FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Model-to-Model Mapping of Semi-Structured Specifications to Visual Programming Languages

Danny de Almeida Soares

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo Sereno Ferreira, PhD

Second Supervisor: André Restivo, PhD

July 30, 2020

Model-to-Model Mapping of Semi-Structured Specifications to Visual Programming Languages

Danny de Almeida Soares

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Gil Gonçalves

External Examiner: Prof. Ângelo Martins

Supervisor: Prof. Hugo Sereno Ferreira

July 30, 2020

Abstract

The Internet of Things (IoT) is most commonly used in home automation, in smart houses equipped with devices that can be interacted with remotely, such as smart lights, ovens, A/C systems, etc. One of its goals is to use technology to control and monitor these smart spaces, such as allowing users to turn the A/C on and off or set it to a specific temperature, or check the doors and windows, even if they are not at home, through their smartphones or computers.

Smart assistants are one way to simplify the interaction between users and devices, providing control via a natural language conversational interface — either voice or text-based. Examples of these tools are Siri, Google Assistant and Alexa, which act as mediators for (verbal) interactions between humans and machines. Despite their demonstrable applicability to basic tasks, smart assistants are still limited. Current solutions do not provide the ability to manage complex or recurring actions, and so are unable to provide a complete management platform for IoT systems using just conversational commands. Besides, one can easily imagine that managing tens or hundreds of devices and rules without a more structured medium would probably be chaotic and prone to failure. Alternatively, Visual Programming Platforms (VPP) are more capable of dealing with this complexity, providing the user a way of dragging blocks into a canvas and connecting them to convey causality. This flexibility, however, comes at the cost of a higher learning curve when compared to conversational interfaces, making it difficult to interact with them and set up an organically-grown IoT system effortlessly, such as those users have at their homes.

We propose mixing the two approaches, by using a semi-structured text-based interface for behavior specification, supported by a semi-structured language (very close to natural language). To do so, we follow an approach similar to a popular semi-structured language, Gherkin, and leverage an also popular VPP, Node-RED. With this, we can then convert between the two representations (textual and visual), through bi-directional model-to-model transformations.

Our goal is to understand whether combining a VPP and a semi-structured text-based interface makes it easier for users to understand how an IoT system is working and how they can manage the interactions of the devices in the system. To evaluate this, we present one survey and one case study that were performed with participants with different technological backgrounds. Our results show that our tool supports 93% of the 177 scenarios submitted by 20 participants, and that participants had a 97% success rate describing scenarios implemented with our tool compared to only 68% with Node-RED. Therefore, we provide evidence that a semi-structured text-based interface is easier for end-users to understand home automation scenarios, comparing to a VPP, and we believe that it improves their capabilities to manage complex IoT systems. Therefore, we believe that this project can have a positive impact on the evolution of IoT management tools.

Keywords: Internet of Things, Visual Programming, Domain-Specific Languages, Semi-Structured Languages

Resumo

A Internet das Coisas (IoT) é frequentemente usada para automação de casas, em casas inteligentes equipadas com dispositivos que podem ser controlados remotamente, como luzes inteligentes, fornos, sistemas de ar condicionado, etc. Um dos objetivos de IoT é usar a tecnologia para controlar espaços inteligentes, permitindo aos utilizadores ligar e desligar o ar condicionado, ou verificar as portas e janelas, mesmo que não estejam em casa, utilizando um computador ou telemóvel.

Os assistentes virtuais inteligentes são uma solução para simplificar as interações entre pessoas e dispositivos, permitindo aos utilizadores controlar os dispotivos através de uma interface conversacional — quer por voz, quer por texto. A Siri, o Google Assistant e a Alexa são exemplos destes assistentes, que atuam como mediadores para interações (verbais) entre humanos e máquinas. Apesar de demonstrarem sucesso ao serem aplicadas a tarefas básicas, os assistentes virtuais são ainda limitados. As soluções atuais não equipam o utilizador com a capacidade de gerir ações complexas ou recorrentes, sendo, portanto, incapazes de gerir um sistema IoT suficientemente complexo, usando apenas comandos conversacionais. Além disso, é fácil imaginar que gerir dezenas ou centenas de dispositivos sem um meio mais estruturado acabaria por se tornar caótico e mais sujeito a falhas. Como alternativa, existem ferramentas de programação visual, que são capazes de lidar com complexidade elevada, dando aos utilizadores a possibilidade de arrastar blocos para uma tela e conectá-los para representar causalidade. Esta flexibilidade traz a desvantagem de uma curva de aprendizagem mais acentuada, comparando com assistentes virtuais, sendo difícil interagir com as ferramentas e configurar um sistema IoT que tenda a crescer, semelhante aos dos utilizadores.

Neste projeto, propomos juntar as duas abordagens, usando uma interface textual semi estruturada para especificar cenários, suportada por uma linguagem semi estruturada (próxima de linguagem natural). Para isso, seguimos uma abordagem semelhantes à de uma linguagem semi estruturada popular, Gherkin, e tiramos proveito de uma VPP também popular, Node-RED. Com isto, podemos depois converter entre as duas representações (textual e visual), através de transformações de modelo bidirecionais.

O nosso objetivo é perceber se combinando programação visual e uma interface textual semi estruturada, fica mais fácil para os utilizadores compreenderem como um sistema IoT funciona e gerir as interações entre os dispositivos do sistema. Para validar isto, apresentamos uma recolha de cenários e um caso de estudo, feitos com participantes de diferentes níveis de experiência. Os resultados mostram que a ferramenta suporta 93% dos 177 cenários submetidos por 20 participantes, e que os participantes tiveram sucesso a interpretar 93% dos cenários descritos com a nossa linguagem, mas apenas 68% dos cenários descritos com Node-RED. Portanto, apresentamos evidências de que uma interface textual semi estruturada facilita a compreensão de cenários de automações para os utilizadores, comparando com uma VPP, e também que aumenta as suas capacidades de gerir sistemas IoT complexos. Sendo assim, acreditamos que este projeto pode ter um impacto positivo na evolução de ferramentas de gestão de IoT.

Palavras-chave: Internet das Coisas, Programação Visual, Linguagem de Domínio Específico, Linguagem Semi Estruturada

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisors, Hugo Sereno Ferreira and André Restivo, for their continuous support during the past months. Their knowledge and experience as well as their patient guidance made this dissertation possible. Also, a "thank you" to João Pedro Dias, for his support and insights during this dissertation.

I would also like to thank my family for their constant support and love, not only during this dissertation, but ever since I can remember. And also a special thanks to my girlfriend, Joana, who, being the closest person in my life, has given me a lot of support and motivation.

To my closest friends, mainly my workout buddies and my calisthenics partners, who have motivated me in ways that I never would have expected, and the ones from *IoTices*, who have accompanied this project and provided some thoughtful insights.

Finally, to all my friends, my family, my teachers, colleagues and everyone that has ever stepped in my way, I would like to express my sincere gratitude, for helping me get to where I am.

Danny de Almeida Soares

"The more that you read, the more things you will know.
The more that you learn, the more places you'll go."

Dr. Seuss

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BDD	Behaviour-Driven Development
DSL	Domain Specific Language
GUI	Graphical User Interface
IDE	Integrated Development Environment
IFTTT	If-This-Then-That
IoT	Internet of Things
JSON	JavaScript Object Notation
MDE	Model-Driven Engineering
NFC	Near Field Communication
NLP	Natural Language Processing
PIM	Platform-Independent Models
PSM	Platform-Specific Models
QVT	Query/View/Transformation
RFID	Radio Frequency Identification
SR	Speech Recognition
TTS	Text-to-Speech
VPL	Visual Programming Language
VPP	Visual Programming Platform

Chapter 1

Introduction

This chapter introduces the context and motivation for this project, as well as the problems it aims to solve. Section 1.1 describes the context of this project in the fields of technology that it involves. Section 1.2 focuses on what motivated this project and its importance. Section 1.3 describes the problem that is focused by this project. Section 1.5 explains what we want to validate with this dissertation and the methodology to do so. Finally, Section 1.6 explains the structure of the rest of the document, as well as what content is present in it.

1.1 Context

The Internet of Things, or IoT, has been around for quite some time, and thus its definition has been evolving with the introduction of new concepts. This concept is described as the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states of the external environment [59].

With the growing number of objects with such capabilities, IoT has become more visible in the world, having many applications, such as home automation, industrial automation, medical aids, mobile healthcare, and many others [62]. As stated by Tanweer [3], the number of connected devices in 2025 is predicted to be around 75 billion, which shows a great increase from the predictions for 2020, which are that around 26 billion devices will be connected.

Through the years of IoT development, the need to simplify the interactions between devices

and humans, brought Smart Assistants into the loop, such as Siri[1], Alexa[2] or Google Assistant[3], which allow humans to interact with devices using their voice [50]. These assistants are still being explored and are very rudimentary, which forces end-users to use more complex configuration tools, such as Visual Programming Platforms (VPP)[4], that require a larger technical expertise.

1.2 Motivation

Even though the Internet of Things has already been around for some years, fully managing a complex IoT system is not an easy task for most end-users. As VPPs are complex tools, designed for experienced users, they require technical knowledge in software or hardware engineering, as well as experience with the tools, which most IoT end-users do not have [19]. Besides the technical knowledge required, managing and interacting with an IoT system requires users to sit in front of a computer, or work with their smartphones, which sometimes is not convenient or even possible. And although smart assistants solve this problem to an extent, by allowing users to sit on their sofas and say "turn the TV on", they do not fully solve it, since they cannot handle the creation of rules as simple as "if it is raining, turn the water sprinklers off".

1.3 Problem Definition

With home automation being one of the main applications of IoT, most end-users are not experienced with software or hardware, being unable to design and program an IoT system, or even manage it after being up and running. VPPs were brought into the field to help end-users manage their systems, but, as mentioned above, these tools require technical expertise and can be overwhelming for most users, due to their complex user interface, which poses a barrier for most users. Smart assistants were introduced to attempt to solve this problem, by allowing end-users to interact with IoT systems using voice commands in their native language. However, these assistants are still very basic and are not ready to manage complex systems or even complex rules. Therefore, the main problem that we have identified in this field is the lack of tools that provide simplicity of use and support for complex system management at the same time. This means tools that have a simple interface for the average user to be comfortable, leveraging natural language for example, and support for users to create automation scenarios for their systems.

1.4 Goals

The main goal of this dissertation is to understand whether a semi-structured text-based interface simplifies the management of IoT systems, providing both technical and non-technical users with the ability to do so. This goal is achieved by developing such interface, focused on textual

[1] https://www.apple.com/siri/
[2] https://developer.amazon.com/en-US/alexa
[3] https://assistant.google.com/
[4] Visual Programming Platforms are supported by Visual Programming Languages (VPL)

interactions, that interprets a semi-structured language very close to natural language, inspired in Behavior Driven Development [52] with Gherkin [29], into a visual programming format, such as Node-RED [43]. After having this mapping, users can then convert between the textual and visual representations, through model-to-model transformations. With this dissertation, we do not want to replicate what already works, we want to leverage the popularity and capabilities of Node-RED and complement it with the semi-structured interface, while still allowing technical users to manage their IoT systems with Node-RED. This system should ease the management of complex IoT systems for experienced users and users with little to no programming background.

1.5 Methodology

Considering the problem identified and the goals defined for this dissertation, we want to validate the following hypothesis:

> *The integration of a semi-structured text-based interface and visual programming improves users capabilities to manage complex IoT systems.*

To do so, we will perform a survey to collect home automation scenarios, which will be used to evaluate whether our solution can support the management of complex IoT systems, and we will perform a case study to understand whether participants understand our semi-structured language better than Node-RED flows. This validation methodology should allow us to understand if the combination of a semi-structured text-based interface with a VPP allows IoT end-users to manage their systems regardless of the systems' complexity using a simple interface that is not overwhelming for them.

1.6 Report Structure

This chapter introduced the context of this dissertation, as well as the motivation, the problem it aims to solve and its goals. The remaining chapters of this document are the following:

- Chapter 2 (p. 5), **Background**, gives a background regarding the main concepts related to this project, explaining the concepts and going through their evolution.
- Chapter 3 (p. 17), **State of the Art**, describes the state of the art concerning this project, explaining what can currently be done in the scope of the project.
- Chapter 4 (p. 25), **Problem Statement**, describes the problem that is intended to be solved by this dissertation and the proposed solution to solve it. It also presents how the success of this project will be measured, in order to validate the hypothesis.
- Chapter 5 (p. 31), **Solution**, describes in detail the solution that we developed to solve the issues found in this field.

- Chapter 6 (p. 43), **Home Automation Survey**, and Chapter 7 (p. 55), **Case Study**, show the validation methodology that was followed, describing the survey and the case study that were performed.
- Chapter 8 (p. 63), **Conclusions and Future Work**, concludes this report, describing the final status of the project, the main contributions and future work.

Chapter 2

Background

Throughout this chapter, the main concepts related to this project will be introduced, along with a description of their evolution throughout the years. First, in Section 2.1, we go over the Internet of Things concept and some of its applications. In Section 2.2 we explain Visual Programming, along with its use in the IoT field. Section 2.3 describes Conversational Interfaces and Smart Assistants, as well as their application in IoT. Section 2.4 explains the concept of Model Transformations and how it can be applied in IoT. Finally, Section 2.5 summarizes the contents of this chapter.

2.1 Internet of Things

This section introduces the term Internet of Things, telling its history throughout the years, describing some of its main applications and how systems can be monitored in this field.

2.1.1 IoT Background

The Internet of Things is a concept that has been evolving throughout the years and has been mixing with other concepts in the realms of engineering, due to its many applications. The term itself was coined by Kevin Ashton[1], in 1999, as the title of a presentation at Procter & Gamble (P&G) [33]. Despite the term being popularized at that time, the concept is older than that, dating to 1990, when John Romkey[2] created what is believed to be the first IoT device, a toaster connected to a computer, that could be controlled over the internet [54]. After that, many devices started to be developed,

[1] Co-founder of Auto-ID Center, in the Massachusetts Institute of Technology.
[2] Co-founder and first president of FTP Software.

such as LG's smart refrigerator [26], in 2000, which became the first consumer product available incorporating the concept of IoT.

Besides the popularization of IoT, in 1999, Radio Frequency Identification (RFID) and Electronic Product Code (EPC) started to be more widely used, after Walmart and the United States Department of Defense started to require suppliers to add RFID tags to shipping pallets [60]. This type of identification allows for similar automatic detection as bar codes, however, RFID does not face some issues that bar code identification does, such as the need for humans in the loop, to align a scanner and the bar code for identification, or the possibility of being obscured or smudged, preventing scanners from reading the codes properly. Since RFID works with radio frequency signals, the tags can be identified without humans in the process or even if they are dirty. RFID also has a high rate of identification, due to parallelism, providing rates as high as hundreds of items per second [60]. This ability to identify objects so easily and fast is an important feature for IoT, allowing computers to identify any product that has an RFID tag attached.

From 2003 on, IoT started to enter the mainstream, and became more popular, with the first report about it being written in 2005 [31], the first European IoT Conference occurring in 2008 [27], and the term being used in popular publications, like The Guardian. Also in 2008, the Internet Protocol for Smart Objects (IPSO) Alliance was founded, with the goal of promoting the use of the Internet Protocol (IP) in smart object communications, allowing for more interoperability between these objects, trying to reach industry standards for this type of communication [7]. The public launch of IPV6, in 2011, also constitutes a great contribution to IoT, which along with the aforementioned developments led to the growth of IoT and of the number of connected devices.

Since the IoT explosion, some numbers have been thrown as predictions for the number of connected devices in the following years, with some predictions for 2020 putting that number close to 50 billion devices [54]. These numbers have, however, been scaled down by the authors, to around 30 billion devices by 2021 [44]. Some authors predict that these numbers will be even higher than 50 billion, by 2025, reaching 75 billion devices [3]. In regard to money, the IoT market was valued at $190 billion in 2018, and the authors predict that it will pass the barrier of $1 trillion, by 2026 [30].

One of the major concerns in IoT is the security and privacy of end-users, since sensors are collecting data related to the users, such as their location, their home routine, the time they spend away from home, medical information, amongst other. This type of information needs to be carefully collected, stored and handled, both on hardware and software levels, from collection with sensors, to storage or processing, respectively, ensuring access control, above all [54].

2.1.2 IoT Applications

As IoT grew, it started to be applied in more fields, and this continues to happen, which means that the Internet of Things has a lot of applications, some more popular than others. In this section, some of the most popular applications, according to Data Flair Team [24], are described and explained.

Smart homes are one of the most common IoT applications, and one of the main applications focused by this work. In this area, the installation of sensors and actuators in houses can improve

Figure 2.1: Nest products

Figure 2.2: Philips Hue kit

the life of the people who live in it, by adding more comfort to their life, but also by preventing energy waste. For example, having heaters, fans or A/C devices being monitored and controlled by an IoT system can improve the efficiency in terms of temperature management, also making the house more comfortable for residents and saving energy in the process [53].

As smart homes become more popular, companies are investing a lot more on the development of this field, and end-users also invest more on devices to automate their homes. As the number of IoT products for smart homes rises, some products reach a higher amount of users due to their popularity, ease of use and success in the tasks they should do. Some of the most popular smart home hubs (products to manage smart homes) are Amazon's Echo, powered by Alexa, and Brilliant Control[3], which also supports Alexa and works with most of the popular smart home platforms, such as Nest[4] or Philips Hue[5] [12]. As for smart devices, to be placed around the house, Nest and Philips Hue are two of the most popular brands in the field, also joined by Samsung's SmartThings[6] [36]. Figure 2.1 and Figure 2.2 show examples of some of these smart devices. The devices mentioned can be connected to and controlled by the hubs, but some devices can also be directly controlled by the end-user, without any hub in the middle, e.g., the Ecobee4 thermostat[7], which is a thermostat that can be interacted with directly, even supporting voice commands, since it works with Alexa, Google Assistant, etc [28].

Wearables are another relevant IoT application nowadays, due to their usefulness and simplicity. These products consist of smartwatches, smart bands, smart glasses, which have features such as monitoring health metrics, like heart rate, blood pressure or body temperature, tracing physical activity, receiving and sending messages or taking calls. There have been products to do each of these features for years, however, having one watch or bracelet that, besides showing the time, also shows the user's health status, activity level and allows them to read messages and emails without needing to pick their smartphones or laptops is useful and make life more comfortable for users. Some examples of these products are Google Glass[8], shown in Figure 2.3 (p. 8), which are glasses that display a small screen on top of one lens where the user can access multiple apps

[3] https://www.brilliant.tech/home
[4] https://nest.com/
[5] https://www.lighting.philips.pt/consumer/hue
[6] https://www.smartthings.com/
[7] https://www.ecobee.com/ecobee4/
[8] https://www.google.com/glass/start/

Figure 2.3: Google Glass

(maps, music, weather, etc), Garmin's Vivo series[9], which has various types of smartwatches that can track physical activity, health status, display the weather, receive messages or emails, besides showing the time, or Nadi X smart yoga pants by Wearable X[10], which help yoga practitioners to improve their form on exercises [1].

