FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Decentralized Orchestration of IoT in End-user Programming Environments

Ana Margarida Oliveira Pinheiro da Silva

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo Sereno Ferreira

Second Supervisor: André Restivo

January 19, 2020

Decentralized Orchestration of IoT in End-user Programming Environments

Ana Margarida Oliveira Pinheiro da Silva

Mestrado Integrado em Engenharia Informática e Computação

January 19, 2020

Abstract

The Internet-of-Things (IoT) is an ever growing network of devices connected to the Internet. Such devices are heterogeneous in their protocols and computation capabilities. With the rising computation and connectivity capabilities of these devices, the possibilities of their use in IoT systems increases. Concepts like smart cities are the pinnacle of the use of these systems, which involves a big amount of different devices in different conditions.

There are several tools for building IoT systems; some of these tools have different levels of expertise required and employ different architectures. One of the most popular is Node-RED. It allows users to build systems using a visual data flow architecture, making it easy for a non-developer to use it.

However, most of these mainstream tools employ centralized methods of computation, where a main component — usually hosted in the cloud — executes most of the computation on data provided by edge devices, *e.g.* sensors and gateways. There are multiple consequences to this approach: (a) edge computation capabilities are being neglected, (b) it introduces a single point of failure, and (c) local data is being transferred across boundaries (private, technological, political...) either without need, or even in violation of legal constraints. Particularly, the principle of Local-First — *i.e.*, data and logic should reside locally, independent of third-party services faults and errors — is blatantly ignored.

Previous work attempt to mitigate some of these consequences, usually through tools that extend existing visual programming frameworks, such as Node-RED. They go as far as to propose a solution to decentralize flows and its execution in fog/edge devices. So far, achieving such decentralization requires that the decomposition and partitioning effort be manually specified by the developer when building the system.

Our goal is to extend Node-RED to allow automatic decomposition and partitioning of the system towards higher decentralization, by inferring computational boundaries. Furthermore, through automatic detection of abnormal run-time conditions, we also intend to provide dynamic self-adaptation. The prototype developed will be first validated with real devices and later with simulations.

As a result, we expect to achieve a more robust and efficient execution of IoT systems, by leveraging edge and fog computational capabilities present in the network, and improving overall reliability.

Keywords: Internet of Things, Visual Programming, Edge Computing

Resumo

A Internet-of-Things (IoT) é uma rede de dispositivos conectados à Internet em constante crescimento. Estes dispositivos são heterogéneos nos seus protocolos e capacidades de computação. Com o crescimento das capacidades de computação e conectividade destes dispositivos, as possibilidades do seu uso em sistemas IoT aumentaram. Conceitos como Cidades Inteligientes são o pináculo do uso destes sistemas, que envolvem um grande número de dispositivos diferentes em diferentes condições.

Existem várias ferramentas para construir sistemas IoT; algumas destas ferramentas requerem diferentes níveis de perícia e usam diferentes arquiteturas. Uma das ferramentas mais populares é Node-RED. Esta permite aos seus utilizadores construir sistemas usando uma arquitetura visual de *data flow*, tornando o processo mais fácil para um utilizador não programador.

No entanto, a maioria das ferramentas convencionais usam métodos centralizados de computação, onde um componente principal - normalmente alocado na *cloud* - executa a maioria da computação nos dados provenientes dos dispositivos *edge*, *e.g.* sensores e *gateways*. Com esta abordagem estão associadas multiplas consequências: (a) capacidades de computação de dispositivos *edge* estão a ser neglegenciadas, (b) introduz um único ponto de falha, e (c) data local está a ser transferida através de limites (privados, tecnológicos, políticos...) sem necessidade ou violando restrições legais. Especificamente, o princípio de *Local-First - i.e.*, dados e lógica devem residir localmente, independentemente de falhas e erros de serviços terceiros - é totalmente ignorado.

Trabalhos feitos até agora tentam mitigar algumas destas consequências, construindo ferramentas que estendem ferramentas existentes de programação visual, como Node-RED. Algumas propõe uma solução que consiste na descentralização de *flows* e a sua execução em dispositivos de *fog* e *edge*. Atualmente, para obter a este tipo de descentralização é necessário que o esforço de decomposição e partição seja manualmente efetuado pelo programador quando este constrói o sistema.

O nosso objetivo é estender a ferramenta Node-RED para permitir a decomposição e partição automática do sistema com o fim de obter uma maior descentralização. Para isso é necessário deduzir os limites de computação do sistema. Para além disso, também pretendemos que o sistema se adapte automaticamente às mudanças do ambiente, detectando automaticamente condições anormais em *run-time*. O protótipo construído será validado, numa primeira fase, com dispositivos reais e, mais tarde, com o uso de simulações.

Como resultado, esperamos construir uma execução de sistemas IoT mais robusta e eficiente, aproveitando as capacidades de computação presentes nos dispositivos *edge* e *fog* da rede, e melhorando a confiança e segurança do sistema.

Keywords: Internet of Things, Visual Programming, Edge Computing

Acknowledgements

TODO

Ana Margarida Silva

"Until I began to learn to draw,
I was never much interested in looking at art."

Richard P. Feynman

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
IoT	Internet of Things
VPL	Visual Programming Language
WWW	*World Wide Web*
IFTTT	*If This Then That*

Chapter 1

Introduction

This chapter introduces the motivation and scope of this project, as well as the problems it aims to solve. Section 1.1 details the context of this project in the area it is based on. Section 1.3 explains the reason why this work and the area it belongs to is important. Then, section 1.2 defines the problem we aim to solve and the goals of this dissertation are described in section 1.4. Finally, the section 1.5 describes the structure of this document and what content it contains.

1.1 Context

The Internet of Things paradigm states that all devices, independently of their capabilities, are connected to the Internet and allow for the transfer, integration and analytic of data generated by them [14]. This paradigm has several characteristics, such as the heterogeneity and high distribution of devices as well as their increasing connectivity and computational capabilities [5]. All this factors allow for a great level of applicability, enabling the realization of systems for management of cities, health services and industries [17].

The interest in Internet of Things has been growing massively, following the rising of connected devices along these past years. According to Siemens, in 2020 there will be around 26 billion physical devices connected to the Internet and in 2025 the predictions are pointing at 75 billion [4]. Although this allows for more opportunities, it is important to note that these devices are very different in their hardware and capabilities, which causes several problems in terms of development the systems, as well as their scalability, maintainability and security.

Visual Programming Languages (VPLs) allow the user to communicate with the system by using and arranging visual elements that can be translated into code [15]. It provides the user with an intuitive and straightforward interface for coding at the possible cost of loosing functionality. There are several programming languages with different focuses, such as education, video game development, 3D building, system design and even Internet of Things [40]. Node-RED[1] is one of

[1] https://nodered.org/

the most famous open source visual programming tool, originally developed by IBM's Emerging Technology Services team and now a part of the JS Foundation, which provides an environment for users to develop their own Internet of Things systems.

Non-functional attributes in a system are very important, specially attributes such as resiliency, fault-tolerance and self-healing in Internet-of-Things systems. All these attributes mean that when an error or problem occurs, the system can adapt and overcome them in a dynamic and automatic way. ADD REFERENCES

Node-RED, mentioned above, is a centralized system, as well as most of the visual programming environments applied to IoT. A centralized architecture has a central instance that executes all computational tasks on the data provided by the other devices in the network. On the other hand, in a decentralized architecture the central instance, if it exists, partitions the computational tasks in independent blocks that can be executed by other devices. In IoT, these decentralized architectures are mentioned in Fog and Edge computing.ADD REFERENCES

1.2 Problem Definition

Most mainstream visual programming tools focused on Internet of Things, Node-RED included, have a centralized approach, where a main component executes most of the computation on data provided by edge devices, e.g. sensors and gateways. There are several consequences to this approach: (a) computation capabilities of the edge devices are being ignored, (b) it introduces a single point of failure, and (c) local data is being transferred across boundaries (private, technological, political...) either without need, or even in violation of legal constraints. The principle of Local-First - i.e, data and logic should reside locally, independent of third-party services faults and errors - is being ignored.

