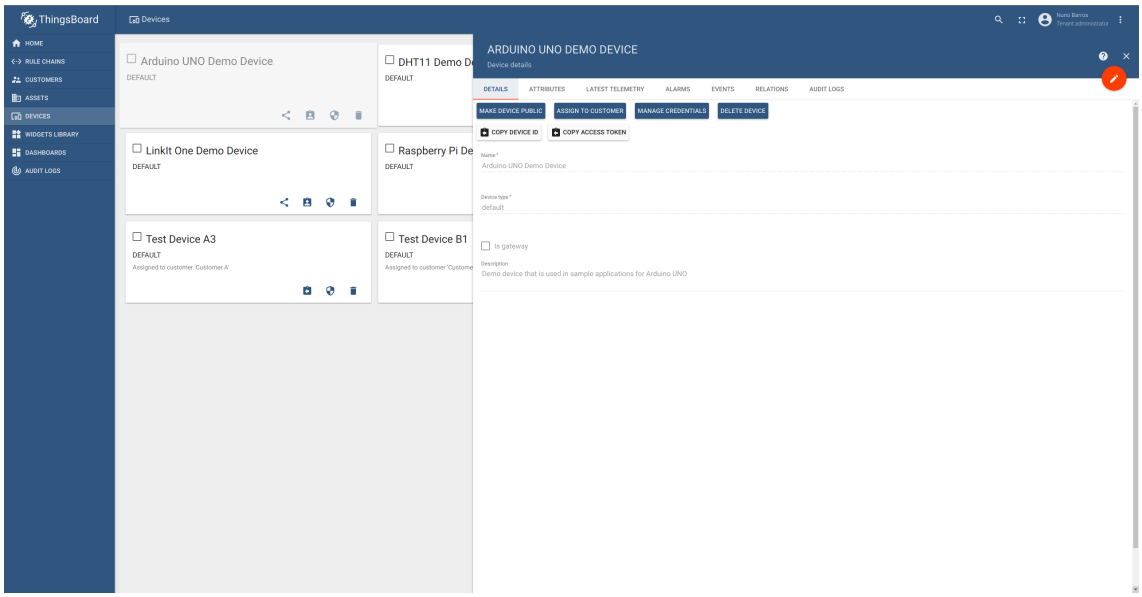

Figure 3.10: Dashboard of device configuration from ThingsBoard

3.4.1.1 Summary

As one can easily depict from the descriptions above, these services try to ease the difficulty of manage and develop an IoT system. Whether they use different methods and ways of doing such, the end objective is always similar. All of them lack the possibility of locating the physical sensor, which is one of the things the solution developed in conjunction with this work tries to solve.

3.4.2 Programmable Devices

There are thousands of different sensors in the industry with distinct functions. Starting from temperature sensors, flow meters, pressure transmitters and so on. Although costly, wired networks are usually preferred to connect sensors to the base station, leading to interconnection between devices [ZJL15]. As well as sensors there are multiple type of programmable devices. Such devices can act as a base platform to the sensors in order to connect them internally, to the Internet and also to hold logic (e.g. running an application that reads the data from sensors and trigger actions based on the data read). At this moment, a multitude of programmable devices exist in the market, however since the time provided for this thesis is not enough to test all the

devices and to develop the application throughout all the levels, a technical decision, to virtualize sensors, was made so that the research could be more focused on principles it is trying to solve rather than branching of to other topics that might have led to other segments, not being addressed by this research. In an example of programmable devices, the ones that came up in the first place, when discussing which ones to start off with, were Raspberry Pi and Arduino (shown below). There are other types of programmable devices, such as *Programmable Interface Controller* (*PIC*), *AdaFruit* and *LoRa*, however since the time provided for this research does not suffice to support multiple systems a technical decision, to virtualize sensors, was made so that the research could be focused on the principles it is trying to solve rather than branching of to other segments.

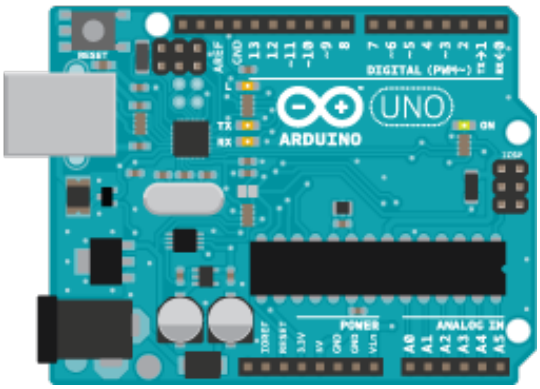

Figure 3.11: Arduino

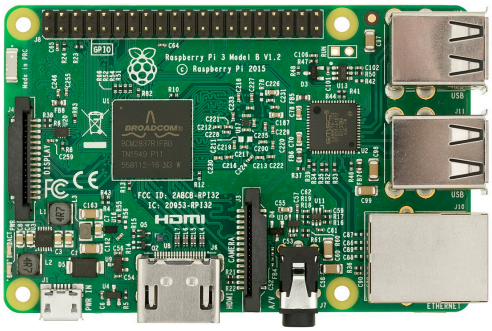

Figure 3.12: Raspberry Pi

To start this thesis, a research on how the sensors and programmable devices work was done, using two Raspberry Pi 3 Model B with Sense Hat. One of the most interesting findings but also one of the biggest drawbacks is the wiring management that had to be done in order to try some non-wireless sensors, and also the fact that the same sensor from different vendors might have different firmwares, which led to inconsistencies in the code base. Since this was only with a small number of sensors one could think that this problem grows as the scale also grows.

The decision of virtualizing sensors has its pros and cons. On one hand, we do not need to care about the wiring management, that in our experience is not easy at all to perform and since we are not mapping actions to a real system it allows us to emulate a playground. However, on the other hand, it increases the overhead of having to add logic to these virtual sensors to

emulate the real one, and it removes the possibility to test it in a real environment, given that performed actions are not mapped. Although we believe that with the strong development that is being made, as we've seen before, it will be possible to map the real sensors into virtual ones with easiness. This would mean that it might turn into a good decision.

As we have seen, we had some drawbacks experimenting with different sensors. But, this is not so bad in the end. It allowed us to experience that interoperability is a real problem in IoT systems, especially when they become big enough. Also, it opened our eyes that the management of these systems is really complex and that it might be solved by the usage of virtualization techniques, abstracting the lower levels of implementation.

3.4.3 Containerization

Containers were designed to provide an isolated work environment when running an application. They share some libraries in lower levels, but the container itself, hosting the application, only contains the configured libraries, binaries, and may also have other necessary middleware [Doc16]. Narrowing down the applications that provide such architecture one can easily find one of the most sounding applications, Docker. Docker is an application that allows the users to run isolated containers on top of it. It shares the same hardware, operating system and the container engine between them, but the containers themselves have different specific binaries and libraries specified at the instantiation of the container. A thorough search reveals more applications that also provide the same goals, as *rkt*, *OpenVZ*, *Windows Containers*, and others, though, this research is not specifically focused on comparing technologies but the only purpose is to help keep the focus on what really matters. Thus meaning that Docker was chosen in order to help reduce work that is not intended to be done by the purpose of this research. Next chapter, one can find how the solution was implemented, it provides an explanation why Docker was used in the context of the application developed during the time of this work and what it tried to solve. Figure 3.13 demonstrates the architecture one can find in a container application.

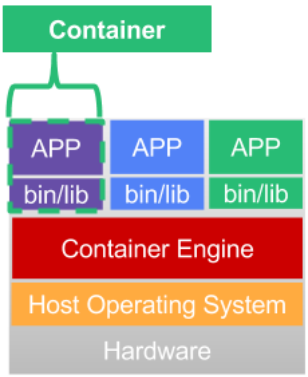

Figure 3.13: Container architectural point of view. Adapted from [Tec16]

3.4.4 Messaging

With the emerging of the Internet of Things and all its devices and sensors yet a new challenge appeared. Transmitting data to multiple applications in multiple different networks. Although this could be possible, it is a lot of charge in the low-cost small devices, having limited capabilities in terms of battery, storage and processing (Figure 3.14). However, if one inverts the flow, the problem could be easier to solve. That is where, one of the various solutions, the architecture of Publish/Subscribe (pub/sub) messaging systems arise (Figure 3.15), from the area of distributed computing.

The concept of pub/sub is to invert the flow of data. This means that instead of being the sender that decides where the information goes, it is the receiver registering interest of consuming such data. The act of registering interest in the information is called subscription. This way it is possible to move the processing to a more powerful component in the network. Therefore, the sender is recognized as the publisher while the receiver is identified as the consumer. But there is still a missing part, which leads to a question of "which component routes all the messages to the different subscribers?". To answer this question, the concept of broker appears. The broker coordinates the subscriptions, producers and consumers to ensure that data flows from the publishers to the subscribers. This type of systems are widely used mainly because they are highly scalable and support high dynamic topology [HTS].