Smart cities are almost an expansion of smart homes, taking the concepts from smart homes and applying them to cities, using sensors to monitor and collect data and improve data sharing and coordination within cities [9]. The application of IoT in smart cities can show improvements in traffic, lighting, surveillance, garbage collection, among other areas [62]. Some countries are already using smart recycling systems, smart lights or parking sensors, in order to improve efficiency in these fields and save energy [38].

Healthcare is also seeing improvements due to the application of IoT technologies in the field, from small monitoring devices, to smart robots that assist in surgeries. The use of smartwatches or smart bands that can monitor the health status of the users can help doctors track patients' health status or even notify the doctors in case of emergency [54]. Using IoT technologies can also help tracking pharmaceutical products through their life cycle, to ensure their best conditions and track the expiry dates, to prevent users from taking damaged or expired products [54].

IoT is also being applied to Industry and Retail, throughout the whole supply chain. Using RFID or NFC tags allows industries to monitor products, resulting in better stock management, to provide better services to customers. NFC is also used in payments, allowing users to pay bills with NFC equipped smartphones. Sensors can detect irregular operations in industrial machines, by tracking temperature and vibration, which can prevent malfunctions and accidents and improve the efficiency and safety of these machines [54].

To sum up, there are already many products being developed for IoT, and many applications of it, simplifying day-to-day tasks, improving comfort at home and public spaces, improving efficiency in industries and retail or helping people track their health easily.

[9] https://explore.garmin.com/en-US/vivo-fitness/
[10] https://www.wearablex.com/

2.2 Visual Programming

In this section, Visual Programming is presented, explaining what it is, some fields where it is applied, with a focus on its application in the Internet of Things.

2.2.1 Visual Programming Background

As most fields in technology, Visual Programming has evolved throughout the years, but many authors continue to define it as a programming paradigm in which two or more dimensions are used, as Brad Myers did in 1986. This multidimensional aspect in programming allows users to express what they are visualizing in their mind easier, since it is the way people visualize their ideas [41]. Therefore, Visual Programming makes programming accessible to a wider range of people, by using visual elements such as blocks and connections to represent program flows.

Visual Programming Platforms (VPP) have been used in many fields, such as education, game development, simulations, automation, among others [4, 8, 48, 37]. Lately, the Internet of Things has also been an application for Visual Programming, with many VPP being used to program and manage IoT systems. Examples of these platforms are Node-RED[11], NETLabTK[12], Ardublock[13] or Scratch[14]. VPP are not all the same, both regarding features and usability, so they have been categorized as purely visual, hybrid text and visual systems, programming-by-example systems, constraint-oriented systems and form-based systems [6]. These categories are not mutually exclusive, and many tools are in fact associated with multiple categories.

2.2.2 Visual Programming in IoT

As a lot of IoT end-users are not experienced in the fields of software or hardware, setting an IoT system up and managing it with code is not feasible. However, as explained before, Visual Programming allows users to perform programming tasks easier, and, as Visual Programming is being applied to IoT, it makes it possible for non-technical users to have the opportunity to work on their IoT systems taking advantage of VPP like the ones mentioned above.

There are many VPP available, both open source and proprietary, which means that anyone can find one that suits them. Among the most popular tools, Node-RED comes on the top places, being one of the most used tools for IoT development [56]. This tool is open source and runs on a wide range of software and hardware, such as Raspberry Pi[15], Arduino[16], Docker[17], among others [48], and is accessed from a web browser, from where the work is done. Node-RED is built in JavaScript and works with visual flows built with nodes and connections, in which the nodes represent devices (sensors, actuators), operations provided by services and receive or produce data,

[11] https://nodered.org/
[12] http://www.netlabtoolkit.org/
[13] http://blog.ardublock.com/
[14] https://scratch.mit.edu/
[15] https://www.raspberrypi.org/
[16] https://www.arduino.cc/
[17] https://www.docker.com/

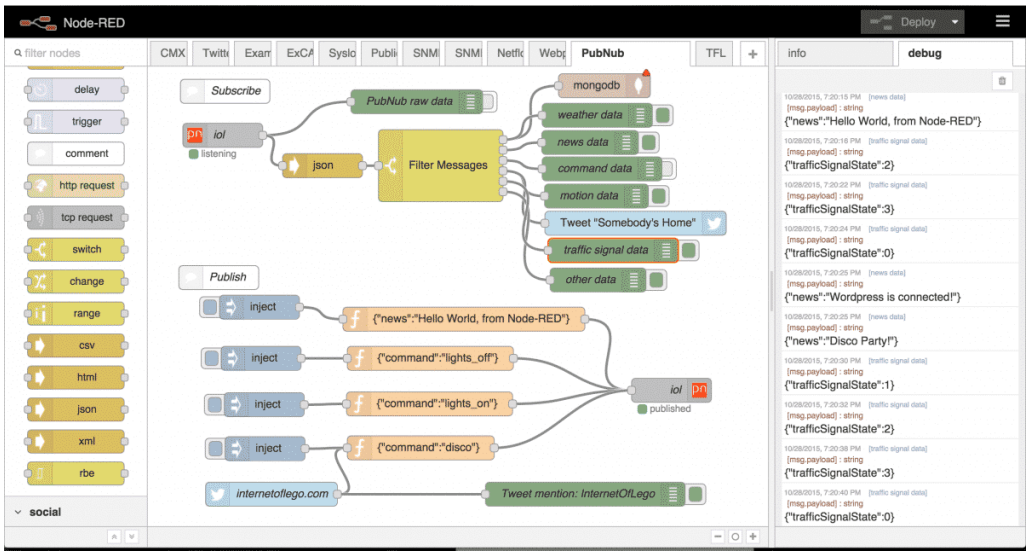

Figure 2.4: Node-RED GUI

and the connections represent the flow of data between nodes. The tool's company focuses a lot on the community, in users sharing with each others, so it has the possibility of storing flows in JSON format and sharing them among other users, even having an online library for this purpose.

Despite being one of the most popular tools, Node-RED has some limitations [16], which affect mainly debugging tasks. The tool does not display sufficient information about the internal status of nodes, or their connectors, does not use types for data, allowing the connection between nodes that give/expect completely different formats of data. Both these conditions make development harder for users and lead to the introduction of errors on the flow. Also, Node-RED provides little debugging support, using only debug nodes that display messages sent to them, which are not sufficient to properly debug a system. Features such as node inspection or history could ease the debugging tasks and, therefore, improve the performance of users. Besides debugging, Node-RED also hinders the development of complex systems, due to the amount of visual components that are displayed, as can be seen on Figure 2.4. This visual complexity makes the development and management of systems very hard for most users, and in general, the lack of feedback has been identified as one of the main current challenges in Software Development [2]

In 2015, Tiago Simões [51] enumerated some of the main problems with Visual Programming, stating that (1) Visual Programming Languages (VPL) are not extensible, allowing users to do a limited set of tasks, making edge cases too hard or even impossible, (2) VPLs generate slow code, which is impossible to optimize, causing performance problems that can hardly be fixed, (3) VPP can be terrible, in the sense that Integrated Development Environments (IDE) are not prepared to work with VPLs, which results in a bad integration and a hard time understanding the interface of some of these tools, (4) VPLs lock users in, since most VPLs generate low-level code that is hard to read and only target niche segments of the market, and, finally, (5) Programmers automatically reject VPLs, due to being used to the complexity of textual programming languages and enjoying the challenge of working with that complexity. Even though the first three issues

mentioned create big challenges for IoT developers, both technical and non-technical, the last two are not big problems for non-technical users, since they are using VPLs only for the IoT segment and are not programmers, so they are not in that same mindset mentioned on the last issue.

The biggest problem with VPLs for non-technical end-users is that they are forced to use a computer to work with interfaces that are too complex to understand and work with, which makes it hard for them to set up their systems.

To tackle some problems of Visual Programming, there are conversational interfaces and smart assistants that allow for an easier interaction with IoT systems.

2.3 Conversational Interfaces and Smart Assistants

This section introduces conversational interfaces and smart assistants, as well as their application in IoT as a means of interaction between end-users and IoT systems.

2.3.1 Conversational Interfaces and Smart Assistants Background

Similar to Visual Programming, conversational interfaces and smart assistants also aim to ease the interaction between end-users and IoT systems. However, the approach is very different from Visual Programming. Conversational interfaces and smart assistants enable users to interact with IoT systems in a conversation format, which means that users can talk to these interfaces and assistants and get responses and actions related to their commands, which makes interaction with IoT systems more natural for most users.

These tools take advantage of technologies like Speech Recognition (SR), Natural Language Processing (NLP) and Text-to-Speech (TTS) to hear, understand and respond to people's voice commands. SR consists on converting voice commands into text strings that express what was said by the user. This technology has been around for a very long time and was made popular in the 1960s and 1970s being used in some movies, such as "2001: A Space Odyssey" and the Start Wars saga [32]. With NLP, computers can extract information from the users text strings, gaining knowledge regarding the user's speech. NLP systems analyze sentences beginning at word level, in which they analyze the word's own meaning, then at sentence level to understand the order of the words, their relation and the meaning of the whole sentence, and finally on a context level, to understand if the meaning of the words changes according to the context in which they are said [10]. Finally, TTS is the technology that converts the computer's response to text and then to natural speech, allowing computers to communicate in the same way humans do.

2.3.2 Conversational Interfaces and Smart Assistants in IoT

Conversational interfaces and smart assistants have been applied in many fields, from providing customer service, in the early days, to helping manage a whole smart home, nowadays, and their evolution has been both regarding the understanding of human speech, but also regarding their capabilities to respond to commands.

Figure 2.5: Google Home products

As mentioned above, some giant companies have been working on these assistants, with the most popular being Amazon's Alexa[18], Google's Google Assistant[19] and Apple's Siri[20]. These assistants are used in IoT accompanied by hub devices, respectively Amazon's Echo[21], Google's Google Home[22] and Apple's HomePod[23]. Figure 2.5 shows some Google Home products, which are hubs, as mentioned above. Some other hub devices also support the integration of some of these assistants, even though they are from other brands, such as the aforementioned Ecobee4 thermostat.

The current most popular smart assistants are capable of satisfying many user commands, such as retrieving information about a specific topic, playing a requested song, interacting with smart homes, among others, and, in the field of smart homes, are capable of performing various tasks, such as interacting with lights, thermostats, security systems, etc. Most of these assistants also integrate with many brands of smart devices, allowing users to have a wide variety of devices managed by the assistants.

Even though these tools solve some problems in the IoT field, such as some of the ones introduced by VPLs, by making user interaction much simpler, they also have some limitations. For instance, these tools are not ready to handle a lot of different instructions, since they have to store a large amount of instructions and may confuse some instructions and act erratically. Some assistants have visual interfaces that allow users to interact with them, such as Alexa's smartphone app shown in Figure 2.6 (p. 13), but that, besides bringing the problems of Visual Programming, also makes users grab another device to interact with, which may cause them to start replacing the assistants with smartphones or computers [17]. Besides these problems, smart assistants also have the problem of not being able to manage complex IoT rules, such recurrent rules, since they do not have the complexity of systems like Node-RED, which are designed for this type of tasks [34].

[18] https://developer.amazon.com/en-US/alexa
[19] https://assistant.google.com/
[20] https://www.apple.com/siri/
[21] https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011
[22] https://assistant.google.com/platforms/speakers/
[23] https://www.apple.com/homepod/

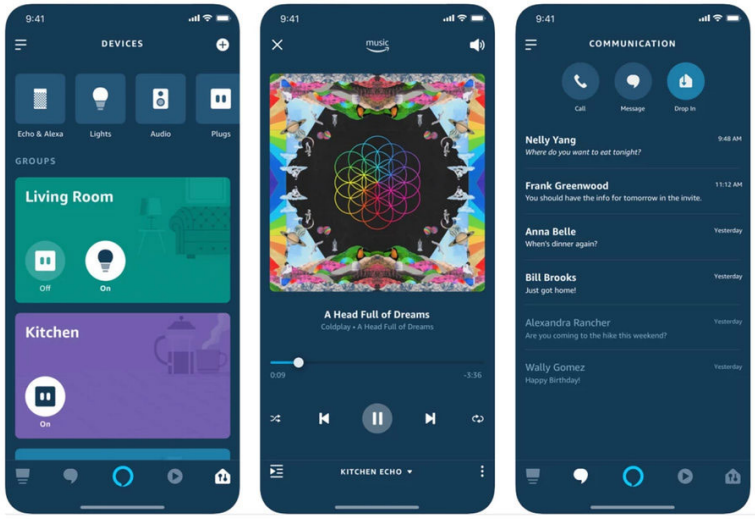

Figure 2.6: Alexa's smartphone app

2.4 Model Transformations

In this section, model transformations are explained, providing background information mainly related to this project, and how this concept can be applied in IoT.

2.4.1 Model Transformations Background

In Model-Driven Engineering (MDE), a model is defined as an abstract representation of a part of a system, being a useful tool to understand said system and understand how to work on it [40, 21, 23, 22, 20]. Diagrams that represent parts of a system (e.g. UML) and source code are examples of models. *Model transformations* are a method to generate a model, using another model and a set of transformation rules, resulting in a model that represents the same system, but in a different format [14]. Figure 2.7 shows a diagram portraying how a common model transformation works, from the source model to the target, with a model that represents the transformation.

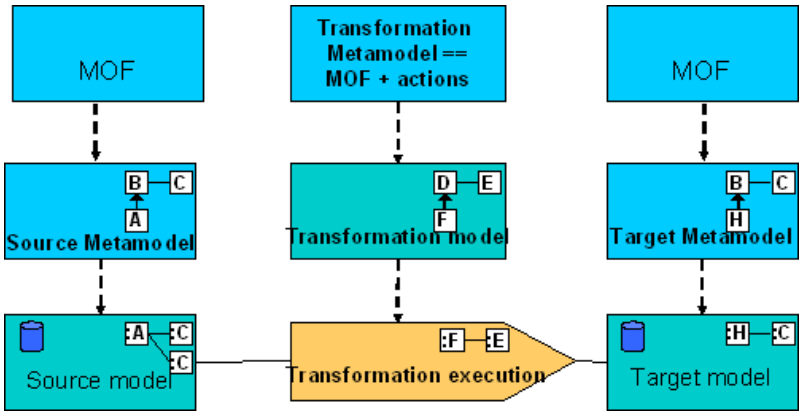

Figure 2.7: Model transformation

In 2007, the Object Management Group specified a standard set of languages for model transformations, the Query/View/Transformation. This standard is not only useful for defining Platform-Independent Models (PIM) and Platform-Specific Models (PSM), but also for defining model transformations, by helping in the conversion of PIMs to PSMs and in having synchronization between both models [13].

Understanding the concept of model transformations, one can understand that this concept can be used to leverage solutions that are already working and integrating them with new solutions, by applying transformations between models that represent each solution. This allows users to still use the solutions that they were already using, leveraging their advantages, and also use the new solutions, which should provide some new advantages and solve issues related to the solutions already available.

Throughout the years, some authors have developed classifications for model transformations, categorizing the transformations according to some of their characteristics. Mens et al. [40] created a taxonomy with a list of criteria, from which the most relevant for this work are:

- **Syntactic and semantic correctness** - Given a well-formed source model, the target model is well-formed (syntactic) and has the same semantic properties as the source (semantic).
- **Syntactic completeness** - For each element in the source model, there is an equivalent in the target model, that can be created by the transformation.
- **Directionality** - Transformations can be unidirectional, if they transform the source into the target, or bidirectional, if they also transform the target back into the source.

Besides these characteristics, it is also relevant to understand the mathematic isomorphism property, which states that a morphism f:X $\rightarrow$ Y is a isomorphism if there is a morphism g:Y $\rightarrow$ X such that $f \circ g = id_Y$ and $g \circ f = id_X$ [61]. This property means that if there are two models s and t such that s transforms into t and t transforms into s, then the transformation is isomorphic. This concept is important when applied to bidirectional transformations, because if the transformations are not isomorphic, it can mean that some representations with one model cannot be transformed into the other, or there may be ambiguity when transforming. If isomorphism cannot be assured, the bidirectionality may not be attainable.

2.4.2 Model Transformations in IoT

In IoT, model transformations can be used to generate platform-specific code from visual platform-independent models, as VPP allow users to define a high-level abstract model of a system, which is then converted by the tools into lower-level code that is applied on the devices [15]. One popular example of this application of model transformations is ThingML [25].

As mentioned above, model transformations can be used to integrate new solutions with solutions that are already commonly used in a specific field. Looking into IoT, there are already solutions being largely used, such as VPP and smart assistants, as mentioned in (*cf.* Section 2.2, p. 9) and (*cf.* Section 2.3, p. 11). However, these solutions face some issues separately that could perhaps

be mitigated by leveraging both solutions. Taking advantage of VPP as management tools for IoT and smart assistants as an interface for users to specify rules or actions. Having a representation for a rule in the interface and one for the rule in the VPP allows users to transform between the two and leverage the advantages of each format as they may see fit.

Despite being a promising approach to IoT development, model transformations still face some challenges, such as model quality measurement or model validation and verification [58]. Besides, as these transformations in the IoT field are usually based on Visual Programming for the source models, the challenges of Visual Programming are also present.

2.5 Summary

As IoT evolved and became more popular, it became a powerful technology for many fields. However, for inexperienced users, the tools available to work with IoT systems are not very familiar nor intuitive, which limits what users can do with their systems.

Even though VPP like Node-RED are able to handle complex IoT systems and can fully manage them, they lack in terms of user experience, being too overwhelming for inexperienced users. These tools also lack in debugging features, making it difficult to understand why and where a system is failing.

Smart assistants make it easier for inexperienced users to interact with IoT systems, allowing them to use voice commands in natural language, which is familiar to them. These assistants tackle some issues of Visual Programming, however also introduce some issues, such as their simplicity, which does not allow users to create complex rules for their systems.

The introduction of Model-Driven Engineering into IoT may be useful to improve some of the issues that IoT management faces, by allowing the re-use of current solutions that are working properly and introducing alternative solutions that can be integrated with the current ones, through model transformations between the representations of both solutions.

In conclusion, IoT lacks in easy to use management tools that can handle high complexity, which acts as a blocker for many people, preventing them from creating their own IoT systems.

Chapter 3

State of the Art

This chapter describes the state of the art for the context of this project, aiming to understand the current solutions for the problem or similar problems and also understand where they are lacking and where there can be improvements. In Section 3.1, we analyze some tools and research done in regard to interactions with smart spaces. In Section 3.2, we do the same, but diving into the use of model transformations in IoT. In Section 3.4, we summarize the contents of the chapter.