Besides being a single point of failure, centralized systems can be less efficient than decentralized ones and in this context it might be the case, since there are computation capabilities that aren't being taken advantage of.

Chapter 4 expands on the problem definition, explaining it in bigger detail, defining its scope, desiderata, use cases and research questions.

1.3 Motivation

Check this

Internet of Things is a rapid growing concept that is being applied to several areas, such as home automation, industry, health, city management and many others. Given the number of existing systems with different protocols and architectures, it becomes difficult for a user to build a system that is in accordance to standards [3].

With the appearance of visual programming languages focused in IoT, more specifically Node-RED, users can build their own systems in an easier and streamlined way, removing the overhead

of learning advanced programming concepts and protocols. However, the existing solutions aren't resilient and fault-tolerant, which are very important requirements in these types of systems.

1.4 Goals

The main goal of this dissertation is to leverage the computation capabilities of the devices in the network, increasing efficiency, fault-tolerance, resiliency and scalability in an Internet of Things system.

To achieve this goal, a prototype will be developed, extending or rewriting Node-RED, that enables IoT devices to communicate their "computational capabilities" back to the orchestrator. In its turn, the orchestrator is able to partition the computation and send "tasks" to the nodes, which are the devices in the network, leveraging their computation power and independence.

As a secondary goal, several other challenges will be tackled, viz: (i) inferring computational capabilities of the devices in the network, (ii) detecting non-availability and using alternative computation resources, and (iii) exploring different alternatives of leveraging current IoT devices, including using firmwares that allow the execution of programs written in Lua, Javascript, Python, etc., amongst others.

1.5 Document Structure

Chapter 2 introduces the background information and explanation about concepts necessary for the full understanding of this dissertation with the use of a Systematic Literature Review on the state of the art of visual programming applied to the Internet of Things paradigm. Chapter 3 describes the state of the art regarding the ecosystem of this project's scope. Chapter 4 presents the problem this dissertation aims to solve, as well as the approach taken to solve it. Finally, Chapter 5 concludes the dissertation with a reflection on the success of the project by presenting the a summary of the contributions made and detailing the difficulties and future work.

Chapter 2

Background

This chapter describes the necessary foundations regarding visual programming tools for the Internet of Things context. Section 2.1 describes the background of the Internet of Things paradigm and important concepts in that area, with description of IoT architecture in Section 2.1.1. Sections 2.1.1.1 and 2.1.1.2 explain Fog and Edge computing concepts, respectively. Section 2.3 describes the Node-RED programming tool and its architecture and uses. Finally, section 2.2 mentions visual programming languages, their uses as well as their benefits and drawbacks.

2.1 Internet of Things

Internet of Things paradigm was defined by the committee of the International Organization for Standardization and the International Electrotechnical Commission [1] as:

> *"An infrastructure of interconnected objects, people, systems and information resources together with intelligent services to allow them to process information of the physical and the virtual world and react."*

This paradigm is built upon the network of heterogenous devices interconnected between themselves, people and the environment. According to Buuya [26], the applications of IoT systems can be divided into four categories: (i) Home at the scale of an individual or home, (ii) Enterprise at the scale of a community, (iii) Utilities at a national or regional scale and (iv) Mobile, which is spread across domains due to its large scale in connectivity ans scale.

However, one might think that IoT only relates to machines and interactions between them. Most of the devices we use in our day-to-day - mobile phones, security cameras, watches, coffee machines - are now computation capable of making moderately complex tasks and are constantly generating and sending information. This relates to the *human-in-the-loop* concept, where humans and machines have a symbiotic relationship [36].

2.1.1 IoT architectures

Internet of Things systems deal with big amounts of data from different sources and has to process it in efficient and fast ways. Typical IoT systems are composed of three layers or tiers, which are:

- **Cloud Layer**, which is composed of data centers and servers, normally running remotely. It is characterized by having high computation power and latency.
- **Fog Layer** is composed of gateways and devices that are normally between the cloud servers and the edge devices. This layer has less latency than the cloud, more heterogeneity and geographical distribution.
- **Edge Layer** contains all the edge devices (sensors, embedded systems, light sources, etc). Since its devices have smaller computational capabilities, this layer is the one with smaller computation power but with the less latency value.

These layers can also be called Application Layer, Network Layer and Perception Layer [34], respectively, which is compatible with the characterizing mentioned above. An illustrative representation can be seen in Figure ...

INSERT HERE PRETTY PICTURE WITH LAYERS

New paradigms of computing appeared related to each of these layers. The majority of IoT systems use a Cloud Computing architecture, where it takes advantage of centralized computing and storage. This approach has several benefits, such as increased computational capabilities and storage, as well as easier maintenance. However, it comes with several problems such as (a) high latency and (b) high use of bandwidth, due to the need to send the data generated from the sensors to the centralized unit [29]. Systems that only use cloud computing are not scalable, specially real-time applications, which are sensible to increased latency. With the increasing computation capabilities of edge devices and the requirements of reduced latency, two new paradigms appeared - Fog and Edge Computing.

2.1.1.1 Fog Computing

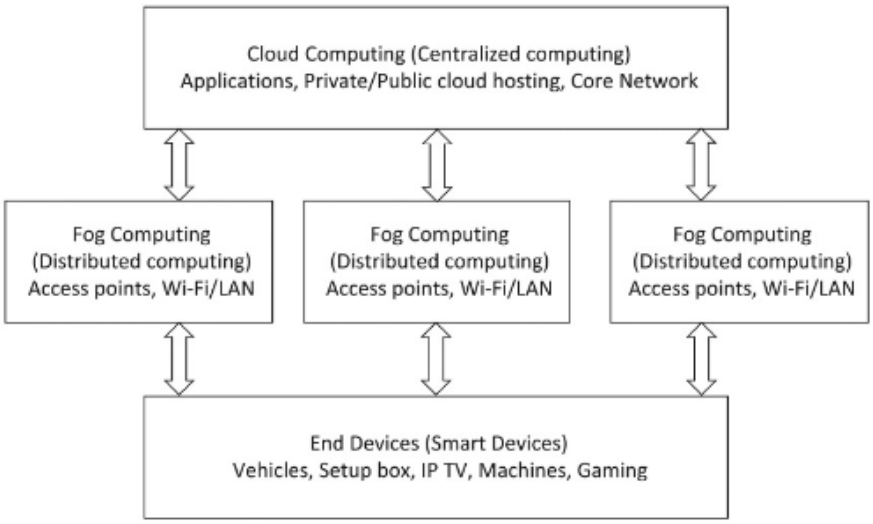
Nowadays, with the improvement of wireless networks and the hardware and software of mobile devices, there is a possibility to take advantage of this variables in the computational execution of IoT systems. This will allow for devices in the network to communicate and share resources between them, reducing latency. The central instance, which in the paradigm before executed all the computation, now serves as a scheduler and state manager of the communication between the devices, occasionally providing necessary resources if needed. The paradigm described before is called Fog Computing, which aims to bring computing closer to the perception layer, extending the cloud closer to the edge of the network [30]. It focuses on distributing data throughout the IoT system, from the cloud to the edge devices, making the system a distributed one.

According to Buuya [14], Fog Computing has several advantages: (i) reduction of network traffic by having edge devices filtering and analyzing the data generated and sending data if necessary, (ii) reduced communication distance by having the devices communicate between them

without using the cloud as middleman, (iii) low-latency by moving the processing closer to the data source instead of sending the data to the cloud to be processed, and (iv) scalability by reducing the burden on the cloud, which could be a bottleneck for the system.

It is possible to see an example of the architecture of a IoT system using the Fog paradigm in image 2.1. The Fog Computing connects the cloud to the edge devices, normally with the use of access points and gateways.