Going into deeper levels of pub/sub systems one can find the multitude of protocols that

follow the pub/sub pattern. Such example of data patterns could be MQTT, MQTT-S, AMQP, CoAP, and many others [HTS; KJ09; Som07].

There are multiple applications that implement the architecture described above, such as Mosquitto, RabbitMQ, CloudAMQP, and others.

Not neglecting the fact that other messaging systems, such as queues, protocols, in different layers, such as infrastructure and identification, and more applications might exist, one has to be aware of the research problem, thus meaning that a description of all the layers and all the applications are not going to be provided. Yet, the reason for the technical choice is provided in the section of the implementation.

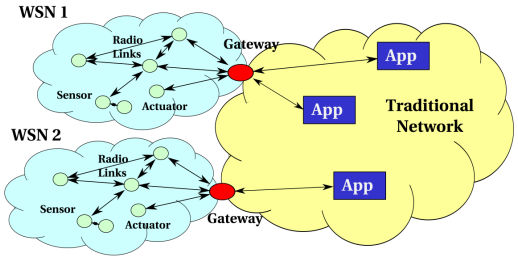

Figure 3.14: Sensors network without pub/sub system. Adapted from [HTS]

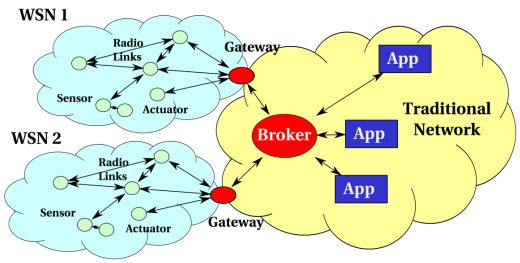

Figure 3.15: Sensors network with pub/sub system. Adapted from [HTS]

To better explain the figures above one could describe a network of sensors in the Figure 3.14 that is doing all the processing of routing the messages to the specific applications. As said above, it is very heavy on the sensors to do such functions. On the other hand, on the Figure 3.15, one could depict that the heavy work is now being done at the broker level, releasing the sensors to act only as message producers. This allows multiple applications to subscribe to multiple topics, as long as they connect to the broker.

3.4.5 Summary

As described above a multitude of technologies exist in the market and try to ease the hassle of developing and maintaining an IoT system. Most of them focus on having an interface allowing the user to perform actions on devices or applications. We've seen a broader example of what

one has at disposition, as of today, at the same time that are slightly described some technologies that might help on the research focus.

3.5 Conclusions

A variety of concepts exists in Information Technology (IT). Specifically, the area of IoT is a complex area, mainly because it is an heterogeneous environment. There are multiple ways to try to solve the complexity of developing and managing IoT systems, but, focusing on the research, it tries to suppress some of the complexity in IoT management by applying some of the concepts described above. The concepts on this chapter will be used in the development of the research's solution and in the section of the implementation a description of how they were used is given.

Chapter 4

Research Problem

The usage of IoT Systems can bring multiple benefits to a variety of sectors, them being, as example: (i) transportation industry, (ii) medical industry, (iii) marketing industry, and many others. However, besides bringing such opportunities, these systems also bring a lot of challenges to the development and maintenance of their infrastructure. These type of infrastructures, having considerable growth, will not be able to avoid increment at complexity level, especially when it demands complex, highly heterogeneous and distributed systems.

4.1 Challenges of developing and managing an IoT System

As all the new upcoming technologies the IoT also suffers a lot of skepticism. With its growth, in terms of devices and applications, a lot more challenges, not mattering if they are already known

or not, are to come. There are multiple areas where IoT systems are trying to be studied, some of them are the health, industry, automation and home/appliances sector.

Agnostic from which area the system is going to work the problem of Naming and Identity Management is one of big concern. Providing new services, billions and billions of devices, with different purposes, will be connected over the Internet. Nevertheless, these billions of devices must be identified, which leads to the confrontation of having an efficient naming and identity management system with enough resources to dynamically assign an unique identifier to a large number of devices [Kha+12].

Specifically talking about the home sector, where this research tries to focus, one is confronted with multiple dispute. In terms of interoperability, one should consider this as one of the most difficult threats to solve. Not only because the end-users must have an easy way of connecting and using the devices, but also because multiple devices from multiple vendors with multiple versions have to be able to work together, on different network interfaces, without demanding much effort on the consumer [Sam16]. Many manufacturers provide devices with proprietary technologies that may not be accessible by others. To provide better interoperability between all devices the standardization of IoT is important [Kha+12].

Thinking about smart homes, and all the data generated from the devices, what is the breakpoint that one is reaching when the information about the user actions is enough to detect patterns. With IoT the information of the consumers' daily routine is being collected and is used to provide better services [Zha+14]. How much of it is colliding with privacy? Essentially because it deals with user private data [Che+14]. In this case, it is important to preserve the information, that can identify the customer, private, if one wants the service to be successful. Since the devices carry its own identification it is also necessary to properly secure and prevent unauthorized access to this information [Kha+12].

Compared against normal networks, security and privacy in IoT is more outstanding [Che+14]. It falls into two different main categories. The data collection policy and data anonymization. The first one refers to the policy that enforces which type and amount of data the devices collect and which access they have to it, while the latter tries to ensure that the data collected and its

relations are anonymous [Zha+14]. As this data travels over the Internet, it is also demanding that the devices carry proper data encryption to guarantee data integrity and confidentiality [Kha+12].

Still on consumers' defense, another problem emerges. How the devices are going to manage themselves. It is known that the devices endure "intelligence" and can perform health checks to self-manage, though it is also demanded that they can recover without human intervention, so they can adapt to failures and environments changes. Because some sensor network applications operate without infrastructure, one loses the ability to do maintenance and repair over the infrastructure, which leads to another challenge [Sam16].

As we've seen before, stated by Lehman's Law [Leh80], changes are constantly happening in the software world. Thus meaning that the maintenance of these systems, desired to be outside of physical reach and human intervention, is also something that has to be thought about. Changes will always happen, either because the environment changed or a node failed. Hence, a quickly and cost-effective maintenance scheme has to be designed, which poses another challenge, since, as we've seen before, it is an heterogeneous world [Sam16].

All these challenges are due to the fact that the Internet of Things lays on an heterogeneous environment. Firstly one has legacy heterogeneous network architectures and applications, secondly has a multitude of communications protocols that should be low-cost and reliable and lastly the variety of applications the behave differently because they answer different requirements [Che+14].

To sum up this section, albeit IoT creates a newer information society and knowledge economy [Che+14], it also creates new challenges much more difficult to deal with, caused by its heterogeneity and complexity [Zha+14]. Above are described some of the challenges the researcher found to be more relevant and could be simplified by the solution (IoTCity).

In Chapter 5, a description of how some of these challenges are addressed by this research is made, in order to better understand how it tries to reduce the complexity of developing and managing and IoT system.

4.2 Thesis Statement

The purpose of this section is to provide an explanation of the statement on this dissertation.

> Is the use of the *IoTCity* easing the process of developing and maintaining Internet of Things systems when compared to the existing tools?

Present in the initial thesis statement are terms whose meaning is not straightforwardly concrete, therefore leading to question deserving further development:

- What is the meaning of *developing and maintaining systems*?
- Which are the *existing tools*?
- *Who* does benefit from the improvements?
- What does it mean *easier*?

4.2.1 What is the meaning of developing and maintaining systems

As stated previously the concept of Software Development Life Cycle is considered to be a process of develop and maintaining software [Ben12]. In the context of this work, the developing part is more related with the phase of design and implementation, described in SDLC. This is given by the fact that the solution tries to ease the way a developer is experimenting the design by allowing them to work with virtual sensors. Achieved by emulating a kind of playground where one can check whether a certain design will work, as well as the subsequent implementation. On the other hand, the maintenance field is related to the testing and evolution phase of the SDLC. As the name states maintaining a system sometimes requires testing new ways of doing something and in this case a playground is an important feature for this since it won't break, supposedly, the service. This also makes sense to the evolution part, where one has to test new features without breaking the system in order to evolve the system. In this case, having a way to do it without messing with the current service is a plus. With the above reasons we consider the development of IoT system to touch most of the parts of the SDLC.

4.2.2 Which are the existing tools

The existing tools refers to those concerning any state-of-the-art practice that provide some degree of development and maintenance of IoT systems. Tools mentioned in Chapter 4 are an example of what tries to ease complex at this level, offering visual support.

4.2.3 Who does benefit from the improvements

The live development, visualization and management approach to IoT systems aims to help IoT architects, developers, administrators. As of clients, it is also applicable but using another set of features and reducing the scope so it is user friendly at the same time as it is a complete solution, depending on the point of view.

4.2.4 What does it mean easier

This term is used in the sense of being "more efficient". This means that the solution provided gives a broader vision of the system, allowing the user to do certain actions in less time, for instance, less difficulty and with lower risk of breaking the current service.

4.3 Research Questions

Q1 : Does the use of IoTCity help to ease the complexity of developing and maintaining an IoT architecture, compared to non-visual, live, configuration tools?