3.1 Interaction with Smart Spaces

Among the current smart assistants, the most popular and complete are the aforementioned Alexa, Google Assistant and Siri, which are widely used in smart homes to help users interact with smart devices. However, as has been stated in Section 2.3 (p. 11), these assistants fail to set complex rules for the IoT devices and also lack in terms of organization, becoming confusing if there are many similar rules set. Apple provides an application named Shortcuts[1] which allows users to create automation scenarios for their apps. For users that have their smart devices set up with their iPhones, it is also possible to use Shortcuts to automate interactions with those devices. This can be done using IFTTT recipes, in which users can use information from other applications (*e.g.* use *Maps* for location purposes) as triggers that cause an action on a device [5]. Users need to use the app Shortcuts to create these IFTTT recipes, but after that, the recipes run automatically and they can ask Siri to run them at any time, which allows them to have some complexity in their IoT systems and interact with them in a conversational basis. However, the rules need to be manually created on the app.

[1] https://support.apple.com/guide/shortcuts/welcome/ios

Priest et al. [45] elaborated a list of the commands that Alexa knows how to handle, classifying them in some categories, such as smart home, search, entertainment, among others. From this list, it is noticeable that Alexa cannot perform complex actions on smart home devices, instead, it can only perform direct actions like turning lights on or off, changing the lights colors, adjusting the temperature of the climate system, etc. Similar to [45], Martin et al. [39] gathered Google Assistant's commands in a list, categorized similarly to the one about Alexa. The results are also similar to the ones about Alexa. Regarding Apple's assistant, Siri, there is also a list of the commands it is able to perform, created by Langley [35], in which it is possible to see the similarities to the other two assistants mentioned. These assistants present a way to create routines (recurring rules), however, it has to be done manually using the mobile application associated to each assistant, instead of being done through voice commands, and the complexity of the routines is somewhat limited. To sum up, the current state of smart assistants does not allow users to create rules for their smart home devices using voice commands.

Clark et al. [11], in 2016, based on previous work that analyzed natural language for patterns in smart home programming, stated that the current smart assistants are too simple and work just as a voice interface for applications that control smart devices. The authors surveyed possible end-users of IoT systems for smart home applications they would want to see implemented, based on a list of smart devices and their capabilities. The survey was split in two, one where the smart home controls were handled by the devices controllers and one where there was an artificial intelligence (AI) agent receiving the user's commands and managing the controls, similar to a smart assistant, but without limitations set in terms of capabilities. The results of the surveys show a difference between both, with responses being better when there is an AI agent helping with the smart home management. The authors then analyzed the queries that were given by respondents and developed a grammar that could express all those queries, concluding that there were a lot of similarities in terms of sentence structure, which they believe allows them to convert any natural language command to a program that could be executed. This paper shows that current smart assistants are not developed enough to complete complex tasks and also that end-users are capable of creating natural language prompts that can be turned into smart home programs.

Like Clark et al. [11], other authors studied the applicability of natural language in smart homes, in a more complex fashion than current smart assistants can handle, concluding that end-users are able to create programmable rules through natural language, even if there is a basic structure attached to the rules, such as with IFTTT programming ("If this then that", also known as trigger-action). The structure of IFTTT can easily be understood by users, even if they are inexperienced, as shown by Ur et al. [57]. Figure 3.1 (p. 19) shows an example of an IFTTT recipe like the ones explored by these authors.

In 2017, Rani et al. [47] presented an approach to simplify user interaction with smart homes, based on voice commands, using NLP and AI, arguing that most of the current systems simulate basic electrical switch operations, only allowing users to turn devices on and off. The system they proposed functioned with voice commands given by the user, with the assistance of a smartphone application that interpreted the commands and sent them to the corresponding smart device. This

Figure 3.1: If-This-Then-That recipe example

article introduces the use of artificial intelligence to optimize the user interaction with the IoT system, however still lacks where the current smart assistants do, not allowing users to create rules and manage complex systems. Figure 3.2 shows the architecture of the system proposed by these authors, in which can be seen the application of voice commands and AI through a mobile application. Concluding, the system only allows users to toggle smart home appliances, using a mobile application to process the voice commands, but is not equipped to handle complex systems or even create rules for recurrent actions.

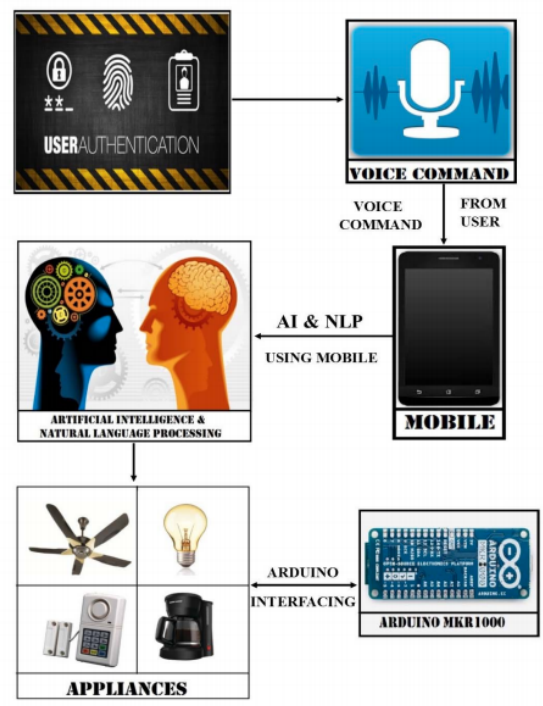

Figure 3.2: Architecture diagram of Rani et al. [47] tool

Also in 2017, Rajalakshmi et al. [46] presented a solution using both Node-RED and Alexa to interact with IoT systems, which they claim to simplify the interaction between users and IoT systems, but also manage complex systems. The system proposed by the authors uses Node-RED to create rules and link devices with each other, allowing the user to create complex rules, while at the same time taking advantage of Alexa, providing a simple way for users to control their smart devices. This system provides both the simple interaction with smart homes, through Alexa, and the complexity needed for some use cases, through Node-RED. However, there is no link

between the voice control and the Visual Programming Platform (VPP), so users need to set up rules in Node-RED that are not understood by Alexa, which means they cannot create these rules using the smart assistant. This condition brings the issues of Visual Programming, described in Section 2.2 (p. 9), such as the complexity of the user interface of these tools.

In 2018, Lago [34] worked on improving the interaction with smart spaces, by developing a conversational interface, named Jarvis, that could handle complex IoT systems and manage their rules, both simple and complex ones. The tool developed also allows users to inquire it to understand why some action occurred (e.g. "Why did the bathroom light turn on?"). The author claims that the system can be integrated with conversational interfaces that are already popular, such as Facebook's Messenger or Slack, or even with smart assistants like Google Assistant or Alexa, since the most popular smart assistants provide APIs for developers to integrate tools with those assistants. Figure 3.3 shows the architecture diagram of this tool, displaying the integration with other tools for user interface, natural language processing and back-end processing. Figure 3.4 shows an example interaction with the tool, in which the user asks the tool to perform some actions and the tool responds accordingly. The author tested the developed system with a set of complex rules, comparing to Google Assistant and Node-RED, to see whether the most popular tools to manage IoT systems were able to handle those rules, and concluded that Google Assistant could only handle 1 out of 10, Node-RED could handle 5 out of 10 and the developed tool could handle all 10. The system was also tested by possible users, mostly inexperienced ones, showing a high success rate. This shows an advance in the management of IoT systems, making it easier for users to interact with systems, even inexperienced users. However, the author points some issues with the system, such as the lack of flexibility in the Natural Language Processing imposed by DialogFlow, or the inability to show what the system can do.

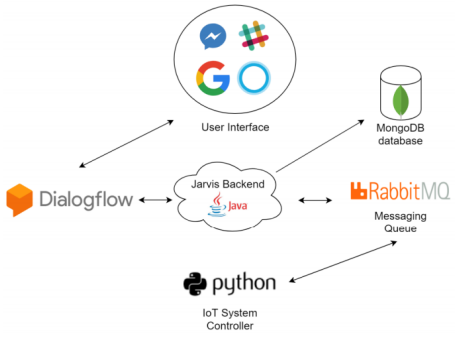

Figure 3.3: Architecture diagram of Jarvis [34]

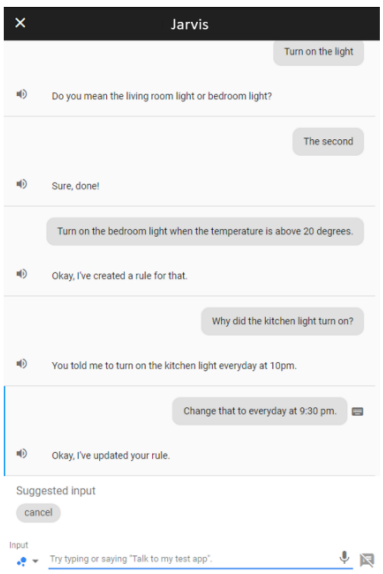

Figure 3.4: Example interaction with Jarvis [34]

3.2 Domain-Specific Languages in IoT

In IoT there have also been some authors studying the application of Model-Driven approaches like Domain-Specific Languages (DSL) and model transformations, to improve the interaction with IoT systems, as well as make the production of platform-specific code more efficient, by abstracting the platforms and allowing users to design systems at a high-level, with visual abstractions.

In 2015, Eterovic et al. [19] proposed a visual DSL for IoT, inspired in UML, aiming to provide non-technical end-users with a tool simple enough for them to be able to work on their IoT systems, but also sufficient complexity for more experienced users to do more complex tasks. UML was designed not only for technical users, but also for non-technical users to specify systems or parts of systems that they want implemented, therefore, it was built focused on simplicity for non-technical users. However, the authors state that when used in an IoT context, advanced UML expressions need to be used, making standard UML too complex for not so advanced users. With this in mind, the authors propose a DSL that looks like UML, but with some changes to simplify those advanced expressions. To validate the tool, the authors performed an experiment with 3 test groups, (1) two CS PhDs with UML experience, (2) three biomedical engineers with no UML experience and (3) seven first-year students with no UML experience. It is relevant that none of the participants had previous IoT experience. The results show that everyone was able to complete the experiment with success and most participants scored the DSL with very high scores. This should mean that the DSL is both simple in regard to interaction and complete in regard to functionality. However, the authors say that the measurements made do not show usability problems, only success rate, and also that the tasks in the experiment were very simple. Therefore, they believe that more experiments should be performed to measure other metrics and using more complex systems and tasks.

Also in 2015, Nguyen et al. [42] developed a framework named FRASAD, which applies Model-Driven Development into IoT systems management, with the goal of simplifying the interaction with complex systems. These authors took the approach of using model transformations to allow users to program with a VPP and convert that into code. The project uses a DSL and a rule-based model to describe the final applications and provides a GUI for users to create those rules, according to the DSL. Figure 3.5 (p. 22) shows an example of a Platform-Independent Model in the visual interface provided by FRASAD, and Figure 3.6 (p. 22) shows an overview of the model transformation and code generation logic that is applied. The authors evaluated the tool by testing it with novice and intermediate programmers, through the completion of tasks with various difficulties using the tool developed and two other tools (TinyOS and ContikiOS), and also by testing the model transformations, to see whether the models generated from the VPP were complete and correct enough to work properly. The results of the test with programmers showed that programmers were more successful using FRASAD than using the other two tools, regardless of their experience. The results of testing the model transformations showed that the tool generates complete and correct code in 40% of cases, however, in 41% of cases the code needed minor fixes to work and in 18% of cases the code was not complete nor correct. The authors conclusion is that the results prove that their tool can be a promising solution for IoT development and suggest that it

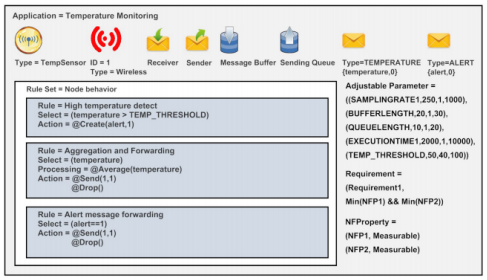

Figure 3.5: FRASAD PIM example

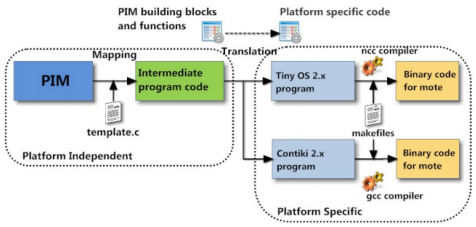

Figure 3.6: FRASAD model transformation and code generation overview

can be improved by extending the programming model to work with more operating systems and more IoT applications.

In 2017, Einarsson et al. [18] also applied model transformations in the field of IoT, developing a Domain-Specific Modeling Language (DSML), that allows users to specify rules which are then used to generate code for Alexa and SmartThings. This was developed by the authors, using Visual Studio Modeling SDK, and uses Text Template Transformation Toolkit (T4) to handle the transformations into platform-specific code. The authors tested the system in a mock cloud service, which allowed them to conclude that it can handle all Alexa's skills and some of SmarthThings', being only limited by the fact that they were only generating code to work with mock cloud-enabled devices.

3.3 Domain-Specific Languages in Other Domains

Outside of IoT, Domain-Specific Languages have also been explored, and it is worth expanding our research to other domains, in order to better understand them and how they can be applied in other fields.

In 2019, Rodríguez-Gil et al. [49] worked on a Visual Domain-Specific Language for educational applications that use intelligent tutors and conversational agents. As most teachers are not experienced with programming, they need a non-programming approach to implement these agents. Therefore, the authors developed a tool that uses a Visual DSL based on Google Blockly that integrates with current web-based educational platforms, in order to leverage the existing tools that teachers and students are used to. Figure 3.7 (p. 23) shows the main view of the tool, with some examples of actions for the conversational agents. These examples show the use of a *condition-action* structure for the actions, with a "WHEN condition DO action" format, which is very simplistic and natural for users. The authors had a group of non-programming university students testing the tool. First, the participants were shown some interactions with an example conversational agent, and then given a tutorial on how to create one using the authors' tool. After that, the participants were asked to create a conversational agent based on what they had learned, assisted by a cheat sheet with examples of actions defined with the tool. The authors concluded that

non-technical users were easily able to use the tool and create working conversational agents. The authors believed that the approach that they took was successful and seemed promising to be more explored and improved.

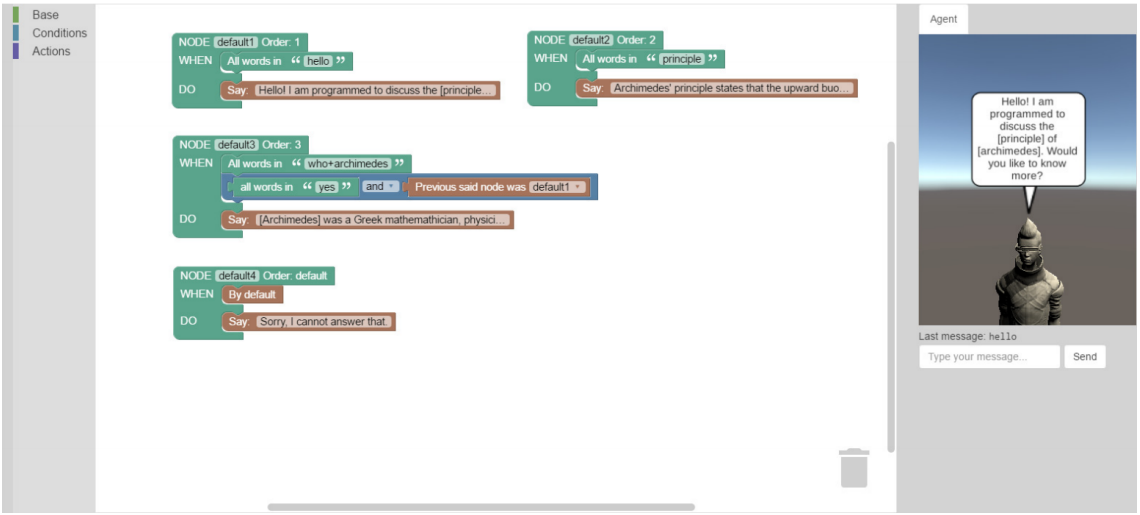

Figure 3.7: Main view of the platform developed by Rodríguez-Gil et al. [49].

Also in 2019, Sutherland et al. [55] developed a text-based DSL for programming social interactions for robots. Instead of build a complete new tool, they decided to leverage the existing tools and build a DSL that could be integrated with those tools. By doing so, they maintain the capabilities of the existing tools and build a simpler interface for non-technical users to be able to also work on programming robots. These authors developed a Python inspired DSL, with which users can create scripts for robots social interactions that are parsed and converted into an Abstract Syntax Tree (AST) to be validated and then can be consumed by an execution engine that runs on the robots, which results in the robots performing the scripted action. The authors concluded that the DSL is complete enough to be used for various robots with slight changes and also short enough to be quickly understood and used as a cheat sheet. However, the tool was not tested by non-technical users, and the authors mention some potential issues. For example, even though the DSL is simple and short, it looks a lot like a programming language, and non-programming users may have difficulties understanding some concepts or structural things. The authors also mention that improving debugging features may be helpful for non-technical users, since the debugging so far is done by reading logs resulting from the execution.

3.4 Summary

The interaction with IoT systems is a field that is being very studied, with the goal to simplify the way users interact with their systems. There have been advances both in smart assistants, which already allow users to interact with their smart spaces with direct actions such as “Turn the A/C on.”, and in VPP, which allow users to create more complex actions for their systems to perform. However, both approaches have some disadvantages, with smart assistants having difficulties in

handling complex systems and complex rules, and VPP being too complex for users to interact with them.

Some authors present solutions that may be applied in this field to improve some of the disadvantages aforementioned, by using DSLs to allow users to program rules for their systems in a simpler interface and then the rules are converted into platform-specific code, through model transformations. However, these solutions do not take advantage of conversational features, which poses as a big disadvantage, due to the simplicity and ease of use that they provide. In fields other than IoT, there are also solutions that show good results by leveraging DSLs and model transformations to simplify user interactions with different systems.

Chapter 4

Problem Statement

This chapter thoroughly describes the problem. Section 4.1 details the current issues in the area of focus. In Section 4.2 the proposed solution to solve these issues is explained. Section 4.3 lists the features of the proposed solution. Section 4.4 presents the main hypothesis to be validated. In Section 4.5 the experimental methodology is described. Finally, Section 4.6 summarizes this chapter, reviewing what was mentioned.

4.1 Current Issues

Throughout Chapter 2 (p. 5) and Chapter 3 (p. 17), the current solutions for IoT development were analyzed and their advantages and disadvantages were detailed. After the review of the solutions aforementioned, the following limitations were encountered:

- **Inability to manage complex systems** - the simplest means of interaction with IoT systems that provide the easiest usability (smart assistants) are not able to manage complex systems, using rules that depict complex scenarios with multiple devices and conditions.
- **Hard to use tools** - the tools that allow users to manage more complex and complete IoT systems are too complex for most users to use them.