Figure 2.1: Fog Computing Architecture [14]



Despite all the advantages, Fog Computing has several requirements and difficulties. In order to make a successful and efficient distribution of computation and communication, it requires knowledge about the resources of the connected devices. The complexity is also bigger than Cloud Computing due to the fact that it needs to work with heterogenous devices with different capacities.

2.1.1.2 Edge Computing

Edge Computing is a distributed architecture that uses the devices computational power to process the data they collect or generate. It takes advantage of the Edge layer, which contains the devices closer to the end user - smartphones, TVs, sensors, etc. This paradigm goal is to minimize the bandwidth and time response of IoT systems while leveraging the computational power of the devices in them. It reduces bandwidth usage by processing data instead of sending it to the cloud to be processed, which is also correlated to reduced latency, since it does not wait for the server response. In addition to these advantages and related to their cause, Edge Computing also prevents sensitive data from leaving the network, reducing data leakage and increasing security and privacy [31, 43].

In this paradigm, each device serves both as a data producer and a data consumer. Since each device is constrained it terms of resources, this brings several challenges such as system reliability

and energy constraints due to short battery life and overall security. Other issues consist in the lack of easy-to-use tools and frameworks to build cloud-edge systems, inexistent standards regarding the naming of edge devices and the lack of security edge devices have against outside threats such as hackers [42].

2.2 Visual Programming Languages

Visual Programming, as defined by Shu, consists of using meaningful graphical representations in the process of programming [44]. With this definition, we can consider Visual Programming Languages (VPLs) as a way of handling visual information and interaction with it, allowing the use of visual expressions for programming. According to Burnet and Baker [13], visual programming languages are constructed in order to *"improve the programmer's ability to express program logic and to understand how the program works"*.

There are several applications of visual programming languages in different areas, such as education, video game development, automation, multimedia, data warehousing, system management and simulation, with this last area being the area with most use cases [40].

Visual programming languages have several characteristics, such as a concrete process and depiction of the program, immediate visual feedback and requires the knowledge of fewer programming concepts (e.g. pointers, memory allocation, etc) [13].

VPLs were categorized by Downes [12] based on their visual paradigms and architecture:

- **Purely Visual Languages**, where the creation is made using only graphical elements and the subsequently debugging and execution is made in the same environment.
- **Hybrid text and visual systems**, where the programs are created using graphical elements but their executions is translated into text language.
- **Programming-by-example systems**, where a user uses graphical elements to teach the system.
- **Constraint-oriented systems**, where the user translates physical entities into virtual objects and applies constraints to them, in order to simulate their behavior in reality.
- **Form-based systems**, which were based in the architecture and behavior of spreadsheets.

The categories mentioned can be present in a single system, making them not mutually exclusive.

2.3 Node-RED

Node-RED[1] is a programming tool applied to the development of Internet of Things systems. It was first developed as a proof-of-concept for visualizing and manipulating mappings between

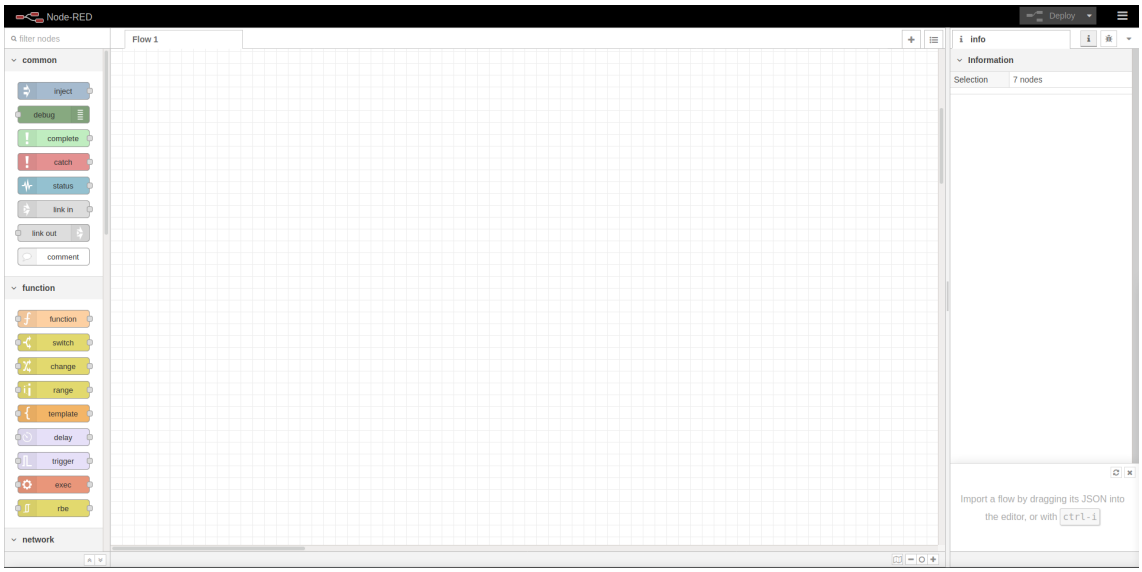
[1] https://nodered.org/

MQTT topics in IBM's Emerging Technology Services group. It then expanded into a more general open-source tool, which is now part ot the JS Foundation.

It is a web-based tool consisting of a run time built with the Node.js framework and a browser-based visual editor. This tool provides the end user with a simple interface to connected devices and APIs, using a flow-programming approach. Programs are called *flows*, built with *nodes* connected by wires. Each node correspond to a action, such has input, output, data processing, etc.

The Node-RED interface has three components: (1) Palette, (2) Workspace and (3) Sidebar. The Palette contains all the nodes installed and available to use, divided into categories. They can be used by dragging them into the workspace and additional features for each node are accessible by double-clicking them. The Workspace is where the flows are created and modified. It is possible to have several *flows* and *subflows* accessible with the use of tabs. Lastly, the Sidebar contains information about the nodes, the debug console, node configuration manager and the context data. Figure 2.2 showcases the visual interface of Node-RED and its elements.

Figure 2.2: Node-RED environment

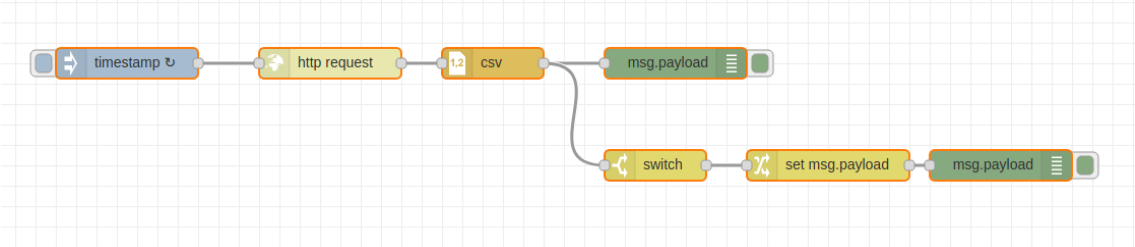


One example of a *flow* can be seen in picture 2.3, where a request is being made in intervals of 5 minutes to a HTTP URL that returns a CSV with the feed of significant earthquakes in the last 7 days. The data from the CSV is then printed to the debug console and, if the magnitude is equal or bigger than 7, the message "PANIC!" is printed to the console.

Regarding the architecture of Node-RED, the `Node` base class is a subclass of Node.js event APIs `EventEmitter`. This class implements an observer design pattern that maintains a subscriber list of all the nodes connected to it by *wires* and emits events to them. When a node finishes processing data from external sources or from another node, it calls the methods `send()` with a Javascript object. In its turn, this method call the `EventEmitter emit()` method that sends named events to the subscribed nodes.