Q2 : Does the use of IoTCity help to ease the complexity of developing and maintaining an IoT architecture, compared to visual, non-live, configuration tools?

4.4 Summary

As we've seen before the IoT world is full of applications and challenges. Not only that but it is also affected by the constant change in software. There is a need, therefore investment, to

enhance these systems and make them more understandable and easier to develop and maintain.

Based on the technologies seen in Chapter 3, this works focus on trying to develop a solution that is capable of reducing the complexity of understanding IoT systems, helps to reduce the burden on how these systems are developed and managed, acting as a playground to test IoT architectures, providing aswell more compatibility for devices. We also expect this work to be able to trigger new investigations within the domain.

To sum up, why not to investigate if a solution that combines Software Visualization, thought to increase understandability, and Live Progamming, thought to reduce complexity by adding instant feedback, is able to tackle some of the challenges described before.

Chapter 5

Architecture and Implementation

In this chapter we present the architecture and respective implementation of the developed tool supporting our approach, as well as some technical decisions made.

5.1 Overview

By definition, an IoT system is something that is not easily mapped into physical entities except the devices themselves. Despite the devices being physical they are not straightforwardly represented into a virtual world, since they come in multiple forms. Instead they can be transposed

to models that the users already know and can easily figure out what they mean, and by this, we mean, for instance, that a temperature sensor is mapped to a thermometer and a pressure sensor mapped to a pressure gauge. Thus allowing the users to gradually becoming more familiar with the virtual world, due to the similarities between the real world versus virtual world. In order to do such, and because a consensus is not well defined in the community, a 2D environment was used. In this case, as the tool is more focused on the development of the system, thus mainly used by systems developers, we considered to be more fruitful using a 2D environment instead of a 3D environment, since it would be harder to depict and move throughout the system. Adding to the fact that, with these requirements, the third axis would not bring benefits, if so, would only increase the learning curve.

Figure 5.1: A representation of IoT City.

IoTCity is a tool that embodies this type of metaphors. Conceptually it allows the development and management of IoT systems through of what we consider an intuitive mapping between city metaphors and IoT resources. For the sake of this work we chose to start providing the management for smart homes, in the view of a system's developer and manager. The main difference from other approaches is that this tool provides a live infrastructure with the

real time state of each resource, as well as a broader vision of the system, recurring to zoom in/out functionality.

5.2 Technology Choices

Unity

The main technology chosen to develop this PoC was Unity, mainly because it is an 2D/3D engine with extensive documentation, has a lot of community support, thus allowing us to accelerate the process of development. It also opens up opportunities for future work in terms of Virtual/Augmented reality.

Kafka

On the moment of the development of this dissertation I was undergoing an internship at CERN. In my section, we were developing internal Kafka as a Service. We, therefore, were very comfortable with the technology. In order to accelerate the process of development and because we are not testing the technologies we use themselves, we decided to use Kafka as a message queue. This allowed us to implement an event system that was capable of acting also as a message buffer. Not only that but because Kafka is built to work with many other layers on top of it, such as Kafka Connect and Kafka Streams. Such, would allow us to provide an out of the box environment of ETL tools, for example.

One other reason that was taken into consideration to use Kafka was its role in big data. Since IoT systems will be able to generate gigantic quantities of information, we'll need, in the future, to have some tool that allows us to easily do transformations on such data, either for data cleaning, data transformation or any other type of technique with data. Kafka, in our opinion is future proof to do that, given that mature tools for such events are built around it.

Docker

Since Docker allows us to deploy self contained images in a Unix based environment, which is what RaspberryPi is running, we decided to use it because it allowed us to host small applications to control and communicate with the sensors remotely. Another reason is that at CERN we were actively using this technology as a helper for the deployment of Kafka as a Service, therefore giving us experience and comfort on Docker, thus accelerating the process of the development.

RaspberryPi

As an introductory device the RaspberryPi is very simple to use. It runs an Unix based distribution, which leads to easier development on the system itself, since it gives a higher level of abstraction. Plus, the fact that the provided device already came with some sensors, allowed us to speed up the development of the solution, as much as we did not have to code the sensors themselves.

5.3 Architecture

As the tools should be able to adapt to the heterogeneous environment that IoT is, we tried to make it as generic as possible. Although one has to think that assumptions and decisions have to be made, so to reduce the scope of the solution, and it is not possible to develop the perfect solution, let alone doing it in the first version. The Figure 5.2 describes, in an high-level representation, the system's architecture.

Taking into account that this PoC only answers to a RaspberryPi the implemented architecture tries to be as generic as possible to take into account other devices that might be useful, opening the possibility of new implementations in the future. As we can see in Figure 5.2, the tools is composed of 5 main packages:

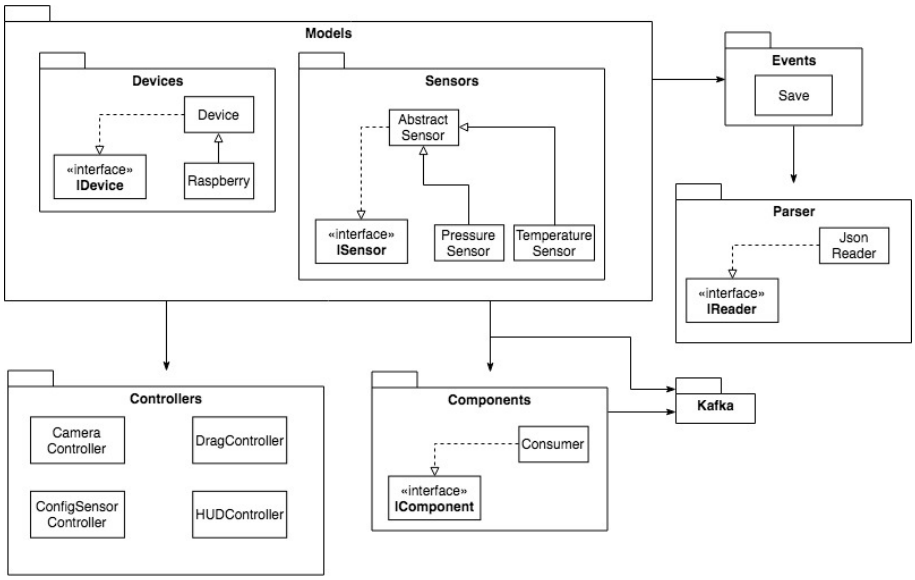

Figure 5.2: A package diagram describing IoTCity architecture.

- Models - Provides the abstraction to develop multiple virtual sensors and devices. For our proof of concept, having time restricted constraints and as we've said before, we are only able to use the Raspberry abstraction, for devices, and pressure and temperature sensors' abstraction. It is also possible to see that this package contains two more packages, Devices and Sensors. We implemented it the way it is described since it allows an easy construction of new sensors and devices.

- Events - This is the package that handles the flow of events that are triggered at the application level. In our case, it is a simple saving action that is triggered when a configuration for a sensor is applied. This way it allows us to store the model and information about the sensors so the application can restart with the same state.

- Parser - As a sensors' configuration is saved this package saves the configuration, in our case, in a JSON file. We implemented it through an interface so it would easily allow the parser to be changed.

- Controllers. Essentially related to Unity behavior this packaged controls the zoom in/-zoom out, drag n' drop functionality, as well as the UI controllers for sensors and devices.

- Components - Not only for this reason but as we wanted to use the zoom in/out functionality to decrease/increase, respectively, the abstraction of the system, it was essential to allow the users to plug in components to the sensors. One other reason was to facilitate a way for the users to see which messages were flowing through the selected sensor in a live way. For that, we allowed the devices to act both producer and consumer. Thus, the flow would be as following: the sensor produces a message and sends it to Kafka. If the consumer component is triggered the same sensor, virtual in this case, will read the messages sent directly from Kafka.

  The reason we did not directly read from code, since it is virtual, is because we tried to get as real world as possible, which means the sensor won't be working in local mode. Implementing it like this would allow the user to read the messages from a remote source.

The proof of concept was implemented using a multipurpose three-dimensional engine, Unity, although just using the two-dimensional capabilities. As previously said, the decision of using it derived from being an engine with extensive documentation and community support, leading to a faster implementation, as well as opening the possibilities of exploring new features related with Virtual or Augmented reality, for instance.

5.4 Resource Mapping

As said before, some of the resources that one can find in IoT were mapped to an easily recognizable resource in the virtual environment.

Smart Home

For the sake of simplicity, and because this research focused on smart homes, a background with an house plant was used, so the developers could better understand which house system they are dealing with. As of this moment, is not possible to design or inject the real house plant, since

it was going out of the scope of this research, though, it might be an improvement for the next versions.

Temperature Sensor

A temperature sensor, as the name indicates, is a sensor that measures the physical temperature of the place it is positioned. Then the temperature value, usually expressed in degrees Celsius (°C) or Fahrenheit (°F), is communicated to the messaging pipeline. In our system, these sensors, may have three states. In the following points a description of those states is provided:

- Green - Configuration is Ok. Sensor is ON and emitting data.
- Yellow - Configuration is not Ok.
- Red - Sensor is OFF, therefore not emitting data.