To make these limitations more clear, we present a use case that portrays these limitations:

> *Whenever I get home before 11pm and it is dark outside, I want the entrance lights to turn on.*

The rule specified in this example is not handled by the popular smart assistants, since they do not support recurring actions or with this type of conditions. It is also too complex for most users to implement with a VPP, such as Node-RED, because the rule has multiple conditions as triggers, which can be overwhelming for most users.

The aforementioned limitations lead to the belief that it is possible to improve the current state of the art in the context of this dissertation.

4.2 Proposed Solution

The goal of this project is to understand if solving the issues detailed in Section 4.1 (p. 25) improves the capabilities of users to manage complex IoT systems, by developing a solution that is able to take the best features of visual programming and smart assistants. This solution consists of an interface that allows interactions in a semi-structured language to specify scenarios for an IoT system and provides the ability to convert the specifications to a visual representation (and the inverse operation as well).

The proposed solution uses a semi-structured language based on Gherkin, which is used to specify behaviour tests in BDD, due to the similarity of specification logic. As for the visual representation of systems, Node-RED is used, due to its high popularity and presence in the area of study. Both parts of the solution (textual and visual) represent models of the scenarios to be implemented. The conversion between both representations is based on MDE, using model transformations. Based on the classification presented in Section 2.4 (p. 13), the transformations should be bidirectional, allowing users to go from a textual model to a visual one and from a visual model to a textual one. However, the transformations are also not isomorphic, because it is very unlikely that the converting from a textual model X to a visual model Y and then back from visual to textual will result in having the same exact X, due to the ambiguity present in programming languages and even more in natural language. This may affect the bidirectionality, since, as mentioned in Section 2.4 (p. 13), when isomorphism cannot be ensured, the bidirectionality may not be attainable. The transformations should be both syntactic and semantic correct and complete, to ensure the proper scenario specification on both representations.

As mentioned in Chapter 3 (p. 17), Lago [34] developed a solution that was capable of managing complex IoT systems with an easy-to-use interface. However, that solution worked with a closed backend, which did not allow end-users to understand the structure of their systems, which can be critical when the systems start to scale in devices or rules. In this project, we will focus on the semi-structured language for the specification of IoT scenarios, the integration with Node-RED and the model transformations between both formats. The semi-structured language provides an easy-to-use, but complete, interface for users, while Node-RED provides the support for complexity and the visible structure for the system's rules.

The described solution should tackle the issues aforementioned, by providing a semi-structured interface that is easy to use and integrates with Node-RED, taking advantage of its ability to manage complex systems and rules.

4.3 Desiderata

This section describes the main features intended for the proposed solution, in order to solve the issues identified in Section 4.1 (p. 25). For the sake of completion, the solution must include features that are already present in other tools, otherwise the solution would not include basic IoT management features. The main features are the following:

- **D1: One time scheduled actions** - the user must be able to specify actions that are to be performed only once, but at a scheduled time, such as "Turn on the bedroom light at 7pm.".
- **D2: Recurring actions** - the user must be able to specify recurring actions that are to be repeated on a schedule, such as "Turn on the bedroom light every day at 7pm.".
- **D3: Actions with multiple conditions** - the user must be able to specify actions that have multiple triggers, such as "Turn the on the entrance lights when I get home before 11pm and it is dark outside.".
- **D4: Specify actions using the semi-structured interface** - the user must be able to specify the D1, D2 and D3 actions using the semi-structured interface.
- **D5: Specify actions using the visual programming tool** - the user must be able to specify the D1, D2 and D3 actions using the visual programming tool.
- **D6: Convert specifications between textual and visual models** - the user must be able to convert the specifications created from textual models to visual models and vice-versa.

4.4 Hypothesis

The main problem with IoT systems described in this dissertation is related to interaction between users and these systems. Therefore, the main hypothesis of this project is:

> *The integration of a semi-structured text-based interface and visual programming improves users capabilities to manage complex IoT systems.*

In this hypothesis, "integration of a semi-structured text-based interface and visual programming" refers to the use of a semi-structured text-based interface and a visual programming approach to interact with an IoT system, allowing users to manage the system using both approaches. Also, "improves users capabilities to manage complex systems" means that users can manage complex IoT systems with more success and fewer errors while implementing rules, in comparison to the existing tools that allow users to do the same tasks. And, finally, "manage complex IoT systems", in the context of this dissertation, refers to the specification of behaviours (rules) for the systems, discarding other aspects related to IoT management, for example, devices installation.

Besides the main hypothesis, there are other relevant questions that the proposed solution can help answer:

- **RQ1: Can users manage complex IoT systems using this semi-structured interface?** So far, the literature shows that it is possible to manage complex IoT system using a visual programming approach, but there are still difficulties in doing so with text-based approaches that need to be understood.
- **RQ2: Is there a pattern in the way users think of home automation scenarios?** As users tend to think in their natural language, there may be a structural pattern that they to follow when describing home automation scenarios textually.
- **RQ3: Do users understand visual programming diagrams more easily than rules specified with this semi-structured language?** Even though users think in their natural language, some things may be more easily understood with a diagram than in text.

4.5 Experimental Methodology

In order to evaluate whether the proposed solution solves the mentioned problems and achieves the detailed features, we follow a two-fold methodology:

1. The system is tested with scenarios created by potential users, to understand how complete it is, and whether it can actually manage complex scenarios, as it should. Those scenarios are collected through a survey, sent to users with different levels of expertise in technology and IoT, to gather a wide variety of scenarios.
2. We perform a case study with potential users to compare how well they understand our semi-structured language and Node-RED flows. This should allow us to understand if a semi-structured interface like ours can improve users capabilities to manage complex IoT systems, or if it introduces more complexity, making it even harder for them.

The results of both tests should prove whether or not the hypothesis and the research questions were satisfied and if the desiderata was achieved.

4.6 Summary

IoT management faces issues, mainly when it comes to novice or non-technical users, due to the lack of tools that provide simple interactions along capabilities to manage complex systems.

In order to tackle these issues, this project focuses on providing simple interactions for IoT users, while retaining the complexity that Visual Programming tools like Node-RED provide. The proposed solution provides a semi-structured language for users to specify textual commands and the ability to convert those commands to Node-RED flows, which can be manipulated and converted back to the semi-structured language. The conversions are bidirectional transformations, which means that there is a model representation of the semi-structured language commands and a model representation of Node-RED flows and it is possible to convert between both models.

To validate the solution, we test it with scenarios provided by potential users, to understand whether the system provides sufficient capabilities to handle complexity, and perform a case study to measure how well participants understand our semi-structured language in comparison to Node-RED flows.

Chapter 5

Solution

This chapter goes over the details of the solution. In Section 5.1, we briefly describe the solution, explaining its goal and overall behaviour. In Section 5.2, Section 5.3, Section 5.4, Section 5.5 and Section 5.6 we thoroughly explain the implementation details regarding every component of the solution. Finally, Section 5.7 summarizes the contents of this chapter.

5.1 Overview

The main goal of this project is to research **if** semi-structured text-based interfaces allow end-users to manage complex IoT systems without having to use tools that are too overwhelming for them, as described in Chapter 4 (p. 25). This goal is targeted by combining visual programming with a semi-structured interface, allowing users to use both the VPP and the semi-structured interface for management purposes.

Considering the research done around IoT and interactions with smart spaces, we decided to use Node-RED due to its presence in the field, and develop an interface (text-focused) based on a semi-structured language, very close to natural language, similar to Gherkin's application for Behaviour Driven Development (BDD). We named the solution SIGNORE: Semi-structured Interface in Gherkin for NOde-REd. The combination of the two approaches allows users to create rules for their systems in a textual way, instead of having to use Node-RED's interface.

Figure 5.1 (p. 32) shows the architecture of the system, detailing the process that occurs when it is used. The semi-structured interface takes the users input and runs it through an interpreter, which

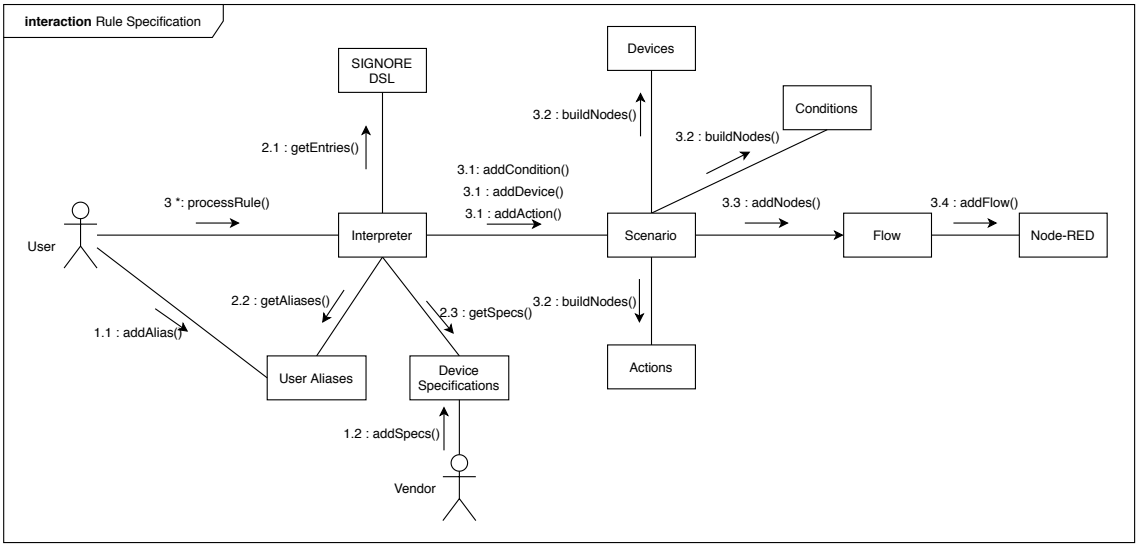

Figure 5.1: SIGNORE's architecture diagram

then builds a structure representing the input and creates a flow that is deployed to Node-RED. The process of building the structure from the input is similar to BDD, beginning with the creation of an object, the Scenario Structure, which is modified with each line that the interpreter processes.

The user input consists of a rule specification for home automation, in which the user states what actions the actuators must perform whenever some conditions are met. This rule specification is written in English, but following a certain structure. The users must begin by specifying which devices will be used and the MQTT topics to which they are subscribed or in which they publish information, so that SIGNORE can communicate with them. Then, the users write conditions that evaluate to true or false, based on the information provided by devices, or other information, such as time. Finally, the users detail the actions that should be performed by devices when the conjunction of the aforementioned conditions evaluates to true.

After receiving the whole specification for a rule, the Interpreter runs each line against the system's DSL and modifies the Scenario Structure object accordingly, by adding devices, conditions or actions. These three objects that can be added to the Scenario Structure correspond to the three parts of a rule specification. After processing the whole input, SIGNORE creates the appropriate flow for the devices, conditions and actions that were created, by understanding how the objects connect with each other. After the flow is complete, it is sent to Node-RED via their Admin API[1] and the user can interact with the flow using Node-RED's interface. Also, when the flow is deployed to Node-RED, the rule created starts to be applied by the IoT system, and its results will be seen when the rule's conditions are met.

[1] https://nodered.org/docs/api/admin/

5.2 Semi-Structured Language - SIGNORE's DSL

SIGNORE has two main components, the Interpreter and the Scenario Structure. The Interpreter is the component responsible for taking in the user input and transforming it into a complete Scenario Structure. To do so, the Interpreter uses a DSL that maps input phrases into functions that generate the Devices, Conditions or Actions to add to the Scenario Structure. Each DSL entry has an *expression* field, a *verify* field and a *do* field, which represent the phrases structure, the regular expressions for the variables in the phrase and the function that is related to the phrase, respectively. An example of these entries can be seen in Listing 1, and in Listing 2 the JavaScript function that corresponds to the DSL entry.

```
{
  expression: "given a $devConfig named $devName at $url",
  verify:
    {
      $devConfig: "([a-zA-Z0-9-_ ]+)",
      $devName: "([a-zA-Z0-9]+)",
      $url: "(/[A-Za-z0-9/]+)+",
    },
  do: "addDevice($scenario, $devConfig, $devName, $url)",
}
```

Listing 1: Example of an entry of the system's DSL, which represents a device instantiation

```
addDevice(scenario, devConfig, devName, url) {
    scenario.addDevice(devConfig, devName, url)
}
```

Listing 2: Function `addDevice`, that is related to the DSL entry to instantiate devices

The DSL has entries to create devices and actions, which are very simple and generic. The device creation depends only on the device specification, that is created by the device vendor, the device name given by the owner and the MQTT topic to which the device is subscribed or in which it publishes information. As for the actions, those depend on the actuator and the message that is sent to it.

The conditions, however, are on a different level of complexity. There are many types of conditions in a home automation system, such as device triggered conditions, time triggered conditions, among others, which means that there need to be different entries on the DSL, for the different types of conditions. The most common conditions are device triggered, for example when there is a motion sensor and it detects movement. However, to write specifications for these conditions, there are cases in which it may not be desirable to specify the device that is needed. The motion sensor example is one of those cases, where users can write "when motion_detected is true" instead of "when the $sensorName detects motion". This type of expressions makes use of variables from the input of devices, without the need to mention the device, as long as a device

with such capabilities exists. There are also conditions that are recurringly triggered, for example, on a daily or weekly basis, specifying the time and days for the condition. Time conditions may also include time intervals, to limit the time of the day for the conditions, for example, during the night, or between start and end times. The following list contains the types of conditions supported by SIGNORE, with the corresponding expression from the DSL entry and a usage example:

- **Device triggered:**
  - **DSL expression:** *"when $devName is $state"*
  - **Example:** *"when kitchenLight is on"*
- **Device triggered with timer:**
  - **DSL expression:** *"when $devName is not $state for $time$units"*
  - **Example:** *"when kitchenLight is not on for 10 minutes"*
- **Variable triggered:**
  - **DSL expression:** *"when $variable is $operator $value"*
  - **Example:** *"when lawn_height is higher than 10"*
  - **Note:** The "$operator" part is optional (*e.g.* "when motion_detected is true")
- **Recurringly triggered:**
  - **DSL expression:** *"when it is $day at $hours"*
  - **Example:** *"when it is monday at 7:00"*
  - **Note:** The "$day at" part is optional, and, if not present, represents "everyday"
- **Recurringly triggered in a time interval:**
  - **DSL expression:** *"when it is between $startHours and $endHours"*
  - **Example:** *"when it is between 7:00 and 10:00"*

With the provided DSL entries users have flexibility to implement home automation scenarios, with any type of device supported by SIGNORE, varied types of conditions and any actions supported by the devices. The DSL can also be extended by experienced users to support other entries that they want to, by adding the DSL entries and the corresponding JavaScript functions, as long as they use appropriate functions to change the Scenario Structure.

Listing 3 shows an example of how the semi-structured language developed can be used to create a home automation scenario. The first and second lines instantiate the devices used in this scenario, the third line is a condition and, finally, the last line is the action to be performed whenever the previous condition is met. The scenario represented is "when there is motion in the entrance,

```
given a motion sensor named entranceSensor at /sensors/entranceSensor
and a basic light named entranceLight at /lights/entranceLight
when motion_detected is true
then send message {on: true} to entranceLight
```

Listing 3: Example of a rule specifications with SIGNORE's semi-structured language, for scenario "when there is motion in the entrance, turn on the entrance light"

turn on the entrance light" and it uses a motion sensor to detect whether there is motion, and a light to be turned on. When the sensor detects motion, the variable "motion_detected" is set to "true" and the light receives a message to turn on.

With this tool, users can also create aliases for entries of SIGNORE's DSL, in order to create phrases that represent one or multiple DSL entries. This can be done, for example, to create a simpler phrase that represents multiple entries that are frequently used, in order to save time writing specifications, or to make the tool even simpler for users that are not technical at all. As mentioned above, there are some conditions that use specific variables from devices, and that is an example of how users can create aliases to simplify original entries of the DSL. For example, if a user has a sensor that detects rain, such as the one represented in Listing 4, the user can create scenario specifications using a condition "when rain_detected is true", which uses the variable `rain_detected` from the rain sensor. To simplify this condition and make it more user friendly, the user can create an alias for that expression that says "when it is raining". Another example for this feature is to specify defaults for example for the lawn size. If a user has a lawnmower such as the one in Listing 5, instead of having to write "when lawn_height higher than 10cm", the user can create an alias that says "when the lawn is tall", simplifying the expression and defining a default height for "tall lawn".

```
{
  type: "rain sensor",
  output: "JSON",
  input: "none",
  types: { on: "boolean", rain_detected: "boolean" },
}
```

Listing 4: Device Specification for a rain sensor

```
{
  type: "lawnmower",
  output: "JSON",
  input: "JSON",
  types: { on: "boolean", battery: "number", lawn_height: "number" },
}
```

Listing 5: Device Specification for a lawnmower

Listing 6 represents the user-defined alias for "tall lawn", and shows how the aliases are stored in SIGNORE. The *expression* field contains the alias that the user wants to use and the *equivalent* field contains an array with all the expressions that the alias represents from the original DSL entries. When parsing the user's input, the Interpreter replaces the alias expression with the original DSL entries, to be properly mapped into devices, conditions or actions.

```
{
  expression: "when the lawn is too tall",
  equivalent: ["when lawn_height is higher than 10"]
}
```

Listing 6: Alias condition for lawn height

5.3 "Examples" Feature in Rule Specifications

In Behaviour Driven Development, there is a concept of "examples" to avoid code duplication, by writing the common part of multiple behaviours and then specifying as "examples" the remaining part of each variation of the behaviour. SIGNORE also supports a similar concept, also named "examples", to allow users to create similar home automation scenarios without having to write the same specification multiple times.

In Section 5.2 (p. 33), we show an example of a rule specification using SIGNORE's semi-structured language, which purpose is to turn on a light when a motion sensor detects motion. Assuming that instead of one light, the user wants to turn on multiple lights whenever that condition was met, instead of having to write a rule specification for each light, the user can take advantage of the "examples" feature and write it only one, specifying each light as an "example". Listing 7 shows the application of this concept for the scenario described. The first four lines of the specification are very similar to the specification in Listing 3, with the difference being that instead of the name of the light, there is a variable "devName" which is defined in the "examples" section with the names of all the lights involved in the scenario. When the Interpreter parses this specification, it replaces the variable with the appropriate light names, to later modify the scenario structure and the Node-RED flow.