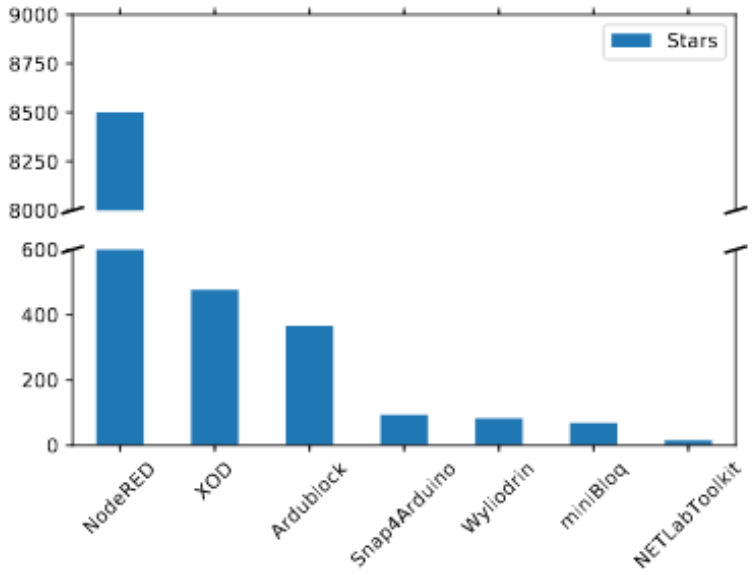
Being open-source, Node-RED takes advantage of a large community that contributes with

Figure 2.3: Example of a Node-RED flow



new nodes and improvements to the tool. It is the most popular open-source visual programming tool for IoT, as seen in figure 2.4. *ADD ARTICLE "Visual Runtime Verification and Self-Healing of IoT Systems" TO THE BIB*

Figure 2.4: Number of stars on GitHub for visual programming tools applied to IoT



2.4 Summary

This chapter introduces two areas that are fundamental for the understanding of this dissertation. Internet of Things is defined, as well as its use cases and categories. Fog and Edge computing paradigms are explained, which will be mentioned throughout this document. Node-RED is introduced as a visual programming tool for IoT and its architecture is explained. Finally, a definition and categorization of visual programming languages is introduced and explained.

Chapter 3

State of the Art

TEMP

This chapter describes the state of the art in visual programming tools in Internet of Things context, as well as decentralized methods of work distribution in flow-based architectures. Section 3.1 presents a systematic literature review on the topic of visual programming tools applied to the Internet of Things paradigm, which aims to answer the research questions defined in section 3.1.1.1. Section 3.1.2 ...

3.1 Systematic Literature Review

A Systematic Literature Review was made to gather information on the state of the art of visual programming applied to the Internet of Things paradigm. The goal of a systematic literature review is to synthesize evidence with emphasis on the quality of the it [39].

3.1.1 Methodology

During this Systematic Literature Review, a specific methodology was followed to reduce bias and produce the best results [39]. We started by defining the research questions to be answered as well as choosing data sources to search for publications.

3.1.1.1 Research Questions

REVIEW

In this Systematic Literature Review we intent to answer the following questions:

RQ1: How did Visual Programming Languages and Internet of Things evolve over time? Internet of Things is a paradigm with several years, but in the last few years it has been increasing

in its applications, specifically with its integration with visual programming tools and environments. It is important to analyze the evolution of these concepts and their integration, to be able to compare with the state of the art.

RQ2: Methodologies implemented in Internet of Things with Visual Programming Languages? With the integration of visual programming tools with Internet of Things, several methodologies were implemented for it to be possible and provide users with a better experience.

RQ3: What is the maintenance and resilience of a Visual Programming Language integrated with an IoT system? Visual programming tools provide users a easy way of programming, with the use of visual elements and relations between them. However, this approach has downsides, such as difficulty in constructing and maintaining complex systems and high level programming that undermines efficiency and resilience.

3.1.1.2 Databases

The publications retrieved during this research were retrieved from the following databases, which are considered good and reliable sources:

- IEEE
- ACM
- Scopus

3.1.1.3 Search Process

To obtain results from the databases chosen, a research question was written with the union of the keywords "visual programming", "node-red", "dataflow" and intersection with the keyword "Internet of Things".

```
((vpl OR visual programming OR visual-programming) OR (node-red OR node red OR
    nodered) OR (data-flow OR dataflow)) AND (IoT OR internet of things OR
    internet-of-things)
```

The search was performed in October of 2019 and the results produced are the ones present in the table 3.1.

Table 3.1: Systematic Literature Review search results per database

Database	Total Results	Extracted Results
IEEE	410	379
ACM	171,768	2021
Scopus	540	500

3.1.1.4 Inclusion Criteria

To be included in the results, all publications should respect the inclusion criteria. If one of the criteria were not checked, the publication would not be included in the results. The inclusion criteria are the following:

1. On the topic of visual programming in internet of things;
2. Includes sufficient explanation of the research findings;
3. Publication year in the range between 2008 and 2019.

3.1.1.5 Exclusion Criteria

In addition to the inclusion criteria, all publications were analyzed in their compliance to the exclusion criteria. If any publication failed to comply with at least one of the exclusion criteria, it would not be included in the results. The exclusion criteria are the following:

1. Has less than two (non-self) citations when more than five years old;
2. Presents just ideas, tutorials, integration experimentation, magazine publications, interviews or discussion papers;
3. Not in English.

3.1.1.6 Quality Assessment

In order to classify if a publication is relevant to the research field, 4 assessments were made in order to better facility the process. The quality assessments are the following:

Table 3.2: Parameters for measuring the quality of a publication

Quality Assessment Query	**Quality Indicator (0-2)**
Is the publication relevant to us?	BARELY-PARTIALLY-SATISFACTORILY
Does the publication include and define research objectives adequately?	NO-PARTIALLY-YES
Are limitations and challenges well defined?	NO-PARTIALLY-YES
Is the proposed contribution well described?	NO-PARTIALLY-YES

Each assessment was posed in the form of a questions, and to each question there were three possible answers, with a numeric value each. If a publication didn't address the assessment the value with be 0, if the assessments was partially addressed the value would be 1. If the assessment was successfully satisfied, the value would be 2. In the end, the sum of all the assessments would represent the quality of the publication.

3.1.1.7 Evaluation Process

The evaluation process of the publications followed six steps with specific purposes:

1. **Range:** Publications are evaluated on date range, between 2008 and 2019;
2. **Relevance:** Title and abstract are scanned for relevance regarding the defined research field;
3. **Inclusion:** Publications are assessed against inclusion and exclusion criteria. Any publications not meeting the full inclusion criteria are discarded as well as all publications failing to comply to any exclusion criteria;
4. **Specificity:** Reading the publication to verify if it relates closely enough to the defined research field;
5. **Data:** Selected publications are analyzed for data related to the research questions and contribution details;
6. **Publication quality:** Publications are assessed using quality criteria defined in Table 3.2.

The results from the evaluation process can be seen in Table 3.4.

Table 3.3: Publications per step

Step	Nº of publications	Nº of excluded publications
Search	2698	N/A
Duplicates	2626	72
Exclusion/Inclusion criteria (Titles and Abstracts)	65	2561
Exclusion/Inclusion criteria (Introduction and Conclusion)	29	36

3.1.2 Results

After analyzing the 29 publications, we organized them by categories. [47], [40] and [32] were surveys and the remaining 26 were frameworks or tools.

Regarding surveys, publication [47] investigates the use of trigger-action (*IFTTT*) programming in the customization of smart-home devices by non-developer users. It produced positive results in the way end users received and used this programming paradigm. [40] makes an indept review of 13 visual programming languages in the field of IoT, comparing them under four attributes: (1) programming environment, (2) license, (3) project repository and (4) platform support. The author concluded with some advantages of using visual programming languages, such as the ease of visualizing programming logic, useful for rapid development and less burden on handling syntax error. However, some negative aspects were also mentioned, being the large amount of time building simple IoT applications the most important one. Publication [32] proposes an hypothesis that flow-based programming will be better than block-based programming as a visual programming language for users with no previous programming experience. The results were inconclusive but the stated problem was interesting, seeing as the future of programming can involve

casual users with little programming experience, who use visual programming languages to solve problems, similar to the use of spreadsheets nowadays.