Pressure Sensor

A pressure sensor, as the name indicates, is a sensor that measures the physical pressure of the place it is positioned. Then the pressure value, usually expressed in kilopascal (kPa), is communicated to the messaging pipeline. In our system, these sensors, may have three states. In the following points a description of those states is provided:

- Green - Configuration is Ok. Sensor is ON and emitting data.
- Yellow - Configuration is not Ok.
- Red - Sensor is OFF, therefore not emitting data.

Raspberry Pi

Acting as a base station, as of this moment, the Raspberry Pi, which we choose to use because it was already provided with the SenseHat, allowing us to reduce the time to develop the solution, is used to connect the sensors to the messaging pipelines. Since it acts as a computer, it contains

the full stack needed to put it to work without too much effort. The system is designed to support more devices, but, today, this is the only one supported.

Connecting resources

In order to visualize the connections between devices a simple line is used. This type of lines serves the purpose of showing that there is a connection between two resources. Multiple colors can be seen, ranging between green, yellow and red, seen in Figure 5.3, Figure 5.4 and Figure 5.5, respectively.

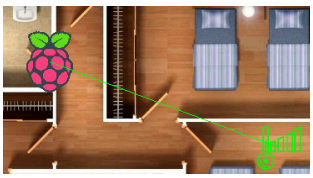

Figure 5.3: Representation of a green connection between two resources.

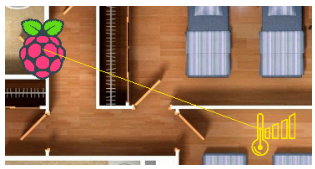

Figure 5.4: Representation of a yellow connection between two resources.

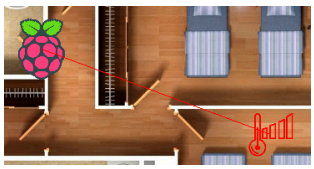

Figure 5.5: Representation of a red connection between two resources.

5.5 Resource Interaction

Resources contain a set of attributes and actions, depending on their type. In order to reveal such a context panel was developed. The panels appear when a specific component is clicked, as shown by Figure 5.6. The actions are as following:

- ON/OFF - allows the user to turn on or off the resource. This can be seen in Figure 5.6.
- CREATE - allows the user to create a resource. This action is only available when a point in the house is right clicked, as shown by Figure 5.8.
- ALTER - allows the user to edit the resources' configuration. Figure 5.6.
- DELETE - allows the user to delete a resource. Figure 5.6.
- CONNS - allows the user to enable the connections for that resource. This action is only available in sensor devices. Figure 5.6.

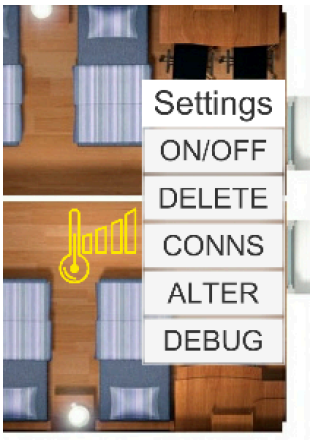

Figure 5.6: Representation of a component (sensor) click.

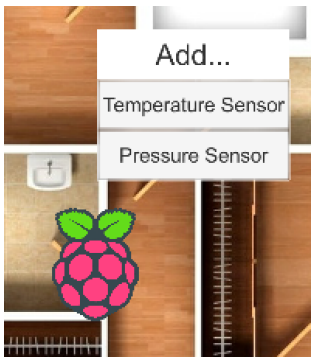

Figure 5.7: Representation of a component (device) click.

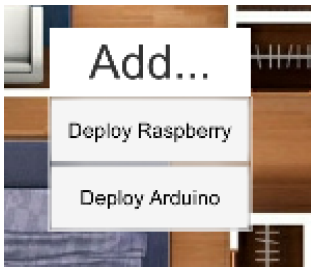

Figure 5.8: Representation of a component (house) click.

- DEBUG - allows the user to plug a component to check the messages being sent. This action is only available in sensor devices. Figure 5.6.

A specific action that only the sensors possess is the drag n' drop feature. This allows the user to place the sensor on the desired place on the house, facilitating the user to see where the values it is reading come from.

5.6 Infrastructure Updates

As the title suggests, Live, this solution only made sense if, in some way, it followed the principle of liveness. In order to that, we thought that the best technique that could manage these requirements would be to implement a publish/subscribe system. This is the reason why all the actions that are executed on a resource go through a messaging system, allowing us to recognize certain pattern of messages and trigger actions based on such. This allowed us to have a system that is as live as latency allows.

In our solution we have components that follow liveness level 3 and others that follow level 2. One clear case where we could have implemented level 3 was the component that saves the configuration for a determined device. This component only triggers feedback when it is saved, triggered by pressing a combination of keys (CTRL+Return). For this, we consider it level 2 but the level 3 would be achieved by implementing a listener to the input text area and trigger the

parser for every keystroke. Although, what is triggered by the save configuration action is level 3, since, given that the configuration is correct, it will immediately change the status, therefore its connection, to green and start pumping messages to Kafka, providing a live feedback of the change.

One of the components that support liveness level 3 is the toggling on/off sensors. This is implemented recurring to Kafka. What is happening behind the scenes is the sensor generating virtual messages and pushing it to Kafka when toggled on, but, at the same time, listening to specific configuration topic. The toggle on/off is basically a message sent to this configuration topic, with a specific regex ([sensor-ID:action]) that the virtual sensor will receive, understand and trigger the action of toggling off/on. Once again we had direct access to the code and could do it without going through Kafka, but, in this way, it allows us to closely emulate what would be a remote sensor. Given this we can have an edit-trigger update as instantaneous as latency allows.

Another action that we consider it being liveness level 2 because it is not triggered by the zoom in/out functionality is the functionality to debug the messages that are being sent by such sensor. We consider it level 2 because the user has to click the debug button to actually trigger the functionality whereas if it was done through the zoom in/out, it would still be user input, but he was not requesting for it directly.

The idea, and why we looked into Docker, was because this virtualization could be achieved recurring to docker images, instead of being coded at the application level, and then the sensor would be mapped to this docker image, which would mean that, in an embryonic phase, each image would correspond to one sensor. This allowed us to connect two different systems using the same standard for communication, messaging through Kafka. After some experimentation, we managed to put it working, although it was slow when we tried to deploy more instances. We did not dig into lower levels why such was happening, though, as we were using a Raspberry Pi, and it is quite limited in terms of hardware, we decided to do this virtualization at Unity level. We can only assume that either using docker for this case is not a good case, or that Raspberry Pi is not a good gateway in somewhat big IoT systems.

To sum up, we learned that it is not so easy to provide a full liveness level 3 system, within the timespan we had. Also we found out that is hard to define what belongs to level 2 or level 3. We do not consider our solution to be complete, let alone covering all the use cases. We expect it to be an introductory work for this domain and trigger new ideas to developed.

5.7 Infrastructure Scaling

One left open question in this section is whether the management of massive infrastructures, composed of extensive amount of resources, is successful or not. Even though this is, as of this moment, not possible to achieve, it is possible to emulate. For the purpose of testing the visualization of a considerable size system, we simulated and composed environments of 10 houses, each with 10 resources, as represented by Figure 5.9. Considering that houses are not placed within the same region and because the system should be able to support multiple areas, one concept that could be introduced is to group it by regions, as Figure 5.10 suggests. Though, considering this work as exploratory, this question was not addressed in the initial phase, although is something to induce for future work.

This scaling was done by increasing the number of components of the system. It tried to demonstrate that IoT systems, when we increase the number of components working together just by a little number can become very complex. Mainly, it attempts to depict that by using a layout technique, one of the ideas to use in future work, it might be able to increase understandability.