5.4 Device Specifications

Besides the SIGNORE's DSL, the Interpreter also makes use of a structure that contains Device Specifications built by the device vendors, that allow it to understand what type of devices are available and their capabilities. In this structure, each device is an entry, with a *type* field that identifies the product, one *output* and one *input* fields that represent the type of output and input the device uses and, finally, a *types* field that contains all the variables that can be used to specify actions for the devices, and their types. Listing 8 represents an example of a Device Specification

```
given a motion sensor named entranceSensor at /sensors/entranceSensor
and a basic light named $devName at /lights/$devName
when motion_detected is true
then send message {on: true} to $devName

examples:
   | $devName       |
   | entranceLight1 |
   | entranceLight2 |
   | entranceLight3 |
   | entranceLight4 |
```

Listing 7: Example of a rule specification for scenario "when there is motion in the entrance, turn on the entrance lights", using the "examples" feature of SIGNORE

for a *basic light*. With these two sources of information, the Interpreter can map the user input to a proper Scenario Structure and ensure that the user is using the proper devices, conditions and actions, to build a working rule for their devices.

The usage of vendor created device specifications allows SIGNORE to work with any type of device, as long as the vendor creates a specification with the aforementioned structure. Besides, a more experienced user can code virtual devices and create specifications for them. These virtual devices can be used, for instance, to act as a layer between the system and a real device, to add new features to the real device. For example, if a user has a smart lamp that only allows them to turn it on/off and set a brightness value, the user can create a virtual device that is capable of performing brightness increments or decrements, so that the user does not need to pick a value for the brightness. This example can be understood with Listing 8 and Listing 9, that represent the two devices mentioned, a *basic light* that is capable of being turned on/off and receiving a value for the brightness, and a *super light* that is capable of the same features as the *basic light*, but is also capable of receiving `briUp` or `briDown` actions that increment or decrement the brightness by an offset that was coded by the user.

```
{
  type: "basic light",
  output: "JSON",
  input: "JSON",
  types:
    {
      on: "boolean",
      bri: "number",
    },
}
```

Listing 8: Device Specification for *basic light*

```
{
  type: "super light",
  output: "JSON",
  input: "JSON",
  types:
    {
      on: "boolean",
      bri: "number",
      briUp: "boolean",
      briDown: "boolean",
    },
}
```

Listing 9: Device Specification for *super light*

5.5 Scenario Construction

As the Interpreter parses and processes each line of the rule specification given by the user, it stores the rule's information in a specific structure that represents the complete specification. This structure is the basis for the generation of the Node-RED flow, in the final step of the process.

In Figure 5.1 (p. 32), there is a component named Scenario Structure, which is the mentioned structure, that stores the information of the rule specification. As this object represents a home automation scenario and, as explained in Section 5.3 (p. 36), it is possible to specify multiple scenarios in the same rule specification, the Scenario Structure is a composition of Subflows, that represent each variation of the rule specification. The Scenario Structure is responsible for managing the Subflows, *i.e.* creating new Subflows, modifying the existing ones and deploying them, in the end. This means that the Interpreter does not interact with the Subflows directly, but only with the Scenario Structure, which then forwards the information to the appropriate Subflow.

The Subflow is the object that contains the Devices, Conditions and Actions for the scenarios described in the rule specifications. This object represents, as aforementioned, a specific variation of the rule specification, and is responsible only for that variation. The Subflow object stores its Devices, Conditions and Actions and is capable of understanding the connections between each of those. This means that the Subflow can understand if a condition depends on a device, or what device an action depends on, since those connections are stored in the Devices, Conditions and Actions objects.

In the following list, we describe the Device, Condition and Action objects, in order to clarify their roles in the Subflow:

- **Device:** The Device object is the representation of a specific device, containing the information that identifies the device in the network and the links between the device and other objects of the Subflow.
- **Condition:** The Condition object represents a condition variation from the conditions list shown in Section 5.2 (p. 33). This object contains the condition specifications (*i.e.* the

devices involved, the schedules, the time intervals, etc) and contains the links to other objects of the Subflow that have a relationship with the Condition.

- **Action:** The Action object represents an action to be performed by a device from the Subflow, and contains the message to be sent to the device, as well as the link to that device.

When the Interpreter receives a rule specification, if it contains the "examples" section, the rule specification is split into the multiple variations and the Interpreter processes one by one, letting the Scenario Structure know that there are multiple variations and, therefore, there must be multiple Subflows in the Scenario Structure. Otherwise, only one Subflow is needed. As the Interpreter processes each variation of the rule, it matches each line to an entry of SIGNORE's DSL, as explained in Section 5.2 (p. 33), and the function mapped by the matching entry tells the Scenario Structure to add a Device, a Condition or an Action, according to the purpose of the processed line.

After the whole rule specification is processed, the Scenario Structure object is completely built and ready to be deployed.

5.6 Flow Generation

The final step of SIGNORE's process for each rule specification is the generation of the Node-RED flow that represents the rule specification. This is done according to the information contained in the Scenario Structure.

The Device, Condition and Action objects contain all the information that is needed to create the Node-RED nodes that represent those objects and also the information of the links between the objects. This means that each object is capable of creating the proper nodes to represent itself and also the links to the objects to which it must be connected. Therefore, when a rule is completely processed and the Scenario Structure is ready to be deployed, each Subflow builds the flow that represents the rule variation, by getting the nodes and connections from the Devices, Conditions and Actions that it stores. At this point, each Subflow has an object that can be deployed to Node-RED as a working flow. However, as there may be multiple Subflows in one Scenario Structure, the Scenario Structure processes the flows that each Subflow built, in order to merge duplicate nodes, and properly merge all the flows into one.

To clarify the steps mentioned above, we provide an example of a rule specification and its effect on the Scenario Structure and the Subflow, as well as the flows that each Subflow would have and the final flow that is deployed. Considering the rule specification that was used in Section 5.3 (p. 36), we have:

When there is motion in the entrance, turn on the entrance lights.

This home automation scenario is represented in our semi-structured language by the specification in Listing 10.

When the Interpreter starts to process the specification, it splits the specification into four variations, one for each "entranceLight" and then works on the four, one by one. Considering the first variation, for "entranceLight1", the Interpreter tells the Scenario Structure to create one Subflow and tells it to add two Devices (one motion sensor and one light), one Condition (the motion detection) and one Action (send a message to the light to turn on). It does the same for the other three variations and, after that, it tells the Scenario Structure to get ready to be deployed. At this point, the Scenario Structure has all Subflows building their Node-RED flows, which are represented by the flow in Figure 5.2 (the difference between the flows of each Subflow is only the rightmost node, which represents each light). In the resulting flows, the nodes for the "entranceSensor" and the condition "motion_detected=true" are repeated in all variations, and can be merged into only one node of each. The nodes representing the Action ("on: true") for each light also look duplicated, but will not be merged into only one, because each Action is related to one specific Device, therefore each Action is different from the others.

```
given a motion sensor named entranceSensor at /sensors/entranceSensor
and a basic light named $devName at /lights/$devName
when motion_detected is true
then send message {on: true} to $devName

examples:
   | $devName       |
   | entranceLight1 |
   | entranceLight2 |
   | entranceLight3 |
   | entranceLight4 |
```

Listing 10: Rule specification for scenario "when there is motion in the entrance, turn on the entrance lights"

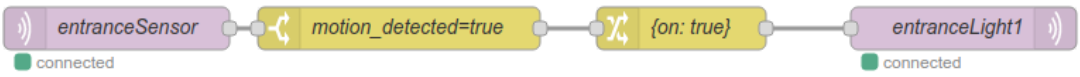

Figure 5.2: Node-RED flow for "when there is motion in the entrance, turn on entranceLight1" variation

After each Subflow builds its Node-RED flow, the Scenario Structure does the processing to merge duplicates and merges the four flows into one, which is represented in Figure 5.3 (p. 41). In this flow we can see that the nodes for the "entranceSensor" and the condition "motion_detected=true" are not repeated for the four lights, producing a smaller and cleaner final flow.

After the steps described are complete, the flow is deployed to Node-RED, via their Admin API, and is applied to the user's system. To deploy the final flow, we have a Communicator object that interacts with Node-RED's Admin API and constructs the flow's layout. When the flows are built by the Subflows and Scenario Structures, the nodes are not properly positioned for the flow to

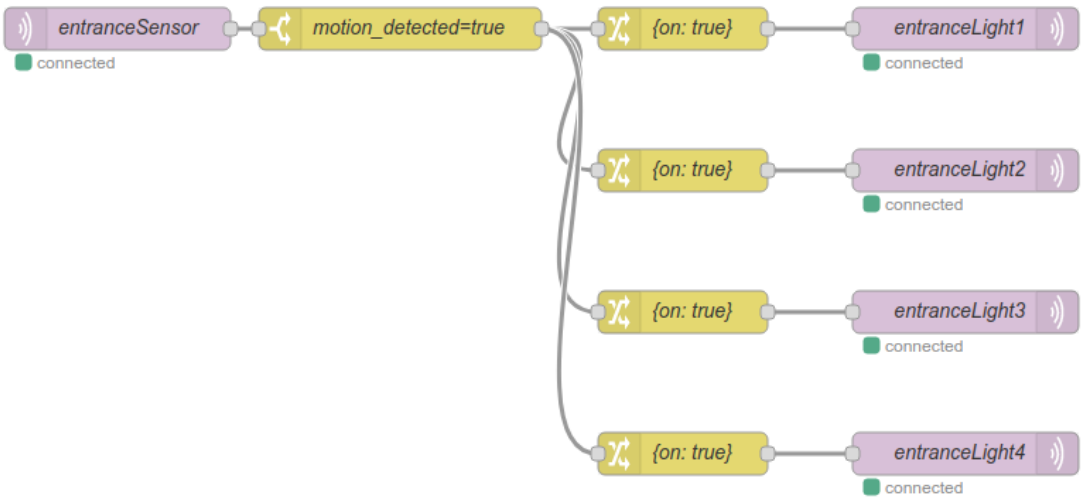

Figure 5.3: Node-RED flow for complete rule specification for home automation scenario “When there is motion in the entrance, turn on the entrance lights.”

be understandable, therefore, the Communicator positions the nodes according to their connections. After that, the Communicator uses the `POST /flow` method to inject the flow into Node-RED’s active flow configuration.

5.7 Summary

Considering the issues with IoT management and the features that we wanted to implement, described in Chapter 4 (p. 25), we developed a semi-structured interface that allows users to manage complex IoT systems using a textual format to specify rules for the systems. This solution was named SIGNORE: Semi-structured Interface in Gherkin for NOde-REd.

SIGNORE uses a semi-structured language to specify scenarios, similarly to the usage of Gherkin in BDD. This language maps text phrases to code functions that build objects representing the scenarios and after the object is complete, it is deployed to Node-RED. After this, the scenario is applied to the users system and can be interacted with using Node-RED.

The solution tackles the issues and provides the features described in Chapter 4 (p. 25), except for the transformations from Node-RED to SIGNORE. As mentioned in Section 4.2 (p. 26), there is ambiguity in programming languages and natural language, which causes the transformations not to be isomorphic, and, therefore, bidirectionality is potentially not attainable.

In Chapter 6 (p. 43) and Chapter 7 (p. 55) we describe the methodology that was followed to evaluate the tool.

Chapter 6

Home Automation Survey

This chapter describes the survey that was performed as a part of the validation for this project. First, Section 6.1 introduces the motivation for this survey and Section 6.2 describes the methodology that was followed to perform it. Section 6.3 contains the descriptive analysis and Section 6.4 contains the discussion regarding the results of the survey. In Section 6.5 we describe the threats to validity that were identified and, finally, in Section 6.6 we summarize the contents of this chapter.

6.1 Motivation

The main goal of this survey is to gather as many and as varied home automation scenarios as possible, from individuals with different backgrounds and technical experiences, to test whether SIGNORE supports only a few scenarios, or a wide range of them. To test SIGNORE, we want to have many scenarios to understand how many different types of scenarios participants submit and how many types are commonly implemented in smart homes. For this, we group the scenarios into categories, according to similarities in their structure and types of conditions used.

Additionally, we want to understand how people describe home automation scenarios using text, to understand if different individuals use very different phrases to describe the same scenarios. For that, we analyze the scenarios to find similarities between them.

6.2 Methodology

We asked 20 participants from different education fields and different ages to create some home automation scenarios, given a smart house and many smart devices.

To spike the participants' creativity, we designed a home using a tool that creates 2D and 3D models, which can be seen in Figure 6.1 and Figure 6.2 (p. 45). Along with the home model, we provide a list of smart devices, containing various types of sensors and actuators for the participants to use. The devices present on the list are shown in Appendix B (p. 75). Besides the devices present on the list, we allowed participants to include home automation scenarios using other devices, if needed, in order to not limit their creativity.

Figure 6.1: 2D plan of the smart house used for the survey

The resulting scenarios from the survey were grouped into categories and then we selected one scenario from each category to be implemented on SIGNORE. This allowed us to understand the flexibility provided by our system.

6.3 Analysis

The survey resulted in 177 scenarios submitted, which were split into categories, according to similarities in their structure and format, in order to understand how many types of scenarios there were. These scenarios are collected in Appendix A (p. 67).

Figure 6.2: 3D plan of the smart house used for the survey

The resulting categories were the following:

1. **Sensors and actuators:** scenarios that only use sensors and actuators, where the actuators are triggered by the sensors (*e.g.* "When there is movement in the garage, turn on the garage lights").

2. **Actuators on schedule:** scenarios where actuators are triggered on a fixed schedule (*e.g.* "When time is 7:30 am, turn on the coffee machine, the hot water system and the kitchen lights").

3. **Actuators on time interval, with sensors:** scenarios that combine sensors information and time intervals to trigger actuators (*e.g.* "During night, when there is motion in one room, light that room at 200 brightness").

4. **Sensors with timers:** scenarios where the actuators are triggered when the sensors status does not change for some time (*e.g.* "When there is no one in the pool for 10 minutes, cover the pool and turn off the water heater").

5. **Actuators with timers:** scenarios where the actuators are triggered for some time (*e.g.* "When it is 23:00, turn on the garden watering system for 10 minutes").

6. **External services:** scenarios that depend on external services to trigger the actuators (*e.g.* "When sun is expected during the following hours, turn off the heating system").

7. **One-time actions:** scenarios that are meant the happen only once, instead of being recurring (*e.g* "When it is 7:00 today, turn on the bedroom lights").

Table 6.1 details the distribution of scenarios through the categories, showing the absolute and relative frequency of submissions per category.

Category	**Absolute Frequency**	**Relative Frequency**
Sensors and actuators	103	0.58
Actuators on schedule	42	0.24
Actuators on time interval, with sensors	15	0.08
Sensors with timers	9	0.05
Actuators with timers	1	0.01
External services	5	0.03
One-time actions	2	0.01
Total	**177**	**1.00**

Table 6.1: Categories of submitted scenarios, along with the absolute and relative frequency of submissions for each category.

After collecting and analyzing the scenarios, we consider them all valid and fit for one category. However, many were not detailed enough to be implemented, *i.e.*, some participants submitted scenarios that were too abstract to be implemented without changes. For example, there were submissions such as *"Intensity of lights based on the available natural light"*, *"Blinds inclination system based on outside light"*, *"On schedule turn on the coffee machine"*. These submissions need to be changed into a more programmable format, to be implemented with both Node-RED and SIGNORE, to look more like *"When the luminosity in the living room is below $value, then increase the lights intensity by $increment"*, or *"When time is 7:00, then turn on the coffee machine"*. These scenarios are still valid, because the information portrayed is enough to understand their meaning and to which category they belong. With the scenarios collected, the house plan and devices that were provided to the participants, we created a **resulting ecosystem** that represents the house with all the devices that were used by the participants to create the scenarios. This ecosystem is represented in Figure 6.3 (p. 47), and it shows the house plan with all the devices available, as well as a weather API which use was mentioned in some scenarios, and a person with some wearables that were also mentioned in some scenarios.

Looking at the scenarios from Appendix A (p. 67), we can see that the most common pattern in the scenarios is the structure "when *condition*, then *action*" or "*action*, when *condition*", or the same but with "if" instead of "when". There are some scenarios that follow different patterns, mainly for scheduled actions, such as "*action* at *time*", or "everyday at *time*, *action*". As mentioned above, some of the scenarios are not implementable due to their abstractness, therefore, those do not follow any noticeable pattern.

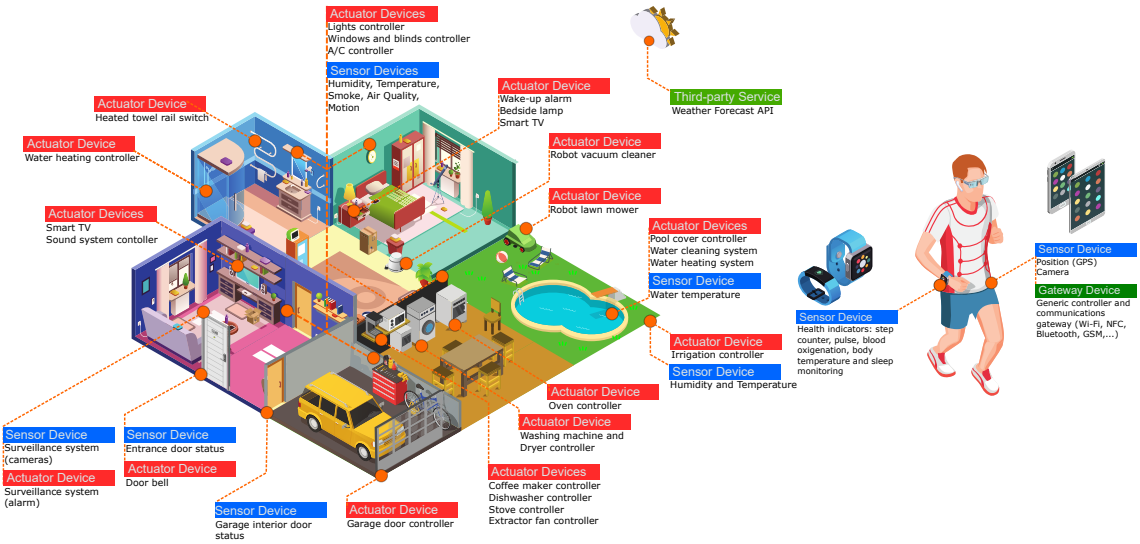

Figure 6.3: 3D plan of the ecosystem resulting from the survey

After categorizing the scenarios submitted by the participants, we picked one scenario from each category to understand whether it was implementable with SIGNORE. The scenarios tested also represent all variations described in Section 4.3 (p. 27). Below we show how the picked scenarios are represented with our semi-structured language and whether the system supports them or not.