The remaining 26 articles are frameworks or tools of visual programming applied to IoT. One of the tool is repeated in two papers, which showcases its evolution. The frameworks are:

1. Solution for connecting devices from different IoT platforms, using Flow Based Programming with Node-RED [8], proposed by **Belsa et al**. Its motivation is based on the limitation imposed by the IoT platform on communication between components and extensibility. This hinders possibilities to interact with services provided by other platforms. To validate their solution, they implemented a use case in the domain of transportation and logistics, with a composed service that used five different types of applications. The developed tool offers access to available services in a centralized visual framework, where end-users can use them to build more complex applications.

2. **Ivy** [20] proposes the next step forward regarding visualization applied to IoT with a visual programming tool that uses immersive virtual reality to allow users to link devices, insert logic and visualize real-time data flows between real-world sensors and actuators. It provides the end users with an immersive virtual reality that allows them to visualize the data flow, access to debugging tools and real-time deployment. Each programming construct called node - data flow architecture - has a distinct shape and color, which makes it easier for the user to understand the system being built or debugged. The experiences made in order to validate the prototype were positive, with the participants being receptive to Ivy and indicating use cases for it.

3. With the rise in number of devices and their applications, it is impossible for developers to predict the way end users will exploit the devices, how they will be arranged and for which objectives will they be used. The goal of this paper [24], proposed by **Ghiani et al.**, is to build a set of tools that allow non-developer users to customize their own Web IoT applications with the use of trigger-actions rules. The proposed solution provides a web-based tool for specifying trigger-action rules using *IFTTT* and a context manager middleware that is able to adapt to the context and events of the devices and apply rules to the system. In order to validate the developed tool, an example home automation application that displays sensor values and directly controls appliances was built. The results were for the most part positive, and the issues found are related to usability and visual clues.

4. **ViSiT** [2] allows its end-users to use a jigsaw puzzle metaphor to implement a system of connected IoT objects. It provides a web-based visual tool connected with a web-service that generates an executable implementation of the jigsaw representation and transformation. Their goal is achievable by adapting model transformations used by software developers into understandable metaphors for non-developers to use. They validated the developed tool with a usability evaluation, which was overall positive, with a great percentage considering the tool useful and providing real-life scenarios where they could implement it.

5. A framework for Ambient Assisted Living (AAL) using IoT technologies is proposed by **Valsamakis and Savidis** [48], which allows for customized automation. It uses visual programming languages to facilitate their end users - carers, family, friends, elderly - to build and modify the automations. They built a visual programming framework that introduces smart objects grouping in tagged environments and real-time smart-object registration through discovery cycles. It runs on typical smart phones and tablets and is built in Javascript, allowing it to run in browsers. Their future work focuses on integrating different visual programming paradigms to fully accomplish the requirements of the end-user.

6. **WireMe** [38] is an intuitive solution for building, deploying and monitor IoT systems, built with non-developer end users in mind but also extensible for advanced users to built over it. The developed solution makes use of Scratch, a visual programming interface, to provide its users with a customizable dashboard where they can monitor and control their IoT system as well as program automation tasks. It has a Main Control Unit responsible for communicating the devices status to the dashboard via MQTT, which is programmable using their visual interface and Lua programming language. Their tool was validated by students around 16 years old and engineering students without programming experience. The results were not totally positive, with some students not being able to create the required simple logic. Future work consists improving programming blocks to become more intuitive.

7. **VIPLE** [18], Visual IoT/Robotics Programming Language Environment, is a new visual programming language and its correspondent visual environment. It provides an introduction to topics such as computing and engineering and tools for more practical domains like service-oriented computing and software integration. It focuses on complex concepts such as robot as a service (Raas) units and Internet of Intelligent Things (IoIT), while studying the programming issues of building systems classified as such. The developed tool is extremely powerful and has been tested and used in several universities since 2015. CHECK THIS ONE, kinda bad

8. **Smart Block** [7] is a block-based visual programming language and visual programming environment applied to IoT systems, that allows non-developer users to build their own systems in an easier way. Their solution is specific to the home automation domain, like Smart Things. The language was designed using IoTa calculus, used to generalize Event-Condition-Action rules for home automation. The environment was built using Blockly, a client-side Javascript library for creating visual block languages. Future work for this project consist of expanding custom blocks for features such as device grouping and security, as well as extending the tool for other domains besides home automation.

9. **PWCT** [23] is a visual programming language applied to building IoT, Data Computing and Cloud Computing systems. Its goal consist of reducing the cost of development of these types of systems by providing an easy and more productive development tool. The language was designed to compete with text based languages such as Java and C/C++. It

uses graphical elements to replace textual code and has 3 main layers: (1) the VPL layer, composed of graphical elements, (2) the middleware layers, responsible for connecting the VPL layer with the system's view, which is the (3) System Layer, responsible for dealing with the source code generated by the first layer. The created solution received positive feedback from the community, with more than 70,000 downloads and 93% of user satisfaction.

10. **DDF** [25] is a Distributed Dataflow (DDF) programming model for IoT systems, leveraging resources across the Fog and the Cloud. They implemented a DDF framework extending Node-RED, which originally is a centralized framework. Their motivation comes from the possibility to develop applications from the perspective of Fog Computing, leveraging these devices for efficiency and reduced latency, since there is a big amount of resources such as edge devices and gateways in IoT systems. They evaluated their prototype using a small scale evaluation, which was positive. The results showed that their DDF framework provides an easy alternative for designing and developing distributed IoT systems, despite having some open issues such as not having a distributed discovery of devices and networks.

11. **GIMLE** [46], Graphical Installation Modelling Language for IoT Ecosystems), is a visual language that uses general-purpose visual programming styles to model domain knowledge through expressive ontological requirements. The goal of this language is to fill the gap of modelling requirements on physical properties of IoT installations by proposing a novel process for configuring industrial installations. It makes use of flow-based and domain-based visual programming in order to separate the logical flow of the requirements from their details. The developed tool supports reuse within the models, which is useful due to the repetitive nature of industrial installations,but it still needs to clarify how it fits within the current practice and its use in production settings.

12. The approach on Runtime Verification proposed by **Leotta et al.** [28] aims to assure the quality of an IoT system. It can be used for detecting bugs in systems during development and maintenance activities, as well as be adopted for monitoring deployed IoT systems. Runtime verification is a software analysis approach in which a running system is observed by monitoring relevant events and their associated information to verify against a given specification of the expected behavior. When efficiently implemented, the verification process can be executed before the deployment of the system, for detecting bugs during the development and maintenance activities, as well as after the deployment, in order to provide an additional level of protection against unforeseen events. The results of the runtime verification tool developed were positive, with 92 % detection of the bugs injected into the system. For future work, they plan to automate the generation of trace expressions and scenarios, as well as compare their approach with testing.

13. **DDFlow** [35] is a macro-programming abstraction that aims to provide efficient means to program high quality distributed apps for IoT. The authors refer a lack of solutions for complex IoT systems programming, causing developers to build their own systems, which leads to a lack of portability/extensibility and results in a lot of similar systems that do the same thing, but are "different" because they were created by different programmers. Developers use Node-Red to specify the application functionalities and DDFlow handles scalability and deployment. The authors describe DDFlow's goal as to allow developers to formulate complex applications without having to care about low-level network, hardware and coordination details. This is done by having the DDFlow accompanying runtime dynamically scaling and mapping the resources, instead of the developer. DDFlow gives developers the possibility to inject custom code on nodes and have custom logic, if the available nodes are not enough for some task.

14. The tool proposed by **Kefalakis et al.** [27] consists of a visual environment that operates over the OpenIoT architecture and facilitates the development of IoT applications with minimal programming effort. Modeling IoT services with the developed tool is made by specifying a graph that corresponds to an IoT application, which can be validated and its code generated and enacted over the OpenIoT middleware platform. It aims to fill the gap of tools that provide support for the development and deployment of integrated IoT applications. The approach taken presents several advantages: (1) it leverages standards-based semantic model for sensor and IoT context, making it easier to be widely adopted, (2) it is based on web-based technologies which opens the possibilities of applications from developers and (3) it is open source.