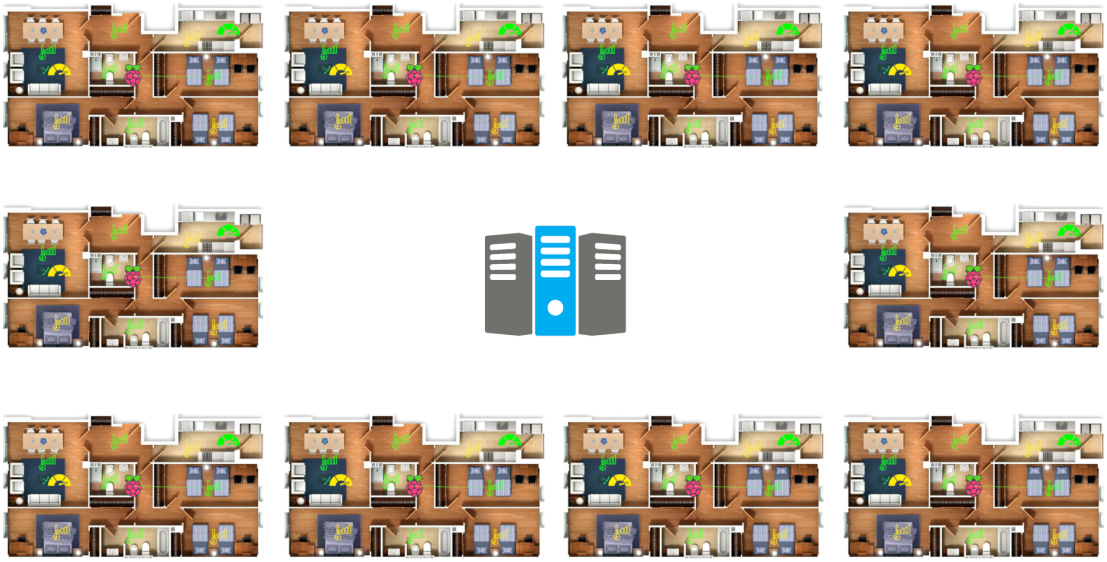

Figure 5.9: Representation of a scaled IoT architecture without recurring to layout techniques

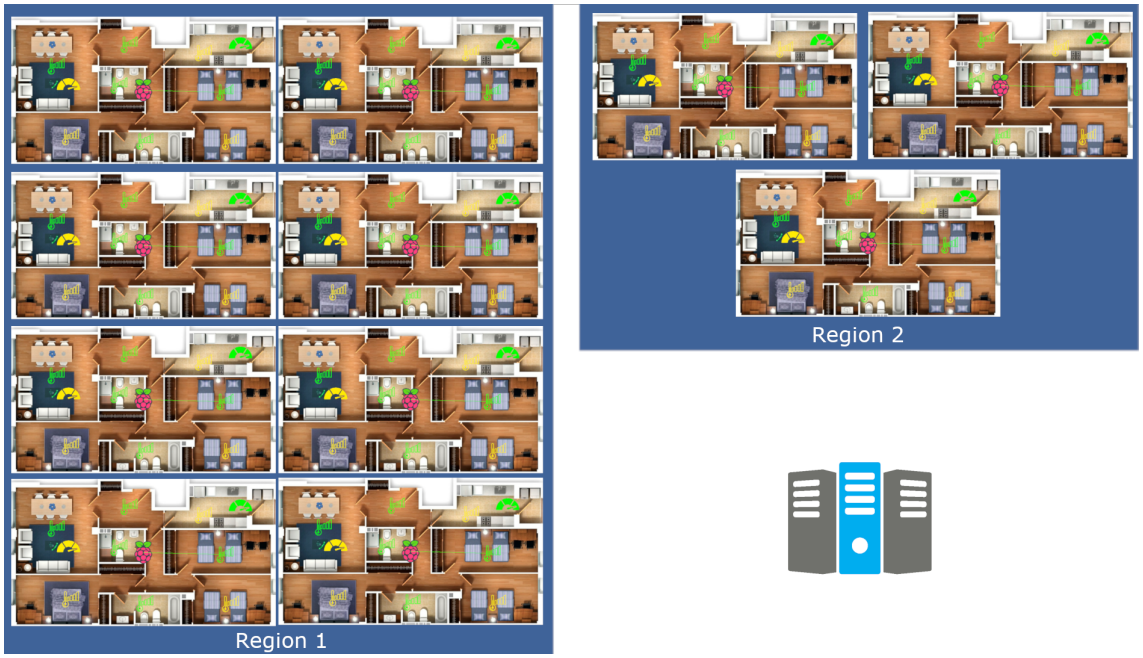

Figure 5.10: Representation of a scaled IoT architecture recurring to layout technique

Chapter 6

Validation

The tool proposed by this research was somehow evaluated with the help of a survey that was developed to gather opinion from the potential users. The survey was not blocked to anyone in specific although most of the answers are given from people within IT sector.

We believe, as this is a new emerging area with many unknown things, that to have a proper validation (very concrete metrics and conclusions), would require a lot of effort, resources and time and it may be out of the scope of this thesis. As these were scarce we opted to make a survey with closed and some open answers, that will be present in the Annex section in order to try to understand if something similar to what the solution provides is going in the direction of what people would use if they had contact with something like that.

6.1 Empirical Evaluation

Wish List

In the following points the researcher provides some wishes that may have led to the development of a better solution.

1. Having a proper validation strategy. There is large consensus in the software visualization and also in the broader information visualization community that a lack of proper evaluation that can demonstrate the effectiveness of tools is detrimental to the development of the field [SOT09].

2. Involve industry. As stated by, Wohlin, in [Woh07], students' behavior when participating in such controlled experiments may differ from industrial behavior and needs. So, one could think that the easy solution would be to transfer all the results to an industry grade, but one would also be naive to expect a student to summarize and present research results at industrial grade [SDJ07].

3. Account for Experience. Taking into account that the participant experience might bias the results, leading to an undesirable outcome, applying *blocking* [Woh+00] would minimize this effect. Therefore, the subjects would be divided into groups according to their experience.

4. Provide enough exposure. It was noticed that providing enough exposure to the tool can have great influence in the outcome of the experiment. Thus, participants should be allowed sufficient time to study and understand the tools they will use [SOT09].

6.2 Research Question and Hypothesis

Q1 : Does the use of IoTCity help to ease the complexity of developing and maintaining an IoT architecture, compared to non-visual, live, configuration tools?

Q2 : Does the use of IoTCity help to ease the complexity of developing and maintaining an IoT architecture, compared to visual, non-live, configuration tools?

Null Hypothesis	Alternative Hypothesis
$H1_0$: The tool does not positively impacts the ability of easing the complexity of developing and maintaining IoT systems	$H1$: The tool positively impacts the ability of easing the complexity of developing and maintaining IoT systems

Table 6.1: The null and alternative hypotheses

6.3 Results Analysis

After analyzing the, small number of answers to the survey, 18, in the following lines is presented the results one could infer. Needless to say, that the survey was anonymously done in order to reduce as much as possible the bias. Before describing it is important to mention that we considered the *maybe* answer as a positive answer although with a skeptical vision.

As shown by Figure 6.1 all of the answers were made by people related to the Information Technologies field, mainly due to the fact that the survey was published in our domain channels. This also generates bias since people working in such field tend to *accept*, read it accept to try, changes claiming that their job would be easier. On top of that, it does not also provide vision from the outside of IT community.

Figure 6.2 represents that only one person is not familiar with IoT at all. Although not visible in this graphic but described in A.1 this same person, 15th row, believe that a solution with real time capabilities may be able to reduce complexity as well as an aggregation of these capabilities with visualization is a go to solution.

Getting deeper into the concepts applied a question to check whether the participants were aware of them was made. It is possible to describe that fifty percent of them were aware of the concept of Live Programming as well as the rest were not aware, plotted in Figure 6.3. Almost

the same happens in the concept of Software Visualization, the difference is that plus 5.55% have knowledge of it. Figure 6.4 describes the above.

Even though we notice that half of the participants do not have knowledge of the Live Programming concept, 83.33(3)% think that a solution with real time capabilities, what can be considered as a form of Live Programming, can reduce complexity of developing and maintaining an IoT system, as shown by Figure 6.5. The same type of behavior is possible to describe in the concept of Software Visualization. In spite of 44.45% of participants not having acquaintance about Software Visualization 94,44% believe that a tool with visualization capabilities also eases the complexity, as drawn in Figure 6.6. To finalize this thought path, in Figure 6.7 it is also possible to infer that 88.88(8)% of the participants believe in a solution mixing both the Live Programming and Software Visualization as a possible step to reduce complexity on developing/maintaining IoT systems.

A more in-depth analysis shows us that even participants that are not familiar with Live Programming have a positive thinking about a solution with such capabilities as presented on Figure 6.8. Following the line of analysis, according to Figure 6.9 it is proven that participants without knowledge about Software Visualization also believe in a solution with such capabilities. To finalize such analysis a last plot, Figure 6.10, shows us that even if people do not have acquaintance of such concepts, Live Programming and Software Visualization, they have a positive attitude towards a solution mixing both concepts. On the same path are the ones that were already familiar with these concepts, as shown by figure 6.11.

To sum up, what we can infer from the previous analysis is that the participants were open to test out such features in a new *way to develop/maintain* systems. Although the survey is not enough to prove that this is one hundred percent true, we believe it is enough to prove that a *market* exists in such domain that is definitely worth to be explored.