6.3.1 Sensors and actuators

- **Scenario description:** When there is movement in the garage, turn on the garage lights
- **Scenario implementation:** Listing 11

```
given a basic light named GarageLight at /lights/GarageLight
and a motion sensor named GarageMotionSensor at /sensors/GarageMotionSensor
when motion_detected is true
then send message {on: true} to GarageLight
```

Listing 11: Implementation of scenario *"When there is movement in the garage, turn on the garage lights"*

- **Result:** Success
- **Resulting flow:** Figure 6.4

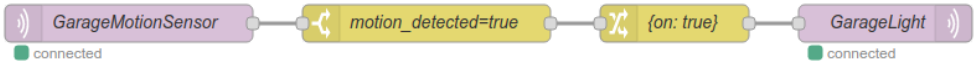

Figure 6.4: Resulting flow for scenario *"When there is movement in the garage, turn on the garage lights"*

6.3.2 Actuators on schedule

- **Scenario description:** When time is 7:30 am, turn on the coffee machine, the hot water system and the kitchen lights
- **Scenario implementation:** Listing 12

```
given a coffee machine named CoffeeMachine at /devices/kitchen/CoffeeMachine
and a hot water system named HotWaterSystem at /devices/HotWaterSystem
and a basic light named KitchenLight at /lights/kitchen/KitchenLight
when it is 7:30
then send message {on: true} to $devName

examples:
    | $devName |
    | CoffeeMachine |
    | HotWaterSystem |
    | KitchenLight |
```

Listing 12: Implementation of scenario *"When time is 7:30 am, turn on the coffee machine, the hot water system and the kitchen lights"*

- **Result:** Success
- **Resulting flow:** Figure 6.5

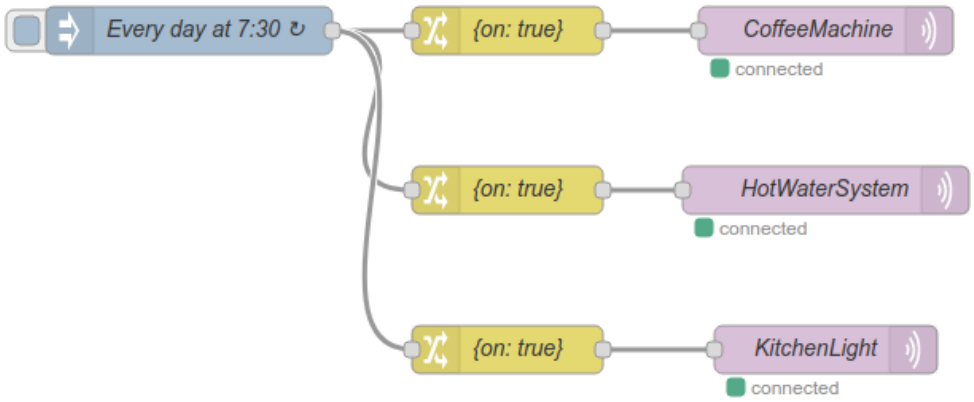

Figure 6.5: Resulting flow for scenario *"When time is 7:30 am, turn on the coffee machine, the hot water system and the kitchen lights"*

6.3.3 Actuators on time interval, with sensors

- **Scenario description:** During night, when there is motion in one room,light that room at 200 brightness
- **Scenario implementation:** Listing 13
- **Notes:** "when motion is detected" is an alias for "when motion_detected is true"

```
given a motion sensor named BedroomMotionSensor at /sensors/BedroomMotionSensor
and a basic light named BedroomLight at /lights/BedroomLight
when motion is detected
and it is between 23:00 and 7:00
then send message {on: true, bri: 200 } to BedroomLight
```

Listing 13: Implementation of scenario *"During night, when there is motion in one room,light that room at 200 brightness"*

- **Result:** Success
- **Resulting flow:** Figure 6.6

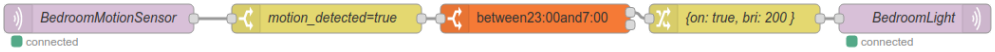

Figure 6.6: Resulting flow for scenario *"During night, when there is motion in one room,light that room at 200 brightness"*

6.3.4 Sensors with timers

- **Scenario description:** When there is no one in the pool for 10 minutes, cover the pool and turn off the water heater
- **Scenario implementation:** Listing 14

```
given a motion sensor named PoolMotionSensor at /sensors/pool/PoolMotionSensor
and a pool cover named PoolCover at /actuators/pool/PoolCover
and a hot water system named PoolHeater at /actuators/pool/PoolHeater
when motion_detected is not true for 10 minutes
then send message $msg to $devName

examples:
     | $msg | $devName |
     | close | PoolCover |
     | on: false | PoolHeater |
```

Listing 14: Implementation of scenario *"When there is no one in the pool for 10 minutes, cover the pool and turn off the water heater"*

- **Result:** Success
- **Resulting flow:** Figure 6.7 (p. 50)

6.3.5 Actuators with timers

- **Scenario description:** When it is 23:00, turn on the garden watering system for 10 minutes
- **Scenario implementation:** Listing 15

Figure 6.7: Resulting flow for scenario *"When there is no one in the pool for 10 minutes, cover the pool and turn off the water heater"*

```
given a watering system named WateringSystem at /actuators/garden/WateringSystem
when it is 23:00
then send message {on: true} for 10 minutes to WateringSystem
```

Listing 15: Implementation of scenario *"When it is 23:00, turn on the garden watering system for 10 minutes"*

- **Result:** Success
- **Resulting flow:** Figure 6.8

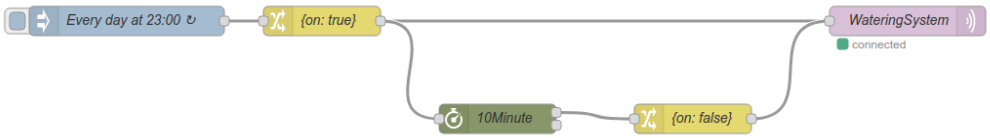

Figure 6.8: Resulting flow for scenario *"When it is 23:00, turn on the garden watering system for 10 minutes"*

6.3.6 External services

- **Scenario description:** When sun is expected during the following hours, turn off the heating system
- **Result:** Failure
- **Notes:** For the scope of this project, we did not consider external services as inputs and outputs for the scenarios, only MQTT, which means that, in its current state, SIGNORE does not support any scenario that depends on external services. However, the tool is scalable and it is possible to add this feature in the future.

6.3.7 One-time actions

- **Scenario description:** When it is 7:00 today, turn on the bedroom lights
- **Scenario implementation:** Listing 16
- **Notes:** This type of scenarios is not commonly implemented with Node-RED, due to its nature, because Node-RED is usually used for recurring scenarios, as the ones above.

```
given a basic light named BedroomLight at /lights/bl
when it is 7:00
then send message {on: true} to BedroomLight once
```

Listing 16: Implementation of scenario *"When it is 7:00 today, turn on the bedroom lights"*

- **Result:** Success
- **Resulting flow:** Figure 6.9

Figure 6.9: Resulting flow for scenario *"When it is 7:00 today, turn on the bedroom lights"*

The implementation attempts of scenarios from all categories show that SIGNORE is capable of handling almost all categories of scenarios submitted, except for scenarios that require external services. Considering the distribution of scenarios per category, shown in Table 6.1 (p. 46), it means that the tool supports around 97% of the scenarios submitted.

6.4 Discussion

The implementation of scenarios shows SIGNORE's success in supporting a wide variety of categories, but with this survey, we also conclude that, for the categories supported by SIGNORE, many participants followed a structure similar to the one that our tool uses, except for the first part, to specify the devices to be used. As aforementioned, the most common pattern found is a structure similar to "when *condition*, then *action*", or "*action*, when *condition*". This shows that it is intuitive for regular users to describe home automation scenarios in a format similar to SIGNORE's. Besides, as we provide the possibility of using aliases for expressions, users can shape the language structure to their preferences, with ease, which allows users to adapt SIGNORE to other patterns that they prefer.

Table 6.2 (p. 52) shows a comparison between SIGNORE, Node-RED and Google Assistant, in regard to the categories supported by each. This table shows that SIGNORE supports as many categories as Node-RED, except for the external services category. As we have explained above, this category is not supported, because for the scope of this project, we did not consider external services as inputs and outputs for the scenarios. Besides, comparing SIGNORE to the popular smart assistants (represented by Google Assistant), it is very clear that we provide a lot more flexibility in terms of scenarios supported, therefore, providing more capabilities for users to manage complex IoT systems.

Category	SIGNORE	Node-RED	Google Assistant
Sensors and actuators	•	•	.
Actuators on schedule	•	•	.
Actuators on time interval, with sensors	•	•	.
Sensors with timers	•	•	.
Actuators with timers	•	•	.
External services	.	•	.
One-time actions	•	•	•

Table 6.2: Comparison of the categories supported by SIGNORE, Node-RED and Google Assistant. The large dots represent success, while the small dots represent failure.

6.5 Threats to Validity

For this survey, we have identified some threats that may affect the validity of the results attained.

We asked participants for home automation scenarios and did not give them any **structure for the phrases**, to understand how they would write the scenarios, which resulted in many scenarios being just a brief description and not specific enough to be implemented. Perhaps, having requested the participants to provide more detailed scenarios would have resulted in more scenarios following a structure similar to our semi-structured language. However, asking participants to use a specific structure for the scenarios descriptions would not have allowed us to evaluate whether users tend to follow a pattern describing the scenarios or not.

The **sample size** for this survey was not very large, since there were only 20 participants. Having a larger sample could have resulted in more varied scenarios, and more scenarios that could or could not be implementable using SIGNORE. This could have changed the success rate that SIGNORE has shown in supporting the scenarios collected.

The **level of expertise** of the participants could impact the scenarios provided by them. For example, participants with more experience with IoT should provided more complex and realistic scenarios than participants with no experience in that field. To tackle this threat, we chose participants with different levels of experience with Node-RED and IoT, collecting scenarios from participants whose experience ranged from never having thought of a home automation scenario, to participants that had already implemented IoT systems and worked with Node-RED extensively.

Even though the participants had different levels of expertise in the field of home automation, and there were some participants with a lot of experience, the results show **little variety in categories** for the scenarios. After collecting 177 scenarios, we only identified 7 categories, and 82% of the submissions belonged to only one category. Maybe having a larger number of participants would have resulted in more varied scenarios, and more categories.

6.6 Summary

In this chapter, we presented a survey with 20 participants, to collect home automation scenarios, which resulted in 177 scenarios to be analyzed. The resulting scenarios were split into categories according to their structure, in order to test how many categories SIGNORE could support. We picked one scenario from each category to be implemented. SIGNORE supported all categories, except for one, which is justified by the fact that this category was not planned for the scope of this project. However, in comparison with other tools, our solution is very close to Node-RED in regard to categories supported, and clearly outperforms the current smart assistants, such as Google Assistant.

We conclude that SIGNORE is capable of managing complex home automation scenarios, providing a semi-structured interface that supports a wide variety of those scenarios, and a semi-structured language that can be shaped by the user, to fit their preferences. Also considering the threats to validity identified, we do not believe that the results would be a lot different after solving those threats. Therefore, we believe this project shows progress in regard to IoT management using textual interfaces.

Chapter 7

Case Study

This chapter details the case study that was done for the validation of this project. Section 7.1 explains why this case study was done and what was expected from it. Section 7.2 describes the methodology followed to perform the study. In Section 7.3 we analyze the data from the study and in Section 7.4 we discuss the results from that analysis. In Section 7.5 we present the threats to validity that were identified and in Section 7.6 we summarize the contents of this chapter.

7.1 Motivation

To understand if participants can interpret and describe home automation scenarios represented with our solution more easily than when represented with Node-RED flows, we performed a case study, conducted with a questionnaire. We presented some scenarios in the flow format and some scenarios in the textual format, and asked the participants to explain them. After that, we asked them to provide feedback regarding their preferred format to understand the scenarios and which format they would prefer to implement scenarios with. The participants were also asked for their level of experience with Node-RED or other VPL, so that we could distinguish between experienced and inexperienced individuals.

7.2 Methodology

To create the questionnaire for this study, we created a question bank with 22 total questions, which represents 11 home automation scenarios, with 2 versions for each, one using Node-RED (flows)

and one using SIGNORE (rule specifications). The scenarios picked for this question bank were taken from the scenarios collected in the survey mentioned in Chapter 6 (p. 43). In order to have a wide variety of combinations of questions, we created 10 questionnaires, using Google Forms[1], with 10 questions randomly picked from the question bank. Figure 7.1 shows the methodology that lead to the creation of the questionnaires for this case study, from the 11 scenarios that were picked, until the random choice of questions for each questionnaire. Every version of the questionnaire had questions for both formats, therefore, each participant experienced them both, to be able to compare them, since in the end of the questionnaire we asked for their preferences in regard to understanding and implementing scenarios. We also asked participants how experienced they were with Node-RED or other VPL, to understand if the previous experience with VPL would affect their capability to describe the scenarios and if it would affect their preferences. Each participant was assigned a randomly picked version, in order to not influence the amount of responses each version got.

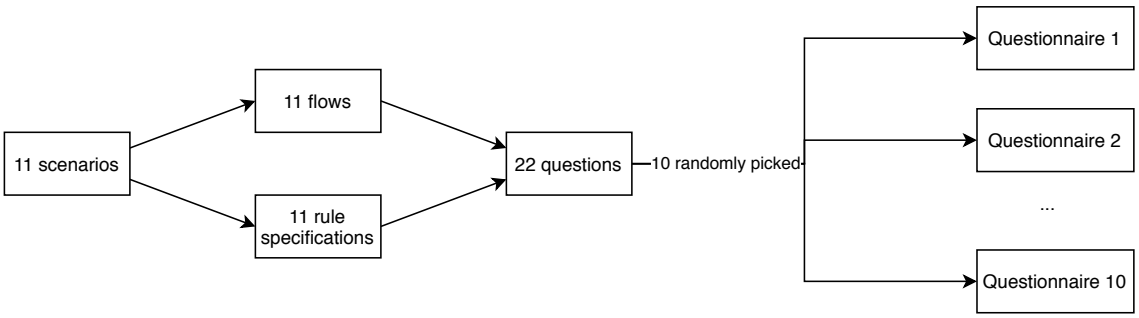

Figure 7.1: Methodology to create the questionnaires for the case study

We wanted to ensure that there was no bias in the assessment of the answers provided for the scenarios descriptions, therefore, when assessing the correctness of an answer, we did not know if the question was made for a Node-RED flow, or a SIGNORE specification. The assessment was made comparing the answer to the original description of the scenario, and if the meaning was the same, the answer is correct, otherwise, the answer is wrong.

7.3 Analysis

Table 7.1 (p. 60) shows the results of this study. The results show that for the first eight scenarios, the success rate is very high despite the format used to represent it. However, the last three scenarios show a different result. For scenario 9, the success rate was very low for both formats, being slightly higher for SIGNORE than for Node-RED. Scenarios 10 and 11 show a low success rate for the Node-RED representation, while showing 100% success for the Gherkin representation. As the focus of this dissertation is on IoT systems with sufficiently high complexity, we should focus on the last two scenarios, i.e. scenarios 10 and 11, as those are the ones that represent really complex home automation scenarios. These scenarios are highlighted in Table 7.1 (p. 60), due to

[1] https://www.google.com/forms/about/

their relevance. The first eight scenarios are not considered complex enough, since they represent scenarios with only one or two devices interacting, or one device working on a schedule. Scenario 9 is considered an outlier, due to the very low rate of success with either format. If we change focus to those two complex scenarios (10 and 11), the results are very different from the overall results from Table 7.1 (p. 60), showing that for home automation scenarios with higher complexity, users have a low success rate understanding the scenarios represented in Node-RED flows, while having a very high success rate understanding them represented in Gherkin.

Besides the success rate per scenario and per format, we also measure the success rate per level of experience with Node-RED or other VPL, to see whether the success rates vary from inexperienced to very experienced users. Table 7.2 (p. 61) contains the result for this measurement, showing that despite the level of experience with Node-RED or other VPL, the success rate for SIGNORE is higher than the success rate for the Node-RED format, being close too 100% for SIGNORE and around 70% for Node-RED. The results do not show a significant difference between the success rate of experienced and inexperienced Node-RED/VPL users.

Finally, after interpreting the scenarios in the study, participants were asked which format they understood better and which one they thought would be easier to implement scenarios with. Table 7.3 (p. 61) shows the results for these questions, showing that more than half of the participants claimed to understand Node-RED better than SIGNORE and would prefer to use Node-RED to implement home automation scenarios.

7.4 Discussion

Considering the results shown in Table 7.1 (p. 60), it is noticeable that the first 8 scenarios are too simple, since the success rate is almost 100% for both formats. It is also clear that scenario 9 is an outlier due to its very low success rate. This can be due to the fact that scenario 9 represents a one-time action, which is very different from the rest of the scenarios. One-time actions are not very common in Node-RED, as aforementioned, and introduce a different component that was not present in any of the other scenarios, the “once” component. Therefore, we should focus on scenarios 10 and 11, which represent complex home automation scenarios and show a clear difference between success understanding Node-RED and SIGNORE. The results for these two scenarios show that participants had 100% success understanding them in SIGNORE’s format, and only 45% success in Node-RED flows. Even though these results are very indicative that a textual format to represent home automation scenarios is easier for regular users to understand, the number of complex scenarios evaluated in the study is very low to prove that participants understand textual representations of home automation scenarios better than flow representations.

Table 7.2 (p. 61) shows that even very experienced Node-RED/VPL users performed better at interpreting home automation scenarios in the textual format than in the VPL format. This shows that introducing a semi-structured interface for home automation management can be useful for users with any level of experience with Node-RED or other VPL. However, just like the results from Table 7.1 (p. 60), the results from Table 7.2 (p. 61) are not sufficient to prove that IoT end-users

understand scenarios implemented with SIGNORE better than with Node-RED, as the population is not very large, but lead us to believe that both inexperienced and experienced Node-RED users can benefit from a textual format to represent home automation scenarios.

Looking at Table 7.3 (p. 61), it appears as if participants have a better understanding of home automation scenarios represented with Node-RED flows, than with SIGNORE. However, looking at Table 7.1 (p. 60) and Table 7.2 (p. 61), it is clear that they do not, since the success rate for SIGNORE is very high, while for Node-RED it is very lower. The results regarding the preference to implement scenarios show that users would prefer Node-RED over SIGNORE to implement home automation scenarios, but this can be affected by their belief that they understand Node-RED better than SIGNORE. When looking at the results per level of experience with Node-RED/VPL, we can see that the participants with less experience claimed to understand Node-RED better and that Node-RED would be their choice for implementation, but for participants with higher experience, the results are different. This may be due to the fact that experienced Node-RED users know how well they understand it and the issues with VPL, and, therefore, they know that SIGNORE may be more easily understood than Node-RED.

7.5 Threats to Validity

Even though the results from this case study are positive and lead us to believe that this project was successful in improving the users capabilities to manage complex IoT systems, there are some threats that may have affected the results.

The **sample size**, since the number of participants in the study was low. Even though the participants were varied in regard to experience with Node-RED, the amount of experienced Node-RED users was really low, and having a larger sample could have shown different results both for the success rate in describing the scenarios and for the preferences questions.