15. The approach presented by **Eterovic et al.** [21] proposes an IoT visual domain specific modeling language based on UML, with technical and non-technical users in mind. The authors defend that, with the evolving nature of IoT, the future end user will be a common person, with no programming knowledge. To solve the problems this future brings, it is important to build a visual language easy enough to be understood by non-technical people but expansible enough to represent complex systems. To evaluate the proposed solution, they invited 11 users of different levels of UML expertise to model a simple IoT system with the developed language. The System Usability Score was positive, as well as the Tasks Success Rate. Despite the positive score, some future actions would be the testing of the language with a more complex task as well as the integration of advanced UML notations.

16. **FRED** [11] is a Frontend for Node-RED, a development tool that makes it possible to host multiple Node-RED processes. It can be used to connect devices to cloud services, coordinate communication between devices, integrate services with each other or creating new web app APIs and applications. To provide all these features, FRED supports the ability to run flows for multiple users and all flows get fair access to CPU, memory and storage resources. It also provides secure access to flow editors and the flow runtime. The authors

concluded that FRED is a useful tool for users learning about Node-RED and to rapidly prototype cloud-hosted applications.

17. **WoTFlow** [10] is proposed as a cloud-based platform that aims to provide an execution environment for multi-user cloud environments and individual devices. It aims to take advantage of data flow programming, which allows parts of the flow to be executed in parallel in different devices. Based on this, the tool will take advantage of the ability to split and partition the flows and distribute them by edge devices and the cloud. The state of the developed tool was in the early stages, with future expansions based on the use of optimization heuristics, automatic partitioning based on calculated constraints, security and privacy.

18. The approach on runtime monitoring of NodeJS applications, proposed by **Ancona et al.** [6], makes use of the dynamic analysis framework Jalangi to detect errors that would be otherwise difficult to catch with other techniques. This paper investigate the use trace expressions in runtime monitoring, which are language and system agnostic through the notion of event domain and type. To implement runtime monitoring, the authors instrument the source code of the program that needs to be verified, adding a piece of code that is able to capture all the relevant events for the domain in use. On the other hand, they have a Prolog server implementing the operational semantics of trace expressions and offering a simple REST interface. The monitor and the server communicate through HTTP requests and responses, effectively implementing a runtime verification system. Future work consist of expanding the current features of logging and error detection into error recovery procedures.

19. **Devify** [16] is a framework that implements a new software architecture based on a peer-to-peer network and interoperability between IoT devices. This framework is divided in three layers: Application Logic Layer, Broker Server Layer and the Web of Things Layer. The application layer provides a flow-based programming like runtime engine with an unidirectional data flow design. However, the flow-based component has only a single output port and single input port. The broker server implements the peer-to-peer networking, REST-style remote procedure call operations and a distributed hash table. Additionally, the Web of Things layer can make distribution possible by providing a service contract without exposing server-side implementation details. In conclusion, the authors propose a decentralized IoT software framework that provides the ability of secure data exchange between IoT devices autonomously without any centralized server. To achieve security wanted, they used Flowchain, a blockchain-based decentralized IoT platform developed by them.

20. An extension to the tool Orcc[1], by **Paller et al.** [37], that adds new features that are specific for the IoT problem domain to a tool dedicated to dataflow programming. The features added to Orcc, a tool that generates source code from RVC-CAL actors and XDF networks,include the improvements on the graphical Data-Flow Language (DFL), novel

[1] https://github.com/orcc/orcc

code generator for heterogeneous platforms and the library of ready-made IoT components or actors. The main goal was to extend the XDF file to address the platform heterogeneity. Furthermore, the utilization of this tool in development is expected to increase the quality, to reduce the development costs of IoT applications. In conclusion, this is an ongoing research but it has a lot of manual work like creation of the XCF configuration file with the smart environment needs. review this, its super confusing

21. An IoT-based GUI, proposed by **Besari et al.** [41] [9], that aims to control sensors and actuators in an IoT system using an android application, in which the users used a visual programming language to configure and interact with the IoT system. The system was tested with a Pybot, which is a robot that is programmable similarly to an IoT system, with sensors and actuators. After testing and evaluating the system, the authors came to a score of 72.917 (out of 100) for the Pybot software, which is considered "GOOD". The overall acceptability of the system was "ACCEPTABLE", which led the authors to consider the application accepted by users.

22. **CharIoT** [45] is an end-user programming environment that promises to unify and support the configuration of IoT environments. It provides three blocks of support: capturing higher-level events using virtual sensors, construction of automation rules with a visual overview of the current configuration and support for sharing configuration between end users using a recommendation mechanism. To enable the capturing of higher-level events, it was developed two types of virtual sensors. The programmed virtual sensor provides a more accessible and understandable abstractions (defining that a room is "cold" if temperature is below 20^{o}C). The demonstrated virtual sensors are more complex, requiring the user to provide demonstration of the occurrence and not occurrence of the event (for example, the event of someone knocking on the door and the absence of someone knocking on the door). This last one requires the training of a Random Forest classifier. This programming environment is similar to IFTTT but goes one step further, with smarter event capturing and reusing of configurations, allowing the end-user to build faster and more robust IoT installations.

23. A new technology, proposed by **Desolda et al.** [19], using a tangible programming language which allows non-programmers to configure the behavior of the smart objects in order to create and customize the smart environments. The main goal was to create, with the developed technology, a scenario of a smart museum. The authors defend that a personalization of a smart environment cannot be limited by the synchronization of smart devices and it may require experts to build the narrative of them, much like a museum said that. With this in mind, they introduced custom attributes to assign semantics to involved objects, in order to empower and simplify the creation of event-condition-action rules. In conclusion, this is an ongoing research focus on developing a new technology with an interaction paradigm to allow domain experts in the creation of smart environments. In addition, the fact that this technology uses expensive material (tabletop surface as digital workspace) doesn't allow a regular user to use it as stated in the introduction.

24. An End User Development (EUD) tool, proposed by **Eun et al.** [22], that allows end users to develop their own personal applications. It uses the dataflow approach, which allows for a more generalized programming experience as well the facility to build more complex programs with simple modules. The proposed tool has three main components: Service Template Authoring Tool, Service Template Repository and Smartphone Application. The first one allows for the end user to build more complex methods using atomic templates (components with simple functionality, like opening a curtain if it receives a command). The Service Template Repository contains the proprietary atomic templates as well as ones built by the user. Lastly, the Smartphone Application runs and manages the applications built by the user, as well as their requirements and dependencies. The developed EUD tool was compared with *IFTTT* and Zapier, other tools focused on end user development. *IFTTT* and the developed tool are more similar, focusing on consumer development, IoT and Home, with Zapier focusing on business. Both Zapier and *IFTTT* use the Triger-Action paradigm (TAP), which differs from the dataflow paradigm used in this paper's tool.

25. **RHEA** [33] is a framework whose purpose is to offer a unified extensible way for reactive applications to be developed with integration between different sources and middlewares, for example MQTT and ROS (Robot operating system). This is done by using a dataflow model. A user writes a program that will be translated into a dataflow graph. Using that graph, the framework will optimize parts of it and assign them to nodes (physical computation resources), which will be distributed across the available machines for execution. It also handles requirements like node placement, where specific nodes need to be placed in specific machines with specific skills. This framework can be useful in integrating data and building complex applications that integrate IoT and Robotics in a easy way. It facilitates the connection between different modules with several requirements and excuses the user of having to deal with asynchronous callbacks. It differentiates itself from other existing dataflow frameworks by allowing the use of other systems to execute tasks, having support for heterogeneity of hardware and the reactivity aspect of it.

The mentioned frameworks and tools were divided into the following categories, according to several characteristics:

Scope Some tools have specific use cases in mind (*e.g.* smart cities, Home automation, industry, etc). Therefore, knowledge of the scope of a tool is useful to assess if it solves a problem or fills a specific gap in the literature. Example values consist of *smart cities, home automation, education, industry* or *many*, if there is more than one.