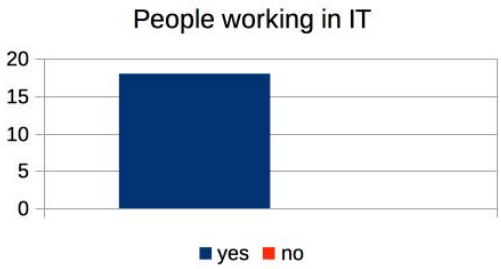

Figure 6.1: Histogram representing the number of people that answered the questionnaire and are related to the IT field

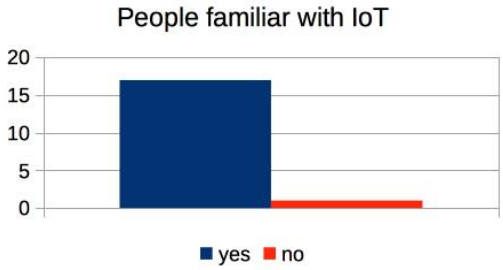

Figure 6.2: Histogram representing the number of people that answered the questionnaire and are familiar with IoT

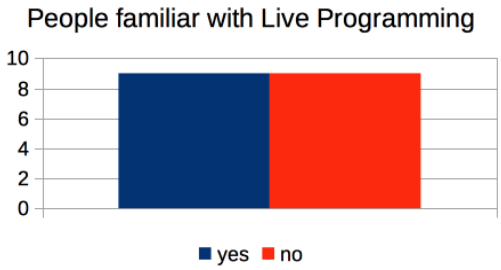

Figure 6.3: Histogram representing the number of people that answered the questionnaire and have knowledge of Live Programming

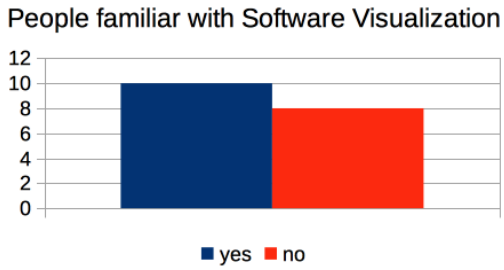

Figure 6.4: Histogram representing the number of people that answered the questionnaire and have knowledge of Software Visualization

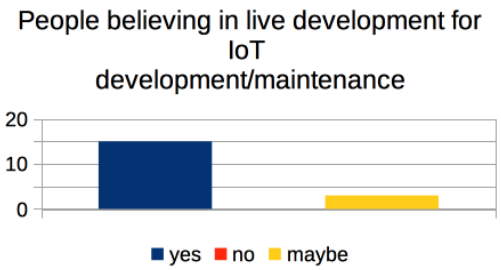

Figure 6.5: Histogram representing the number of people that answered the questionnaire and believe Live Programming eases development/maintenance of IoT systems

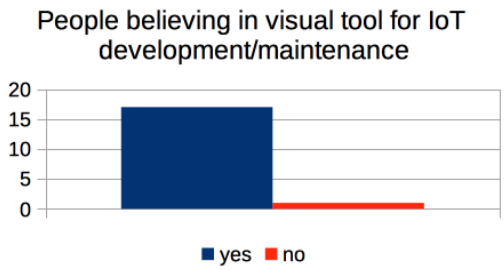

Figure 6.6: Histogram representing the number of people that answered the questionnaire and believe Software Visualization eases development/maintenance of IoT systems

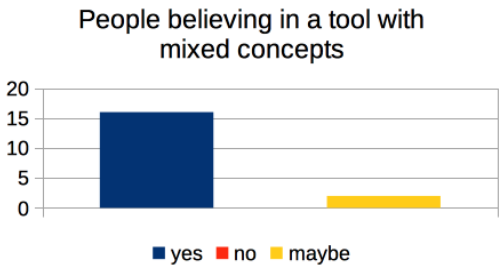

Figure 6.7: Histogram representing the number of people that answered the questionnaire and believe Live Programming plus Software Visualization eases development/maintenance of IoT systems

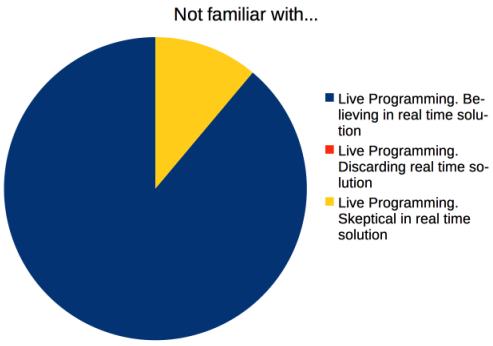

Figure 6.8: Pie chart representing participants attitude out of which are not familiar with Live Programming

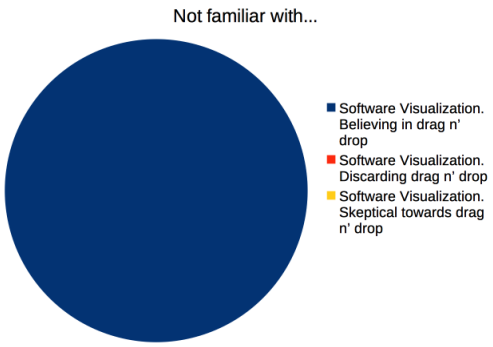

Figure 6.9: Pie chart representing participants attitude out of which are not familiar with Software Visualization

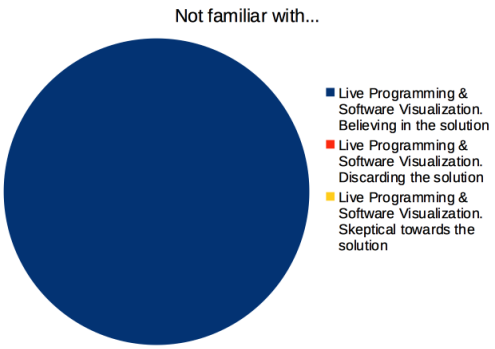

Figure 6.10: Pie chart representing participants attitude out of which are not familiar with Software Visualization and Live Programming

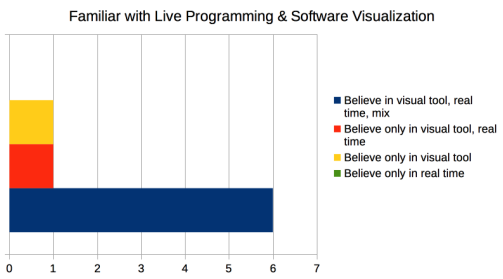

Figure 6.11: Bar chart representing participants attitude out of which are familiar with Software Visualization and Live Programming

6.4 Validation Threats

In this section we present some threats to the validation of the proposed solution. These threats may distort the scientific evidence and jeopardize the outcome of the experience, incorrectly supporting results. Although recognized they're intended to be the focus on future studies.

- Analysis on a considerable size architecture. One of the main concerns with this work is the mitigation of complexity, thus leading to the easing of the development and maintenance of IoT systems. However, this experiment was conducted considering a small size infrastructure, so that we were able to prepare some tasks to be feasible. Nevertheless, it would be interesting to develop a test case where the infrastructure is bigger, trying to extract the same variables, so we could compare if the outcome is still the same and incorporate it in the next researches.

- The support of validation. The proposed validation is likely to have a little ground of support. Although, given the time and scope of the thesis it was interesting to have such findings. However, clearly, it poses a real threat, hence a proper validation should be considered in order to further continue the development.

  There should be a lot of other threats to the proposed validation, as clearly a survey has not enough basis to proper validate such theme, though, these are what we considered to be the most important in this phase.

6.5 Conclusion

As we've been seeing throughout this chapter, there is a strong need to do a real case scenario validation. In this phase, it would be branching off the scope and time constraints. Nevertheless, with the provided survey it is possible to infer that there was a positive attitude towards such domain by participants related to IT. This can be said with only a certain degree of trust, although, we strongly believe that this is an upcoming area and as all the upcoming areas they

start through uncertainty until they reach a mature level where they can start to enhance its certainty.

To sum up, we believe, for what we could extract from the survey, that this project has the potential to grow exponentially, even though, at this moment, we are completely aware that it is in its embryonic phase.

Chapter 7

Conclusion

Throughout this thesis, an analysis to the challenges and evolution about Internet of Things was performed. Starting by describing the challenges the IoT field is surpassing and, afterwards, how it tends to evolve. It is important to notice that this field of work is constantly evolving to match demanding markets therefore causing an unavoidable increment on complexity and heterogeneity, leading to a complete shattered and overwhelming environment, especially for the developers and managers.

What we try to do on our approach is to combine the strength of multiple existing approaches, techniques and tools, mainly focusing on the introducing of liveness concept, providing, as real time as possible, a feedback loop that allows the developers to understand how the infrastructure reacts to the change, plus the immersive experience, allowing them to have a broader vision of the system they are developing or maintaining.

7.1 Summary of Contributions

From the point of view of IoT development and management, the contribution is an integrated environment enabling analysis, architecture, configuration, development and management of devices with a higher level of abstraction. Thus, enabling developers to focus on specific areas plus allowing an easy way of tracking changes as complexity increases.

Another contribution would be the fact of merging visualization concepts with live programming concepts allowing, in this way, to better understand how the system reacts to a change as well as how it influences the outcome of the system. In IoTCity for every interaction one has with the devices an action is triggered though a message pipeline, thus meaning liveness is as fast as the messaging system allows. Implementing it like this allowed us to remotely program devices and inspect the outcome, as soon as possible.

Finally, one last, and what we consider to be the most important contribution is the outline of the main validation threats, that should be taken into consideration for future work related with this thesis.