Besides the sample size, we also identified the **level of expertise** of the participants as a threat, since participants with different levels of experience with Node-RED, or IoT in general, could provide different results for the case study. To mitigate this threat, we made sure to have participants with various levels of experience, from completely inexperienced participants that had never used Node-RED or anything IoT related, to participants that were very experienced with Node-RED and IoT.

The **description** provided for the **Node-RED nodes** was simplified so that participants that were not familiar with Node-RED could understand how the nodes were used in home automation scenarios, which may have caused the success rate in describing the scenarios represented by Node-RED flows to be higher than it would be if the descriptions were the ones shown in Node-RED.

The **scenarios sample** also poses a threat to this validation, since a large part of the scenarios used were not very complex. Looking at the results from Table 7.1 (p. 60) it is noticeable that the success rates for complex scenarios are lower with Node-RED than with SIGNORE, which is not noticeable for the simpler scenarios. As this solution targets issues in managing complex IoT systems, the scenarios sample may have been too simple.

Finally, **not knowing if their answers were correct** may have affected the participants choice towards the preferred format for understanding and implementing scenarios. Comparing the results of the success in the questions to describe the scenarios and the results of the preferences questions, it is clear that some participants claimed to have understood better the scenarios represented by Node-RED flows, but had lower success than on the ones represented by our semi-structured language. This may also have affected their answer towards the preferred format for implementation. If the participants knew whether their answers were correct before providing their preferences, the results could have been different.

Considering these threats, we still believe that our results are valid, and also believe that if the threats were solved, the results would be even more supporting of our solution, since the threats identified appear to have contributed to a higher success in understanding Node-RED flows than would be achieved with these threats solved.

7.6 Summary

To evaluate whether users would have a better understanding of our solution, when compared to Node-RED, we performed a case study conducted by a questionnaire, in which participants had to interpret home automation scenarios implemented both with Node-RED and SIGNORE. We also asked participants for their opinion in regard to which tool they preferred for interpreting and implementing similar scenarios.

The results of this study show us that users have difficulties in understanding complex home automation scenarios represented by Node-RED flows, but understanding those scenarios represented by SIGNORE was easy for them, regardless of the level of experience that they had with Node-RED.

Considering the positive results from this study and also the threats to validity identified, we still believe that users would have more capabilities to manage complex home automation scenarios using a semi-structured interface than using a VPL.

Id	Scenario Description	Node-RED Success	SIGNORE Success	Overall Success (%)	Node-RED Success (%)	SIGNORE Success (%)
1	When the entrance sensor detects motion, turn on the entrance lights	16/17	23/23	98	94	100
2	When it rains, close the pool cover and close bedroom, bathroom and kitchen windows	9/9	31/31	100	100	100
3	When shower door opens and shower tap is running, then shower tap stops	5/6	22/22	96	83	100
4	When there is movement in the bedroom between 23:00 and 7:00, turn lights on at 200 brightness	27/30	8/8	92	90	100
5	When there is movement in the patio between 18:00 and 6:30, turn patio lights on	7/7	34/34	100	100	100
6	When time is 7:30, turn on the coffee machine, the hot water system and the kitchen lights	23/23	22/23	98	100	96
7	Every Saturday at 15:00, turn on the robot vacuum cleaner and the robot lawn mower	6/6	33/33	100	100	100
8	Turn on hot water system from 8:00 to 12:00 and from 18:00 to 21:30	4/4	13/13	100	100	100
9	When time is 7:00 today, turn bedroom light on	3/27	1/6	11	10	17
10	When there is no one in the pool, cover the pool and turn off the water heater	16/30	25/25	75	53	100
11	When it does not rain for 3 days and the watering system did not work in those 3 days, turn the watering system on	4/14	6/6	50	29	100
	Total	120/176	218/224	84	68	97

Table 7.1: Overall success rate and comparison between Node-RED and SIGNORE success rates.

Experience Level	Overall Success	Node-RED Success	SIGNORE Success
1	0.83	0.65	0.99
2	0.86	0.71	0.96
3	0.83	0.67	0.95
4	0.90	0.75	1.00
5	0.85	0.73	1.00
Total	0.85	0.68	0.97

Table 7.2: Success rate comparison between Node-RED and SIGNORE per level of experience with VPL.

Experience Level	Understand Node-RED better	Understand SIGNORE better	Implement with Node-RED	Implement with SIGNORE
1	0.62	0.38	0.62	0.38
2	0.55	0.45	0.55	0.45
3	0.43	0.57	0.86	0.14
4	0.75	0.25	0.50	0.50
5	0.00	1.00	0.50	0.50
Total	0.55	0.45	0.62	0.38

Table 7.3: Users preferences for understanding and implementing home automation scenarios per level of experience with VPL.

Chapter 8

Conclusions and Future Work

This chapter goes over the entire work of this project. First, Section 8.1 summarizes the project. In Section 8.2 we revisit our hypothesis and research questions, to understand how successful the project was. In Section 8.3 we describe the main contributions resulting from this project and, finally, in Section 8.4 we explain the project's limitations and future work.

8.1 Summary

As IoT is a very popular field of engineering in the current days, it is important to find its flaws and ways to solve them, as it has many applications and can have a larger impact in the world. The focus of this dissertation was to study the current state-of-the-art regarding IoT management and improve it.

The state of the art in this field shows that most IoT end-users, mainly novice and non-technical ones, have difficulties in setting up and managing IoT systems, even though there are some tools to help them. There are vendor apps designed to help customers interact with their devices, but these apps are device/brand specific, not offering flexibility and, as IoT is a very heterogeneous field, these apps require end-users to have an app for each brand that they have in their systems. Visual Programming Platforms (VPP) and smart assistants offer the flexibility to work with multiple devices/brands in one place. However, VPPs are too overwhelming and complex for most users and, although these platforms are able to manage very complex IoT systems, most users cannot work with them (*cf.* Section 3.1, p. 17). Smart assistants, on the other hand, are very simple tools, which provide comfortable interactions with IoT systems, but lack in complexity, which prevents users from managing complex systems with these assistants (*cf.* Section 3.1, p. 17). Some Model-Driven Engineering practices, such as model transformations, have been experimented with

to try to improve the management of IoT systems, by allowing the generation of platform-specific code from simpler models that represent that code, but are not very explored yet and facing many issues related to the complex logic of models and transformations (*cf.* Section 3.2, p. 21).

This project focused on developing a system that provides a simple interface for users, through an interface that interprets a semi-structured language, integrated with Node-RED, a VPP, which allows users to manage complex systems. This solution was named SIGNORE. Through the analysis of the state of the art and the features provided by our approach, this solution should tackle some of the current issues of IoT.

8.2 Hypothesis and Research Questions Revisited

In Chapter 4 (p. 25), we stated the issues that were found during the research, and defined the hypothesis for this dissertation:

> *The integration of a semi-structured text-based interface and visual programming improves users capabilities to manage complex IoT systems.*

With this hypothesis, we wanted to understand whether combining a semi-structured text-based interface and a VPP would allow users to manage complex IoT systems more easily and with more success than with the current state-of-the-art solutions (*cf.* Section 4.4, p. 27).

Our validation methodology was split in two parts, (1) a survey to collect home automation scenarios from varied participants, which resulting scenarios were used to test the solution's capabilities, and (2) a case study to measure how well participants understood home automation scenarios described with Node-RED and with SIGNORE.

The first part of the validation showed that SIGNORE is capable of managing complex IoT scenarios, being much more capable than the currently popular smart assistants (*cf.* Section 6.6, p. 53). The categorization and implementation of the collected scenarios showed that our tool can effectively implement almost all the scenarios, and the ones it cannot, could easily be added as features, in future work. The collection of the scenarios also showed that some users naturally describe home automation scenarios in a similar format to the one followed by our semi-structured language.

As for the second part, the results showed that users had difficulties in understanding complex scenarios represented by Node-RED flows, even if they were very experienced with this VPL, while showing no problems in understanding the textual format of our solution (*cf.* Section 7.6, p. 59).

Overall, the results of the validation lead us to believe that this project was successful and, even though there are some threats to this validation, we believe that solving those threats would have given us better results to support our hypothesis. Therefore, we believe that our solution is superior to the current smart assistants, in terms of management capabilities, and is easier to use in comparison to Node-RED on its own.

Looking at the results shown in the Chapter 6 (p. 43) and Chapter 7 (p. 55), we can now provide some answers to the RQ's from Section 4.4 (p. 27):

- **RQ1:** *Can users manage complex IoT systems using SIGNORE?* The results from Chapter 6 (p. 43) show us that SIGNORE is capable of managing a complex IoT system, supporting a very high percentage of types of home automation scenarios.
- **RQ2:** *Is there a pattern in the way users describe home automation scenarios?* The results from Chapter 6 (p. 43) show us that many users describe home automation scenarios following a pattern similar to the one followed by SIGNORE.
- **RQ3:** *Do users understand visual programming diagrams more easily than rules specified with this semi-structured language?* The results from Chapter 7 (p. 55) lead us to believe that users tend to understand scenarios represented with SIGNORE better than the ones represented with Node-RED flows.

Considering the results gathered, we believe that a semi-structured interface is easier for users to understand home automation scenarios, mainly complex ones, when compared to a VPP. Also, as many users follow a pattern similar to the one used by SIGNORE, and the results show that they understand it better than VPP, we believe that the combination of both approaches, as done by SIGNORE, improves users capabilities to manage complex IoT systems, compared to using only visual programming.

8.3 Main Contributions

As of the conclusion of this dissertation, there are some contributions worth noting, mainly in the field of IoT management. These contributions are the following:

1. **SIGNORE:** a complete tool for IoT management using a semi-structured interface.
2. **Model transformations from text to VPP:** for the development of SIGNORE, we studied model transformations and most of the current solutions use them to convert from visual programming formats to platform-specific code, or from text (simplified versions of programming languages) to code. In this dissertation, we implemented model transformations from text in a semi-structured text-based language to VPP flows, which is innovative, according to the state-of-the-art.
3. **Survey on home automation scenarios:** the survey described in Chapter 6 (p. 43), gathered a list of the most desired home automation scenarios from the participants. The analysis of that list resulted in a list of categories for those scenarios, that can be used in other works and be extended from here on. This list can be found in Appendix A (p. 67).
4. **Use of virtual devices:** the flexibility provided by SIGNORE in regard to device support allows the introduction of virtual devices that can be coded and work as a real device. This can be useful to extend the capabilities of a real device, with the virtual one working as a layer between the real one and the system.

8.4 Limitations and Future Work

After the development and validation of this project, we noted its success. However, we understand that there are aspects of this dissertation that can be taken on to continue the contributions in this field, which are the following:

1. **Support external services:** in order to support all the home automation scenarios that were provided by the survey participants, it would be necessary to implement support for external services, which are already supported by Node-RED. In its current state, SIGNORE does not support external services, but as the tool is highly scalable, it is possible to extend it to support these requests. It is also possible to add other features that may appear in the future. One possible way to do so, is to create a class that represents the external service and knows the HTTP request node from Node-RED, and have a list of specifications for those services, similar to the device specifications that already exist.

2. **Case study:** we believe that performing a new case study with more complex scenarios for participants to describe, solving the threats to validity that were identified in Section 7.5 (p. 58) could make it more noticeable that participants have difficulties understanding Node-RED flows, while they easily understand SIGNORE's semi-structured language.

3. **Quasi-experiment using SIGNORE and Node-RED**: we believe that performing a quasi-experiment using SIGNORE and Node-RED would help to properly validate the hypothesis of this dissertation. Having a group of users perform a set of tasks using SIGNORE and Node-RED, in which they would have to describe and implement home automation scenarios using one of the tools and compare the success rate, the implementation time and the number of attempts would contribute to the validation of our approach and contribute to the development of IoT management.

4. **Support model transformations from VPP to text:** in Chapter 4 (p. 25) we mentioned that the solution would allow users to convert the scenario representations in both directions, *i.e.* from text to diagram and vice versa. However, after the research done into model transformations and the implementation of the transformation from text to diagram, the inverse transformation remains as an open challenge. Converting SIGNORE's representation to Node-RED is simpler, because we are converting a more ambiguous and abstract representation to a more specific and less abstract one. However, when trying to convert from a less abstract representation to a more ambiguous and abstract one, there may be loss of meaning or information. Therefore, there needs to be done more research and experimenting into this topic, in order to enlarge the contributions of this dissertation.

Appendix A

Categories of scenarios

This appendix contains the categories and scenarios collected from the answers of the survey described in Chapter 6.

A.1 Sensors and actuators

1. According to temperature and humidity, water the garden
2. Adjust blinds according to brightness
3. Adjust lights intensity to a level
4. Adjust pool water temperature according to outside temperature, if someone is using it
5. Adjust the watering system to the humidity and temperature
6. Alert owner(s) if motion detector is triggered when the house is empty
7. Automatic lights based on presence / movement
8. Blinds inclination system based on outside light
9. Boil water in kettle for coffee when person awakes
10. Coffee machine turns on when first shower of the day
11. Coffee machine, toaster, etc controlled by fitbit/apple watch. Use app to monitor sleep habits and turn on devices when the user wakes up after 6:00
12. Cover the pool, if it is raining
13. Close all blinds when no one is home
14. Close all the blinds when I go to sleep
15. Close the garage door after the entering of a car
16. Digital wishlist/list of items to buy. Inform owner when products are on sale in a specific store
17. Facial recognition to identify unidentified individuals and notify owner

18. Garden automation: stable soil moisture level, temperature stabilisation in adverse weather
19. Heat the pool water if the water temperature is below 23ºC, up to 28ºC maximum
20. Holiday mode: when any device is used/triggered notify the owner
21. Humidifier automation for entire house
22. If carbon monoxide concentration is too high, slightly open kitchen window
23. If I turn on the shower, then turn on the heated towel rack
24. If motion in garden, turn on the lights
25. If the garden humidity drops below X, turn on the watering system
26. If the garden is dry, turn on watering system
27. If the grass is above certain height activate the robot lawn mower
28. If it's dark outside and I turn on the TV, then turn on the bedroom lights
29. If there is a CO2 leak on the garage (air quality sensors), open garage door and inside door and alert through the sound system
30. If there is smoke detection, turn off stove and oven
31. In case of flood, shut the electricity in the flooded room
32. Inform owner about electric consumption in each room (reader connected to the electric circuit)
33. Inform owner when pets food is about to finish
34. Intensity of lights based on the available natural light
35. Maintain room temperature in between a specified permissible range
36. Maintain temperature to 18º
37. Make coffee 30min after waking up
38. Make coffee when the alarm clock rings
39. Notify when washing machine and drier finish
40. Open doors to first responders in case of distress (fire, heart attack, etc)
41. Open garage when car is inside and engine starts, or when car approaches home
42. Open the blinds when there is too much smoke
43. Open the garage door when I start the car
44. Open the living room windows if the temperature outside is bigger than 23ºC and if I'm in the house
45. Open window blinds when the alarm clock rings
46. Pool cover closes when it rains

47. Robot lawn mower turns on if grass not humid and grass above xx cm
48. Send SMS alert if faulty freezer/fridge
49. Shower tap stops when the shower door opens
50. Thermostat to control air temperature / air flow controlled by air quality, smoke and humidity sensors
51. Towel rack heats when humid towel on it
52. Turn coffee machine on when shouting "I want coffee" indoors
53. Turn off TV if person falls asleep
54. Turn on the AC when temperature is higher than X
55. Turn on the bedroom lights if the alarm system is triggered
56. Turn on the garage light if there is movement on the garage
57. Turn on the security system when I leave the house or when I go to sleep
58. Turn on the vacuum cleaner when no one is home
59. Turn on the vacuum robot when the car leaves the garage
60. Turn the lights on when you enter a room and off when you leave
61. Turn the sound system on and play "Stairway to Heaven" on loop when entering the bathroom
62. Turn TV on if very important NEWS and if someone at home
63. TV turns off when someone says good night
64. Vent the bathroom after showers until ideal humidity levels
65. Ventilation strength level automation according to vapour/smoke in kitchen
66. Voice control over coffee machine/blinds/lights/etc
67. Water the lawn only if the lawn mower is not working
68. When a car approaches the garage, open the garage door
69. When a wet towel is put on the heated towel rack, it turns on until the towel is dry
70. When air quality sensor reports low O2 (or high CO2, or any other harmful gas) -> open windows on that specific room
71. When alarm is triggered, send notification to phone
72. When an intruder is detected, call the police, alert owners and alert through the sound system that the police is on the way
73. When bathroom motion sensor reports movement -> turn on heated towel rack
74. When earth humidity sensor reports low humidity -> turn on watering system

75. When everyone leaves home, lock all doors, turn all lights off and shut down unnecessary devices
76. When everyone leaves home, turn on the vacuum cleaner
77. When garage light sensors detect car lights, open the garage door
78. When I say "I want to go to the pool", the pool cover opens, and the water heating is set to 22º
79. When I turn on the security system, close all windows and blinds
80. When it is raining, close the windows
81. When it rains, cover the pool
82. When lawn is too long, activate the lawnmower robot
83. When no one is in the pool, the pool cover closes and the water heating system turns off
84. When pool water temperature is below (air temperature - 5), turn on heating
85. When security cameras detect someone unidentified, lock the doors, windows and close blinds
86. When shower tap starts running, turn on towel rack
87. When smoke detectors are activated, alert through the sound system and alarm (send message to owner) and warn fire department
88. When smoke sensors activated, send notification to phone
89. When someone enters a room, turn the lights on. When someone leaves a room, turn the lights off
90. When someone uses the toilet, it flushes automatically after the person steps away
91. When taking a phone call, turn off vacuum cleaner and sound system (if working) and turn on after call ends
92. When temperature below 10ºC in garden sensor, heat up towel rack and heating system
93. When the bedroom temperature is above 20º and outside temperature is below, open the windows
94. When the dishwasher is full, start working
95. When the entrance sensor detects motion, turn on the entrance lights
96. When the kitchen hood detects steam, turn it on
97. When the TV remote moves, turn the tv on
98. When the washing machine finishes, alert through the sound system
99. When TV on, reduce the lights intensity in the room
100. When user wakes up, inform if any accident can make traffic worse

101. When washing machine finishes, send notification to phone
102. When yell "Never gonna give you up", turn on sound system and TV with Rick Roll
103. With a solar panel; set a given machine on hold (e.g. washing machine), and when producing a lot of extra energy that would go unused (regular people can't sell energy to the network in Portugal) -> turn on machines that were on hold