Architecture Visual programming tools applied to the Internet of Things can have an centralized or decentralized architecture, based on their use of Cloud, Fog or Edge Computing architecture. Possible values are *Centralized*, *Decentralized* and *Mixed*.

License The license of a software or tool is essential in terms of its usability. Normally, an open-source software reaches a bigger user base and allows them to expand and contribute to it. Possible values are the name of the tool license or N/A if it does not have one.

Tier IoT systems, as explained in Section 2.1.1 is composed of three tiers - *Cloud*, *Fog* and *Edge*. A tool can interact in several of these tiers, which shapes the features it contains and how it is built.

Scalability Defines how the tool or framework scales. It can be calculated based on metrics used to test the performance of the system. Possible values are *low*, *medium*, *high* or N/A, it case there is no sufficient information.

Programming According to Downes and Boshernitsan [12] and also mentioned in Section 2.2, visual programming languages can be classified in five categories: (1) Purely Visual languages, (2) Hybrid text and visual systems, (3) Programming-by-example systems, (4) Constraint-oriented systems and (5) Form-based systems. These classifications aren't mutually exclusive. It is important to know which type, so that might be possible to assess the type of experience the tool provides to the user and its architecture.

Web-based Defines if the visual programming language and/or environment can be used in a browser. It is useful in terms of usability of the tool.

Table 3.4: VPLs applied to IoT and their characteristics. Small circles (•) mean *yes*, hyphens (-) means *no information available*, empty means *no* and asterisk (*) means more than one.

Tool	Scope	Architecture	License	Tier	Scalability	Programming	Web-based
Belsa et al.[8]	*	Centralized	-	TODO	High	TODO	•
Ivy [20]	*	Centralized	-	TODO	High	TODO	
Ghiani et al. [24]	Home Automation	Centralized	-	TODO	TODO	TODO	•
ViSiT [2]							
Valsamakis and Savidis [48]							
WireMe [38]							
VIPLE [18]							
Smart Block [7]							
PWCT [23]							
DDF [25]							
GIMLE [46]							
Leotta et al. [28]							
DDFlow [35]							
Kefalakis et al. [27]							
Eterovic et al. [21]							
FRED [11]							
WoTFlow [10]							
Ancona et al. [6]							
Devify [16]							
Paller et al. [37]							
Besari et al. [41] [9]							
CharIoT [45]							
Desolda et al. [19]							
Eun et al. [22]							
RHEA [33]							

WIP explicar caracteristicas de classificação(DONE); fazer o quadro com tudo organizado;

3.1.3 Expanded Search

TODO verificar os surveys e tirar as tools de lá tmb

3.1.3.1 Expanded Results

3.1.4 Analysis and Discussion

TODO

3.1.4.1 Result Analysis

TODO Organizar os artigos por categorias?

3.1.4.2 Research Questions

Responder às research questions com os resultados

3.1.5 Conclusions

TODO

3.2 Decentralized Architectures in Visual Programming Tools applied to the Internet of Things paradigm

Colocar aqui os resumos sobre os artigos que li sobre decentralização, mais específicos ao meu tema Ainda nao sei como organizar esta parte

3.3 Summary

TODO

Chapter 4

Problem Statement

This chapter describes the problem, as it can be seen in Section 4.1. In Section 4.2 it is presented the wanted features for the proposed solution and in Section 4.3 the scope of the project is defined. Section 3.1.1.1contains the research questions to be answered by this dissertation. The experimental methodology is outlined in Section 4.5. Chapter 4.6 contains a Gantt chart with a planning of this dissertation. Finally, this chapter is summarized by Section 4.7 with an overview of the topics mentioned before.

4.1 Current Issues

Chapter 3 contains several solutions that provide decentralized architecture in visual programming tools applied to the internet of things paradigm. However, some of this tools are specific to a certain paradigm, like Smart Cities or industry. Check this after SOTA We can define the problem in these issues:

1. **Discovery of computation capabilities**: the current work lacks the automatic discovery of the computational capabilities of the devices in the network. This information is normally manually introduced by the developer.

2. **Leveraging devices in the network**: since most tools use a centralized architecture, including Node-RED, they do not leverage the devices in the network. Fog Computing introduces a decentralized solution, one that can be applied to Node-RED by distributing the computational tasks across the edge devices.

3. **Inferring computational capabilities**: current tools require the developer to manually introduce the resources of each device in the network, which is not a scalable solution. This information is vital for the successful distribution of computation across the devices.

4. **Detecting non-availability**: when a device fails or becomes unavailable, it is important for the system to automatically realize and adapt. The majority of current solutions do not

possess this feature, which is vital if a system aims to dynamically adapt to changes in the environment.

4.2 Desiderata

Desiderata is a Latin word that translates to "*things wanted*". In the context of this document, this section contains requirements wanted in a solution that aims to solve all the issues identified in Section 4.1. The requirements are the following:

D1: Infer computational capabilities of devices connected so that this information can be sent to an orchestrator that will decompose the total computation workload based on this data.

D2: Decomposition and partition of the computation so that the total computational requested can be distributed through all the devices in the network, using information about the computational capabilities and availability of the devices in the network.

D3: Convert computational tasks into runnable code so that each computational task can be executed in edge and fog devices, which contain limited resources.

D4: Provide self-adaptation of the system so that it can adapt to non-availability of resources or even appearances of new devices.

4.3 Scope

The focus of this dissertation is the development of a prototype that allows for a decentralized orchestration of an IoT system. Despite security being a critical feature, it is considered a secondary goal, allowing the dissertation to focus on the its primary goals.

O que coloco mais?

4.4 Research Questions

Given the scope of the project defined in Section 4.3 and the current state of the art in decentralized solutions for visual programming environments applied to IoT, this dissertation's research questions are the following:

RQ1: Is a decentralized system more efficient than a centralized one?

Are there more? How to I explain this one?

4.5 Experimental Methodology

In the interest of validating whether or not the solution implemented achieves the *desiderata* and solves the current issues, there will be developed two test scenarios, the first one in a smaller scale and the second in a bigger one, with the use of simulations. Each one of the test scenarios will be verified against each *desiderata*.

Check if ok

4.6 Planning

TODO Gantt chart

4.7 Summary

TODO

Chapter 5

Conclusions

TODO

5.1 Expected Results

TODO

References

[1] ISO/IEC JTC 1. Internet of things (iot) - preliminary report. *ISO,Tech. Rep.*, 2014.

[2] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. Visual simple transformations: Empowering end-users to wire internet of things objects. *ACM Transactions on Computer-Human Interaction*, 24(2):10:1—-10:43, apr 2017.

[3] S. A. Al-Qaseemi, H. A. Almulhim, M. F. Almulhim, and S. R. Chaudhry. Iot architecture challenges and issues: Lack of standardization. In *2016 Future Technologies Conference (FTC)*, pages 731–738, Dec 2016.

[4] Tanweer Alam. A reliable communication framework and its use in internet of things (iot). 3, 05 2018.

[5] Fahed Alkhabbas, Romina Spalazzese, and Paul Davidsson. Iot-based systems of systems. *Proceedings of the 2nd edition of Swedish Workshop on the Engineering of Systems of Systems (SWESOS 2016)*, 2016.

[6] Davide Ancona, Luca Franceschini, Giorgio Delzanno, Marina Leotta, Marina Ribaudo, and Filippo Ricca. Towards runtime monitoring of node.js and its application to the internet of things. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*, volume 264, pages 27–42, 2018.

[7] Nayeon Bak, Byeong Mo Chang, and Kwanghoon Choi. Smart Block: A Visual Programming Environment for SmartThings. In *Proceedings - International Computer Software and Applications Conference*, volume 2, pages 32–37, 2018.

[8] Andreu Belsa, David Sarabia-Jacome, Carlos E. Palau, and Manuel Esteve. Flow-based programming interoperability solution for IoT platform applications. In *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, pages 304–309, 2018.