7.2 Future Research

Some issues were addressed by this thesis although some other new ways of improvement were thought. Some of these improvements might as well be the following:

- Development of a proper validation. One of the biggest challenges was trying to develop a validation form that allowed to properly extract knowledge to support the basis of conclusions, due to multiple factors, such as, time constraints, scope limitations and disparity of tools features. In spite of not achieving it in this phase, it would be very important to do so, and in the next iteration it will be something to definitely look at in the first place.

- Usage of a 3D environment. To provide a real immersive environment a 3D model would have to be used. It is still an open question whether this type of environment would have helped or not in this specific case, which would be a valid reason to try it in a future version

and compare the results. Such way we would be able to infer whether the system would be more efficient in a 3D environment.

- Enhancement of metaphors. Considering this as an exploratory work, we do not consider the solution to be an elixir to represent IoT infrastructures and devices. Although the ones we present could be enhanced, it is still needed to find new ones to strengthen the relation between the real and virtual world.

- Reading already implemented systems. Another technique that might as well be a must have in future versions is the ability to scan through already implemented systems. It is known that companies do not want to waste time and money on a new system, if they have one going. Therefore, such capability would be the perfect excuse to further testing the solution in a real case.

- Usage of a layout technique. As of this moment and because we didn't work with such big infrastructures, the solution does not provide an easy way of positioning the resources. This might be a drawback in user experience, thus a technique such as this might be needed in future versions of the solution.

- Study in an industrial case scenario. To better evaluate our solution an industrial test case should have been made. Although this was not possible at the time, due to time constraints and the huge effort needed, in the future, when the solution becomes more mature, we would like to test it in an industrial case scenario, to test possible different conclusions.

- Exploration of liveness level four. Creating an architecture supporting liveness level four would possibly be a burden. Adding to the fact the time constraints we decided to stay in level three for the initial phase of implementation. However, it would be interesting to support the level four as it would allow us to *navigate through time* and explore different states.

7.3 Epilogue

Recalling to where it all started, one year ago, it is impossible to not be proud of the work done. Although working at CERN during the course of this thesis, I was broadly stimulated with the idea of developing something related to IoT, one of my gray areas, plus the fact of adding visualization and liveness concepts to it. Just the feelings of starting the development of something that might allow me to *feel* the system, got into my soul. However, it was critical to understand that it was the starting of a big investigation on an upcoming area which could mean the results were not as expected, therefore not answering to the initial vision. Analyzing what was done it is humble to say that this is not a complete solution for the development and management of IoT resources, let alone being completed. Nevertheless, modest to say it is a contemporary approach intending to inspire new ideas within the domain.

Appendix A

Survey

A.1 What does this survey try to grasp

This survey tries to mine from peoples' opinion whether this domain has potential to be continuously developed and if it has a place on the market. Although the number of answers is not enough to prove anything a table with the answers is provided in Figure A.1.

Do you work in IT?	Are you familiar with Internet of Things?	Are you familiar with the concept of Live Programming?	Are you familiar with the concept of Software Visualization?	Would you think your life would be easier if the development/maintenance of an IoT system was to be done with a visual tool (drag n' drop)?	Would you think your life would be easier when you're developing/maintaining an IoT system if you could see the changes happening in real time?	Would you think a visual solution with real time capabilities would be able to reduce the complexity of understanding/developing/maintaining IoT systems?
Yes	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	No	No	With drag n' drop already limits the amount of possoble choices making easy the process.	Yes, it helps to get other perspective	Yes
Yes	Yes	Yes	Yes	yes	yes	Yes
Yes	Yes	Yes	Yes	Yes. Currently these systems still use a lot command line to build the modules and components which when we are starting to learn to work with these systems it is not always a smooth path. So a drag n' drop tool it will be great, because it decreases the complexity of building an IOT system.	Yes. Sometimes when leading with IOT systems some errors may ocurre and we can't determin where it happen because of the delay. If we can visualize somehow these changes in real time it will reduce a lot the difficulty in mantain and develop these sistems.	Yes
Yes	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	No	Yes	yes	yes	Yes
Yes	Yes	Yes	Yes	Yes, to reduce the complexity from having multiple ioT instances.	For sure, however from a visualization prespective, if we consider understanding how the system reacts to change.	Yes
Yes	Yes	Yes	Yes	Yes, although typically these tools can't handle the inherent complexities in IoT. It would be very dependant on the tool's intelligence and ability to interact with other systems.	Maybe. But then, how real time is real time? If I'm working on something, I can push code and see the results in under two minutes. So, for me, that's probably fast enough.	Maybe
Yes	Yes	No	No	Yes as long as the resulting code would be clean	Yes	Yes
Yes	Yes	No	No	yes, because of usability	yes, as it gives the user some instant response to their actions	Yes
Yes	Yes	No	No	Yes. Since these are technologies that will become part of our daily lives, they should be as closest to reality as possible. That is making development/ maintenance user-friendly.	Yes. Having the chance to see changes happening in real time will also help users to better understand how the IoT system works.	Yes
Yes	Yes	No	No	yes	yes	Yes
Yes	Yes	Yes	Yes	Yes.	Yes.	Maybe
Yes	No	No	Yes	No, I'm not a IoT consumer	Maybe, if it is on an test environment yes. I see it a little big dangerous for running productions.	Yes
Yes	Yes	No	No	yes. It would be easy to steer the IoT system without spending too much time on learning non-intuitive tools.	Yes it would be easier to debug.	Yes
Yes	Yes	Yes	No	Yes, it would be easier. Although, you have to make sure that the user will do the appropriate actions that will end up in the desired behavior of the system.	Maybe.	Yes
Yes	Yes	Yes	Yes	Yes. As a developer would be interesting to develop/maintain a system where drag and drop features would turn our life a bit easier and less prone to commit mistakes. As a user it would be great since it improves the UX hence making the users attracted to it.	Yes, definitely. It depends on how real time it is. Though, it would be great to react to changes in real-time.	Yes
Yes	Yes	No	No	Yes, because it could be easy create an interaction between IoT devices.	Yes because you have an immediate feedback of what you are developing.	Yes

Figure A.1: The table with the answers to the developed survey

References

[AlF+15] Ala Al-Fuqaha et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys and Tutorials* 17.4 (2015), pp. 2347–2376. ISSN: 1553877X. DOI: 10.1109/COMST.2015.2444095. arXiv: arXiv:1011.1669v3 (cit. on p. 3).

[Ben12] Richard Bender. "Systems Development Life Cycle : Objectives and Requirements". In: *Bender RBT Inc.* (2012), pp. 5–29. URL: http://www.benderrbt.com/Bender-SDLC.pdf (cit. on pp. 14, 42).

[BL14] Michael Blackstock and Rodger Lea. "Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED)". In: *Proceedings of the 5th International Workshop on Web of Things - WoT '14* (2014), pp. 34–39. ISSN: 15708705. DOI: 10.1145/2684432.2684439. arXiv: 9605103 [cs]. URL: http://dl.acm.org/citation.cfm?doid=2684432.2684439 (cit. on pp. 30, 31).

[BPB12] Gianmarco Baldini, Trevor Peirce, and Maarten Botterman. "Internet of Things". In: *International Workshop, IOT 2012* 2.1 (2012), pp. 2–5. ISSN: 1074-5351. DOI: 10.1002/dac.2417. arXiv: 96332259. URL: http://link.springer.com/content/pdf/10.1007/978-3-642-11710-7.pdf (cit. on p. 2).

[Cal+13] Oscar Callaú et al. *How (and why) developers use the dynamic features of programming languages: The case of smalltalk*. Vol. 18. 6. 2013, pp. 1156–1194. ISBN: 1066401292. DOI: 10.1007/s10664-012-9203-2 (cit. on pp. 21, 22).

[Che+14] Shanzhi Chen et al. *A vision of IoT: Applications, challenges, and opportunities with China Perspective*. 2014. DOI: 10.1109/JIOT.2014.2337336 (cit. on pp. 3, 18, 40, 41).

[Col18] Louis Columbus. Available at https://www.forbes.com/sites/louiscolumbus/2018/06/06/10-charts-that-will-challenge-your-perspective-of-iots-growth. 2018 (cit. on p. 2).

[Cru+16] Adriana Cruz et al. "Software Visualization Tools and Techniques: A Systematic Review of the Literature". In: (2016) (cit. on pp. 27, 29).

[CZB11] Pierre Caserta, Olivier Zendra, and Damien Bodenes. "3D hierarchical edge bundles to visualize relations in a software city metaphor". In: *Proceedings of VISSOFT 2011 - 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis* (2011). DOI: 10.1109/VISSOF.2011.6069451 (cit. on pp. 27–29).

[Die07] Stephan Diehl. *Software visualization: Visualizing the structure, behaviour, and evolution of software*. 2007, pp. 1–187. ISBN: 9783540465041. DOI: 10.1007/978-3-540-46505-8. arXiv: arXiv:1011.1669v3 (cit. on pp. 23, 24).

[Doc16] Docker Inc. "Docker for the Virtualization Admin". In: (2016), p. 12 (cit. on p. 35).