A.2 Actuators on schedule

1. After 8pm, close the window blinds
2. Bedroom blinds open gradually from 6:30 to 6:40
3. Close all windows, blinds, doors, turn on alarm and turn off all lights on schedule
4. Close the bedroom windows at 8pm
5. Close the blinds at the end of the day (related to sunset or a schedule)
6. Clean the pool water, weekly
7. Coffee machine, toaster, etc turn on at 7:00 on weekdays and 9:00 on weekends
8. Every saturday at 3pm o'clock, turn on the robot vacuum cleaner and the robot lawn mower
9. Every tuesday at 2am turn on the pool cleaning system
10. Every weekday at 9h turn on coffee machine
11. Fake inhabited house for security in holiday periods - open blinds and turn on lights at different times during the day
12. Feed pets on a schedule with specific amounts
13. From 00h to 10h, unless deactivated, turn on security alarm
14. From 16h to 17h daily turn on the vacuum cleaner robot
15. Gradually open the blinds before alarm clock goes off, or along with sunrise if there is no alarm
16. Heat the house when it is occupied (based on time)
17. If hot water system is based on electricity, heat when electricity is cheaper
18. Keep living room temperature around 20C on weekends
19. Lawn mower working on schedule
20. Mow lawn and vacuum periodically
21. On schedule (like wake up) make coffee in the kitchen's coffee machine, start heating the towel rack, turn on bedroom bedside lamps and TV at low volume on news channel, open the pool cover and start heating the water and open the living room window' blinds

22. On schedule (morning), ring an alarm and open the blinds
23. On schedule turn on coffee machine
24. On schedule, activate lawn mower robot
25. Open the bedroom blinds and windows at 10 am
26. Open the living room blinds at 8am
27. Open and close blinds on schedule
28. Schedule robot lawn mower to run every Saturday at 9am
29. Turn lights on and off and/or open and close blinds on schedule when you're away for extended periods of time
30. Turn lights on/off on schedule
31. Turn on hot water system everyday from 8h to 12 and from 18h to 21h30
32. Turn on the bathroom heating 1h before waking up
33. Turn on the robot vacuum cleaner on saturdays in the afternoon
34. Turn the heating system on, set to the preferred temperature, on schedule
35. Turn the oven on and off on schedule
36. When it is 9:30 pm, turn on the dishwasher
37. When it is 6am and week day, gradually increase the bedside lights intensity
38. When it is 6am and week day, turn on the hot water system
39. When it is wednesday and 14:00, turn on pool heating system on 24º
40. When time is 7:30 am, turn on the coffee machine, the hot water system and the kitchen lights (if they were turned off)
41. When time is 7 am, open the blinders on the windows and when the time is 8 pm, close the blinders
42. While taking a shower, water heating adjust to bathroom temperature and if humidity > level, turn on heated towel rack

A.3 Actuators on time interval, with sensors

1. Bathroom heating starts at 6:32 until the temperature is 20º
2. Bedside lamps turn on if movement around the bed during night
3. During night, when there is motion in one room, light that room at 50% intensity
4. During the night, if there is any window or door open, send a notification to phone

5. Every morning (after x hours) gradually increase intensity of bedside lamps until max, if blinds closed
6. Every weekend, if humidity sensor is below X, turn on watering system
7. If sound system is on and volume is over X after 22h00, dial volume down to Y
8. Robot vacuum cleaner when nobody is home 2x/week
9. Turn on patio lights in case of motion between 18h and 6h30
10. Water garden on schedule (dawn or dusk) if humidity is below optimal
11. When a tv series starts and I am home, turn on the TV on that channel and prepare popcorn. If I am not home, record it
12. When certain devices (tv, sound system, lawn mower) are on at 4h turn them off and notify owner
13. When garage inside door opens during the night, turn on living room lights
14. When turning on the dishwasher (or washing machine) after 20h, wait for 00h to start
15. Windows motorised which opens during the night in summer to cool the house

A.4 Sensors with timers

1. Activate robot vacuum cleaner at 10h, if it did not work on the previous day
2. Cleaning system of the pool turning automatically while the heating system is on based on a 10h/day working schedule
3. Cover the pool when no one is inside after xx minutes
4. If alarm is off and no motion is detected in the house for 1 hour, turn alarm on
5. If hot water system is on for more than 15 minutes and bathroom lights are on, turn on heated towel rack
6. If no one is at home for more than x hours and stove is on -> turn off stove
7. Mow the lawn, if its height is above 5cm and it did not rain for 24h
8. Turn off lights in a room when there is no motion for 15 minutes
9. Water the lawn, if it did not rain in the last 3 days

A.5 Actuators with timers

1. When it is 23:00, turn on the garden watering system for 10 minutes

A.6 External services

1. If it will rain in the next hour, inform that drying outside is not the best plan
2. Turn on washing machine if capacity is at over 80% and the weather forecast for the next few hours is sunny during daytime
3. Water the garden regularly, depending on weather prediction. Do not water if tomorrow is going to rain heavily
4. With a solar panel; and with weather information; schedule a machine to run sometime during the day (e.g. washing machine) -> system automatically schedules it to the time of day that is most likely to be sunny
5. With a solar thermal collector (for heating water); when sun is expected during the following hours -> turn off traditional water heating system

A.7 One-time actions

1. Shut all unnecessary devices
2. When it is 7:00 today, turn on the bedroom lights

Appendix B

Smart devices list

This appendix contains a table with the smart devices that were provided to the participants of the survey described in Chapter 6.

Common	Motion, temperature, humidity, smoke and air quality sensors
	Security system (cameras and alarm)
	Lights
	Windows and blinds
	Hot water system
	Heating system
	Robot vacuum cleaner
	Sound system
Garage	Garage door
	Inside door
	Washing machine
	Dryer
Patio	Entrance door
Pool	Pool cover
	Cleaning system
	Water temperature sensor
	Water heating system
Garden	Watering system
	Humidity sensor
	Temperature sensor
	Robot lawn mower
Living room	TV
Kitchen	Stove
	Oven
	Hood
	Dishwasher
	Coffee machine
Bedroom	TV
	Bedside lamps
Bathroom	Heated towel rack

Table B.1: Smart devices provided for survey, organized by location.

References

[1] 42gears. 6 Types of Wearable Technology You Must Know. https://www.42gears.com/blog/6-wearable-technologies-you-must-know-right-now/, 2015. [Online; accessed January 2020]. Cited on p. 8.

[2] Ademar Aguiar, André Restivo, Filipe Figueiredo Correia, Hugo Sereno Ferreira, and João Pedro Dias. Live software development: Tightening the feedback loops. In *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming*, Programming '19, New York, NY, USA, 2019. Association for Computing Machinery. Cited on p. 10.

[3] T. Alam. A reliable communication framework and its use in internet of things (iot). *Journal of Engineering and Applied Sciences*, 3, 05 2018. Cited on pp. 1 and 6.

[4] Tiago Almeida, Hugo Sereno Ferreira, and Tiago Boldt Sousa. A collaborative expandable framework for software end-users and programmers. In Yuhua Luo, editor, *Cooperative Design, Visualization, and Engineering*, pages 163–166, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. Cited on p. 9.

[5] S. Blanc. iOS Shortcuts: The Ultimate Guide for Resources, Examples, Libraries, Triggers, and More. https://thesweetsetup.com/ios-shortcuts-guide/, 2019. [Online; accessed February 2020]. Cited on p. 17.

[6] M. Boshernitsan and M. Downes. Visual Programming Languages: A Survey. *Computer Science Division (EECS)*, 2004. Cited on p. 9.

[7] M. Castro, A. J. Jara, and A. F. G. Skarmeta. Smart lighting solutions for smart cities. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 1374–1379, March 2013. Cited on p. 6.

[8] Omar Castro, Hugo Sereno Ferreira, and Tiago Boldt Sousa. Collaborative web platform for unix-based big data processing. In Yuhua Luo, editor, *Cooperative Design, Visualization, and Engineering*, pages 199–202, Cham, 2014. Springer International Publishing. Cited on p. 9.

[9] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang. A vision of iot: Applications, challenges, and opportunities with china perspective. *IEEE Internet of Things Journal*, 1(4):349–359, Aug 2014. Cited on p. 8.

[10] G. Chowdhury. Natural language processing. *ARIST*, 37:51–89, 01 2005. Cited on p. 11.

[11] M. Clark, P. Dutta, and M. W. Newman. Towards a natural language programming interface for smart homes. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, page 49–52, New York, NY, USA, 2016. Association for Computing Machinery. Cited on p. 18.

[12] A. Colon and E. Griffith. The Best Smart Home Devices for 2020. https://www.pcmag.com/article/303814/the-best-smart-home-devices-for-2019, 2019. [Online; accessed January 2020]. Cited on p. 7.

[13] K. Czarnecki and S. Helsen. Classification of model transformation approaches. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003. Cited on p. 14.

[14] J. den Haan. MDE - Model Driven Engineering - reference guide. http://www.theenterprisearchitect.eu/blog/2009/01/15/mde-model-driven-engineering-reference-guide/#model, 2009. [Online; accessed January 2020]. Cited on p. 13.

[15] J. P. Dias, J. P. Faria, and H. S. Ferreira. A reactive and model-based approach for developing internet-of-things systems. In *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*, pages 276–281, 2018. Cited on p. 14.

[16] João Pedro Dias, Bruno Lima, João Pascoal Faria, André Restivo, and Hugo Sereno Ferreira. Visual self-healing modelling for reliable internet-of-things systems. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 357–370, Cham, 2020. Springer International Publishing. Cited on p. 10.

[17] B. Dickson. The fundamental problem with smart speakers and voice-based AI assistants. https://bdtechtalks.com/2018/09/03/challenges-of-smart-speakers-ai-assistants/, 2018. [Online; accessed January 2020]. Cited on p. 12.

[18] A. F. Einarsson, P. Patreksson, M. Hamdaqa, and A. Hamou-Lhadj. Smarthomeml: Towards a domain-specific modeling language for creating smart home applications. In *2017 IEEE International Congress on Internet of Things (ICIOT)*, pages 82–88, June 2017. Cited on p. 22.

[19] T. Eterovic, E. Kaljic, D. Donko, A. Salihbegovic, and S. Ribic. An internet of things visual domain specific modeling language based on uml. In *2015 25th International Conference on Information, Communication and Automation Technologies, ICAT 2015 - Proceedings*, 10 2015. Cited on pp. 2 and 21.

[20] H. S. Ferreira, A. Aguiar, and J. P. Faria. Adaptive object-modelling: Patterns, tools and applications. In *2009 Fourth International Conference on Software Engineering Advances*, pages 530–535, 2009. Cited on p. 13.

[21] Hugo Sereno Ferreira, Filipe Figueiredo Correia, and Ademar Aguiar. Design for an adaptive object-model framework: An overview. In *4th Workshop on Models@run.time at MODELS*, 2009. Cited on p. 13.

[22] Hugo Sereno Ferreira, Filipe Figueiredo Correia, and Leon Welicki. Patterns for data and metadata evolution in adaptive object-models. In *Proceedings of the 15th Conference on Pattern Languages of Programs*, PLoP '08, New York, NY, USA, 2008. Association for Computing Machinery. Cited on p. 13.

[23] Hugo Sereno Ferreira, Filipe Figueiredo Correia, Joseph Yoder, and Ademar Aguiar. Core patterns of object-oriented meta-architectures. In *Proceedings of the 17th Conference on Pattern Languages of Programs*, PLOP '10, New York, NY, USA, 2010. Association for Computing Machinery. Cited on p. 13.

[24] Data Flair. IoT Applications | Top 10 Uses of Internet of Things. https://data-flair.training/blogs/iot-applications/, 2018. [Online; accessed January 2020]. Cited on p. 6.

[25] F. Fleurey, B. Morin, A. Solberg, and O. Barais. Mde to manage communications with and between resource-constrained systems. In *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems*, MODELS'11, page 349–363, Berlin, Heidelberg, 2011. Springer-Verlag. Cited on p. 14.

[26] A. Floarea and V. Sgârciu. Smart refrigerator: A next generation refrigerator connected to the iot. In *2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6, June 2016. Cited on p. 6.

[27] C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S. Sarma. *The Internet of Things: First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008. Proceedings*, volume 4952. Springer, 01 2008. Cited on p. 6.

[28] C. Forsey. The 13 Best Smart Home Devices and Systems for 2020. https://blog.hubspot.com/marketing/smart-home-devices, 2019. [Online; accessed January 2020]. Cited on p. 7.

[29] Gherkin. Gherkin Syntax - Cucumber Documentation. https://cucumber.io/docs/gherkin/. [Online; accessed January 2020]. Cited on p. 3.

[30] Fortune Business Insights. Internet of Things Market Size, Growth | IoT Industry Report 2026. https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307, 2019. [Online; accessed January 2020]. Cited on p. 6.

[31] ITU. *ITU Internet Reports 2005: The Internet of Things*. International Telecommunication Union - ITU, 2005. Cited on p. 6.

[32] B. Juang and L. Rabiner. Automatic speech recognition - a brief history of the technology development, 01 2005. Cited on p. 11.

[33] K. Ashton. That 'internet of things' thing. *RFiD Journal*, 22:97–114, 2009. Cited on p. 5.

[34] A. S. Lago, J. P. Dias, and H. S. Ferreira. Conversational interface for managing non-trivial internet-of-things systems. In *Computational Science – ICCS 2020*, pages 384–397, Cham, 2020. Springer International Publishing. Cited on pp. 12, 20, and 26.

[35] H. Langley. The best Siri commands for controlling HomeKit and the Apple HomePod. https://www.the-ambient.com/guides/best-siri-commands-apple-homekit-homepod-557, 2020. [Online; accessed January 2020]. Cited on p. 18.

[36] J. Lee. 9 Reputable Smart Home Brands With Products You Can Trust. https://www.makeuseof.com/tag/smart-home-brands-can-trust/, 2016. [Online; accessed January 2020]. Cited on p. 7.

[37] Pedro Lourenço, João Pedro Dias, Ademar Aguiar, Hugo Sereno Ferreira, and André Restivo. Experimenting with liveness in cloud infrastructure management. In Ernesto Damiani, George Spanoudakis, and Leszek A. Maciaszek, editors, *Evaluation of Novel Approaches to Software Engineering*, pages 58–82, Cham, 2020. Springer International Publishing. Cited on p. 9.

[38] T. Maddox. The 4 IoT products a smart city needs in 2017. https://www.techrepublic.com/article/the-5-iot-products-a-smart-city-needs-in-2017/, 2016. [Online; accessed January 2020]. Cited on p. 8.

[39] T. Martin, D. Priest, D. Smith, and A. Gebhart. Every Google Assistant command to give your Nest speaker or display. https://www.cnet.com/how-to/every-google-assistant-command-to-give-your-nest-speaker-or-display/, 2019. [Online; accessed January 2020]. Cited on p. 18.

[40] T. Mens and P. Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152, 03 2006. Cited on pp. 13 and 14.

[41] B. Myers. Visual programming, programming by example, and program visualization: A taxonomy. *ACM SIGCHI Bulletin*, 17:59–66, 04 1986. Cited on p. 9.

[42] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs. Frasad: A framework for model-driven iot application development. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 387–392, Dec 2015. Cited on p. 21.

[43] Node-red. Flow-based programming for the Internet of Things. https://nodered.org/, 2017. [Online; accessed November 2019]. Cited on p. 3.

[44] A. Nordrum. The internet of fewer things [news]. *IEEE Spectrum*, 53(10):12–13, October 2016. Cited on p. 6.

[45] D. Priest, T. Dyson, and T. Martin. Every Alexa command to give your Amazon Echo smart speaker or display. https://www.cnet.com/how-to/every-alexa-command-to-give-your-amazon-echo-smart-speaker-or-display/, 2019. [Online; accessed January 2020]. Cited on p. 18.

[46] A. Rajalakshmi and H. Shahnasser. Internet of things using node-red and alexa. *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, pages 1–4, 09 2017. Cited on p. 19.

[47] P. Rani, J. Bakthakumar, B. Kumaar, U. Kumaar, and S. Kumar. Voice controlled home automation system using natural language processing (nlp) and internet of things (iot). *2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM)*, pages 368–373, 03 2017. Cited on pp. 18 and 19.

[48] P. P. Ray. A Survey on Visual Programming Languages in Internet of Things. *Scientific Programming*, 2017, 2017. Cited on p. 9.

[49] L. Rodríguez-Gil, J. García-Zubia, P. Orduña, A. Villar-Martinez, and D. López-De-Ipiña. New approach for conversational agent definition by non-programmers: A visual domain-specific language. *IEEE Access*, 7:5262–5276, 2019. Cited on pp. 22 and 23.

[50] J. Santos, J. Rodrigues, B. Silva, J. Casal, K. Saleem, and V. Denisov. An iot-based mobile gateway for intelligent personal assistants on mobile health environments. *Journal of Network and Computer Applications*, 71, 03 2016. Cited on p. 2.

[51] T. Simões. Visual Programming is Unbelievable... Here's Why We Don't Believe In It. https://www.outsystems.com/blog/visual-programming-is-unbelievable.html, 2015. [Online; accessed January 2020]. Cited on p. 10.

[52] C. Solis and X. Wang. A study of the characteristics of behaviour driven development. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 383–387, Aug 2011. Cited on p. 3.

[53] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, European Commission. Directorate-General for the Information Society, and Media. *Vision and Challenges for Realising the Internet of Things*. Publications Office of the European Union, 2010. Cited on p. 7.

[54] P. Suresh, J. V. Daniel, V. Parthasarathy, and R. H. Aswathy. A state of the art review on the internet of things (iot) history, technology and fields of deployment. In *2014 International Conference on Science Engineering and Management Research (ICSEMR)*, pages 1–8, Nov 2014. Cited on pp. 5, 6, and 8.

[55] C. J. Sutherland and B. MacDonald. Robolang: A simple domain specific language to script robot interactions. In *2019 16th International Conference on Ubiquitous Robots (UR)*, pages 265–270, 2019. Cited on p. 23.

[56] T. Szydlo, R. Brzoza-Woch, J. Sendorek, M. Windak, and C. Gniady. Flow-based programming for iot leveraging fog computing. In *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 74–79, June 2017. Cited on p. 9.

[57] B. Ur, E. McManus, M. Ho, and M. Littman. Practical trigger-action programming in the smart home. *Conference on Human Factors in Computing Systems - Proceedings*, 04 2014. Cited on p. 18.

[58] R. Van Der Straeten, T. Mens, and S. Baelen. Challenges in model-driven software engineering. *MODELS 2008. LNCS*, 5421:35–47, 09 2008. Cited on p. 15.

[59] O. Vermesan and P. Friess. *Internet of things: from research and innovation to market deployment*. River Publishers, 2014. Cited on p. 1.

[60] S. A. Weis. Rfid (radio frequency identification) : Principles and applications. *International Journal of Librarianship and Administration*, 2007. Cited on p. 6.

[61] Wikipedia. Isomorphism. https://en.wikipedia.org/wiki/Isomorphism, 2020. [Online; accessed January 2020]. Cited on p. 14.

[62] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, Feb 2014. Cited on pp. 1 and 8.