[9] Adnan Rachmat Anom Besari, Iwan Kurnianto Wobowo, Sritrusta Sukaridhoto, Ricky Setiawan, and Muh Rifqi Rizqullah. Preliminary design of mobile visual programming apps for Internet of Things applications based on Raspberry Pi 3 platform. In *Proceedings - International Electronics Symposium on Knowledge Creation and Intelligent Computing, IES-KCIC 2017*, volume 2017-Janua, pages 50–54, 2017.

[10] Michael Blackstock and Rodger Lea. Toward a distributed data flow platform for the Web of Things (Distributed Node-RED). In *ACM International Conference Proceeding Series*, volume 08-October, pages 34–39, 2014.

[11] Michael Blackstock and Rodger Lea. FRED: A hosted data flow platform for the IoT. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs, MOTA 2016*, 2016.

[12] Marat Boshernitsan and Michael Downes. Visual programming languages: A survey. 08 1998.

[13] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, and P. Van Zee. Scaling up visual programming languages. *Computer*, 28(3):45–54, March 1995.

[14] Rajkumar Buyya and Amir Vahid Dastjerdi. *Internet of Things: Principles and Paradigms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2016.

[15] S K Chang. *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing Company, 2002.

[16] Jollen Chen. Devify. *ACM SIGBED Review*, 15(2):31–36, jun 2018.

[17] 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.

[18] Gennaro De Luca, Zhongtao Li, Sami Mian, and Yinong Chen. Visual programming language environment for different IoT and robotics platforms in computer science education. *CAAI Transactions on Intelligence Technology*, 3(2):119–130, 2018.

[19] Giuseppe Desolda, Alessio Malizia, and Tommaso Turchi. A tangible-programming technology supporting end-user development of smart-environments. In *Proceedings of the Workshop on Advanced Visual Interfaces AVI*, AVI '18, pages 59:1—-59:3, New York, NY, USA, 2018. ACM.

[20] Barrett Ens, Fraser Anderson, Tovi Grossman, Michelle Annett, Pourang Irani, and George Fitzmaurice. Ivy: Exploring spatially situated visual programming for authoring and understanding intelligent environments. In *Proceedings - Graphics Interface*, pages 156–163, 2017.

[21] Teo Eterovic, Enio Kaljic, Dzenana Donko, Adnan Salihbegovic, and Samir 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*, 2015.

[22] Seongbae Eun, Jinman Jung, Young Sun Yun, Sun Sup So, Junyoung Heo, and Hong Min. An end user development platform based on dataflow approach for IoT devices. *Journal of Intelligent and Fuzzy Systems*, 35(6):6125–6131, 2018.

[23] Mahmoud S. Fayed, Muhammad Al-Qurishi, Atif Alamri, and Ahmad A. Al-Daraiseh. PWCT: Visual language for IoT and cloud computing applications and systems. In *ACM International Conference Proceeding Series*, 2017.

[24] Giuseppe Ghiani, Marco Manca, Fabio Paterno, and Carmen Santoro. Personalization of context-dependent applications through trigger-action rules. *ACM Transactions on Computer-Human Interaction*, 24(2):14:1—-14:33, apr 2017.

[25] Nam Ky Giang, Michael Blackstock, Rodger Lea, and Victor C.M. Leung. Developing IoT applications in the Fog: A Distributed Dataflow approach. In *Proceedings - 2015 5th International Conference on the Internet of Things, IoT 2015*, pages 155–162, 2015.

[26] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29, 07 2012.

[27] Nikos Kefalakis, John Soldatos, Achilleas Anagnostopoulos, and Panagiotis Dimitropoulos. *A visual paradigm for IoT solutions development*, volume 9001. 2015.

[28] Maurizio Leotta, Davide Ancona, Luca Franceschini, Dario Olianas, Marina Ribaudo, and Filippo Ricca. *Towards a runtime verification approach for internet of things systems*, volume 11153 LNCS. 2018.

[29] D. S. Linthicum. Connecting fog and cloud computing. *IEEE Cloud Computing*, 4(2):18–20, March 2017.

[30] Wei Liu, Takayuki Nishio, Ryoichi Shinkuma, and Tatsuro Takahashi. Adaptive resource discovery in mobile cloud computing. *Computer Communications*, 50:119 – 129, 2014. Green Networking.

[31] C. Martín Fernández, M. Díaz Rodríguez, and B. Rubio Muñoz. An edge computing architecture in the internet of things. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 99–102, May 2018.

[32] Dave Mason and Kruti Dave. Block-based versus flow-based programming for naive programmers. In *Proceedings - 2017 IEEE Blocks and Beyond Workshop, B and B 2017*, volume 2017-Novem, pages 25–28, 2017.

[33] Orestis Melkonian and Angelos Charalambidis. RhEA: A reactive, heterogeneous, extensible, and abstract framework for dataflow programming. In *REBLS 2018 - Proceedings of the 5th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems, Co-located with SPLASH 2018*, pages 11–20, 2018.

[34] Miao Yun and Bu Yuxin. Research on the architecture and key technology of internet of things (iot) applied on smart grid. In *2010 International Conference on Advances in Energy Engineering*, pages 69–72, June 2010.

[35] Joseph Noor, Hsiao Yun Tseng, Luis Garcia, and Mani Srivastava. DDFlow: Visualized declarative programming for heterogeneous IoT networks. In *IoTDI 2019 - Proceedings of the 2019 Internet of Things Design and Implementation*, IoTDI '19, pages 172–177, New York, NY, USA, 2019. ACM.

[36] D. S. Nunes, P. Zhang, and J. Sá Silva. A survey on human-in-the-loop applications towards an internet of all. *IEEE Communications Surveys Tutorials*, 17(2):944–965, Secondquarter 2015.

[37] Gábor Paller, Endri Bezati, Nebojša Taušan, Gábor Farkas, and Gábor Élo. Dataflow-based heterogeneous code generator for IoT applications. In *MODELSWARD 2019 - Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*, pages 428–434, 2019.

[38] D. Pathirana, S. Sonnadara, M. Hettiarachchi, H. Siriwardana, and C. Silva. WireMe - IoT development platform for everyone. In *3rd International Moratuwa Engineering Research Conference, MERCon 2017*, pages 93–98, 2017.

[39] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1 – 18, 2015.

[40] Partha Pratim Ray. A Survey on Visual Programming Languages in Internet of Things. *Scientific Programming*, 2017, 2017.

[41] Ricky Setiawan, Adnan Rachmat Anom Besari, Iwan Kurnianto Wibowo, Muh Rifqi Rizqullah, and Dias Agata. Mobile visual programming apps for internet of things applications based on raspberry Pi 3 platform. In *International Electronics Symposium on Knowledge Creation and Intelligent Computing, IES-KCIC 2018 - Proceedings*, pages 199–204, oct 2019.

[42] W. Shi and S. Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, May 2016.

[43] W. Shi, G. Pallis, and Z. Xu. Edge computing [scanning the issue]. *Proceedings of the IEEE*, 107(8):1474–1481, Aug 2019.

[44] N. C. Shu. Visual programming: Perspectives and approaches. *IBM Syst. J.*, 38(2–3):199–221, June 1999.

[45] Matúš Tomlein, Sudershan Boovaraghavan, Yuvraj Agarwal, and Anind K. Dey. CharIoT: An end-user programming environment for the IoT. In *ACM International Conference Proceeding Series*, 2017.

[46] Matúš Tomlein and Kaj Grønbæk. A visual programming approach based on domain ontologies for configuring industrial IoT installations. In *ACM International Conference Proceeding Series*, 2017.

[47] Blasé Ury, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. Practical trigger-action programming in the smart home. In *Conference on Human Factors in Computing Systems - Proceedings*, CHI '14, pages 803–812, New York, NY, USA, 2014. ACM.

[48] Yannis Valsamakis and Anthony Savidis. Visual end-user programming of personalized AAL in the internet of things. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10217 LNCS, pages 159–174, 2017.