[Gab96] Richard P. Gabriel. "Patterns of Software Tales from the Software Community". In: *Architecture* 239 (1996), xx, 235 p. ISSN: 02896540. URL: http://www.questia.com/PM.qst?a=o%7B%5C&%7Dse=gglsc%7B%5C&%7Dd=86946694 (cit. on p. 17).

[GEC98] Nahum Gershon, Stephen G. Eick, and Stuart Card. "Information Visualization". In: *Interactions* 5.2 (1998), pp. 9–15. ISSN: 10725520. DOI: 10.1145/274430.274432. arXiv: 9809069v1 [arXiv:gr-qc] (cit. on p. 23).

[Goo] Google. Available at https://cloud.google.com/solutions/iot/ (cit. on p. 30).

[Gra12] Chris Granger. Available at https://www.kickstarter.com/projects/ibdknox/light-table. 2012 (cit. on p. 22).

[HTS] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. "MQTT-S – A Publish/Subscribe Protocol For Wireless Sensor Networks". In: () (cit. on pp. 36, 37).

[IBM15] IBM. Available at https://www.ibm.com/blogs/bluemix/wp-content/uploads/2015/02/nodered.png. 2015 (cit. on p. 31).

[Kha+12] Rafiullah Khan et al. "Future internet: The internet of things architecture, possible applications and key challenges". In: *Proceedings - 10th International Conference on Frontiers of Information Technology, FIT 2012*. 2012. ISBN: 9780769549279. DOI: 10.1109/FIT.2012.53. arXiv: 1207.0203 (cit. on pp. 3, 4, 18, 40, 41).

[KJo09] Reza Sherafat Kazemzadeh and Hans Arno Jacobsen. "Reliable and highly available distributed publish/subscribe service". In: *Proceedings of the IEEE Symposium on Reliable Distributed Systems* (2009), pp. 41–50. ISSN: 10609857. DOI: 10.1109/SRDS.2009.32 (cit. on p. 37).

[Lea+12] YB Leau et al. "Software Development Life Cycle AGILE vs Traditional Approaches". In: *International Conference on Information and Network Technology (ICINT 2012)* 37.Icint (2012), pp. 162–167 (cit. on p. 14).

[Leh80] Meir M. Lehman. "Programs, Life Cycles, and Laws of Software Evolution". In: *Proceedings of the IEEE* 68.9 (1980), pp. 1060–1076. ISSN: 15582256. DOI: 10.1109/PROC.1980.11805 (cit. on pp. 2, 3, 41).

[Lib] Node-RED Flow Library. Available at https://flows.nodered.org/?num_pages=1 (cit. on p. 31).

[LP05] Welf Löwe and Thomas Panas. "Rapid Construction of Software Comprehension Tools". In: *International Journal of Software Engineering and Knowledge Engineering* 15.06 (2005), pp. 995–1025. ISSN: 0218-1940. DOI: 10.1142/S0218194005002622. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218194005002622 (cit. on p. 25).

[Lyn12] Matt Lynley. Available at http://www.businessinsider.com/this-startup-is-finally-changing-the-way-programming-works-after-more-than-30-years-2012-8?IR=T. 2012 (cit. on p. 22).

[McD07] Sean McDirmid. "Living it up with a live programming language". In: *ACM SIGPLAN Notices* 42.10 (2007), p. 623. ISSN: 03621340. DOI: 10.1145/1297105.1297073 (cit. on pp. 21, 22).

[Nod] Node-RED. Available at https://nodered.org/ (cit. on p. 31).

[Pan+07] Thomas Panas et al. "Communicating Software Architecture using a Single-View Visualization". In: *Babel* Iceccs (2007), pp. 217–228. DOI: 10.1109/ICECCS.2007.20. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4276318%7B%5C%7D5Cnhttp://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Communicating+Software+Architecture+using+a+Unified+Single-View+Visualization%7B%5C#%7D0 (cit. on p. 26).

[Pan05] Thomas Panas. "The vizzanalyzer handbook". In: *Architecture* October (2005) (cit. on pp. 25, 26).

[Rei01] Steven P Reiss. "An overview of BLOOM". In: *Proc. 2001 ACM SIGPLAN-SIGSOFT Work. Progr. Anal. Softw. tools Eng. - PASTE '01* (2001), pp. 2–5. DOI: 10.1145/379605.379629. URL: http://portal.acm.org/citation.cfm?doid=379605.379629 (cit. on p. 27).

[Ros+94] L Rosenblum et al. *Scientific Visualization—Advances and Challenges*. 1994 (cit. on p. 23).

[RR02] Steven P Reiss and Manos Renieris. "The BLOOM Software Visualization System". In: (2002), pp. 1–29 (cit. on pp. 27, 28).

[Sam12] Mohamed Sami. *Software Development Life Cycle Models and Methodologies*. Available at https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies. 2012 (cit. on p. 14).

[Sam16] S. Sujin Issac Samuel. "A review of connectivity challenges in IoT-smart home". In: *2016 3rd MEC International Conference on Big Data and Smart City, ICBDSC 2016*. 2016. ISBN: 9781509013654. DOI: 10.1109/ICBDSC.2016.7460395 (cit. on pp. 40, 41).

[SBF14] IEEE Computer Society, Pierre Bourque, and Richard E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014. ISBN: 0769551661, 9780769551661 (cit. on pp. 8–14).

[SDJ07] Dag I K Sjøberg, Tore Dybå, and Magne Jørgensen. "The Future of Empirical Methods in Software Engineering Research". In: *Future of Software Engineering FOSE 07* 1325 (2007), pp. 358–378. ISSN: 00985589. DOI: 10.1109/FOSE.2007.30. URL: http://dx.doi.org/10.1109/FOSE.2007.30 (cit. on p. 58).

[Som07] Abhijan Bhattacharyya Soma Bandyopadhyay. "Lightweight Internet Protocols for Web Enablement of Sensors using Constrained Gateway Devices". In: *European Physical Journal C* 50.2 (2007), pp. 299–314. ISSN: 14346044. DOI: 10.1140/epjc/s10052-007-0257-z. arXiv: hep-ex/0609050 [hep-ex] (cit. on p. 37).

[SOT09] Mariam Sensalire, Patrick Ogao, and Alexandru Telea. "Evaluation of software visualization tools: Lessons learned". In: *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis* (2009), pp. 19–26. DOI: 10.1109/VISSOF.2009.5336431. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5336431 (cit. on p. 58).

[SS17] Jordi Salazar and Santiago Silvestre. *Internet of Things*. 2017, pp. 1–31. ISBN: 9781786301512 (cit. on pp. 15, 18).

[Tab] Light Table. Available at http://lighttable.com/ (cit. on p. 22).

[Tan13] Steven L. Tanimoto. "A perspective on the evolution of live programming". In: *2013 1st International Workshop on Live Programming, LIVE 2013 - Proceedings*. 2013, pp. 31–34. ISBN: 9781467362658. DOI: 10.1109/LIVE.2013.6617346 (cit. on p. 20).

[Tec16] Kumulus Technologies. Available at https://www.quora.com/What-is-a-software-container. 2016 (cit. on p. 36).

[Thi] Thingsboard. Available at https://thingsboard.io/ (cit. on p. 30).

[TS08] David Trenholme and Shamus P. Smith. "Computer game engines for developing first-person virtual environments". In: *Virtual Reality* (2008). ISSN: 13594338. DOI: 10.1007/s10055-008-0092-z (cit. on p. 29).

[WL07] Richard Wettel and Michele Lanza. "Program Comprehension through Software Habitability.pdf". In: (2007) (cit. on pp. 24, 25).

[WLR11] Richard Wettel, Michele Lanza, and Romain Robbes. "Software systems as cities". In: *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. 2011, p. 551. ISBN: 9781450304450. DOI: 10.1145/1985793.1985868. arXiv: arXiv:1011.1669v3. URL: http://portal.acm.org/citation.cfm?doid=1985793.1985868 (cit. on pp. 24, 25).

[Woh+00] Claes Wohlin et al. *Experimentation in software engineering An Introduction*. 2000. ISBN: 9781461370918. DOI: 10.1007/978-1-4615-4625-2 (cit. on p. 58).

[Woh07] Claes Wohlin. "Empirical Software Engineering : Teaching Methods and Conducting Studies". In: *Dagstuhl Seminar Proceedings (LNCS 4336)* Lncs 4336 (2007), pp. 135–142. ISSN: 03029743 (cit. on p. 58).

[Zha+14] Zhi Kai Zhang et al. "IoT security: Ongoing challenges and research opportunities". In: *Proceedings - IEEE 7th International Conference on Service-Oriented Computing and Applications, SOCA 2014*. 2014. ISBN: 9781479968336. DOI: 10.1109/SOCA.2014.58 (cit. on pp. 3, 40, 41).

[ZJL15] Cheah Zhao, Jayanand Jegatheesan, and Son Chee Loon. "Exploring IOT Application Using Raspberry Pi | BibSonomy". In: *International Journal of Computer Networks and Applications* 2.1 (2015), pp. 27–34. ISSN: 2395-0455 (cit. on p. 33).