FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Decentralized Orchestration of IoT in End-user Programming Environments

Ana Margarida Oliveira Pinheiro da Silva

PREPARAÇÃO DA DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo Sereno Ferreira

Second Supervisor: André Restivo

February 2, 2020

Decentralized Orchestration of IoT in End-user Programming Environments

Ana Margarida Oliveira Pinheiro da Silva

Mestrado Integrado em Engenharia Informática e Computação

February 2, 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

"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
CPSCN	Cyber Physical Social Computing and Networking
CPU	Central Processing Unit
IEC	International Electrotechnical Commission
ISO	Internation Organization for Standardization
IFTTT	*If This Then That*
IoT	Internet of Things
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
MTTR	Mean Time To Recover
QoS	Quality of Service
REST	Representational State Transfer
VPL	Visual Programming Language
WWW	*World Wide Web*

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 1.2 defines the problem we aim to solve. Then, SSection 1.3 explains the reason why this work and the area it belongs to is important 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 these 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 [16]. 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 [48]. Node-RED[1] is one of 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.

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.

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 [35] - i.e, data and logic should reside locally, independent of third-party services faults and errors - and NoCloud [47] - i.e, on-device and local computation should be prioritized over cloud service computation - 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] https://nodered.org/

1.3 Motivation

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, which is very important requirement 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) communicating 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. Chapter 3 describes the state of the art regarding the ecosystem of this project's scope, including a Systematic Literature Review on the state of the art of visual programming applied to the Internet of Things paradigm. Chapter 4 presents the problem this dissertation aims to solve, as well as the approach taken to solve it. Finally, Chapter 5 concludes this dissertation with a reflection on the future contributions of this project.

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 [32], 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 [44].

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 [40], respectively, which is compatible with the characterizing mentioned above. An illustrative representation can be seen in Figure ...

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 [36]. 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

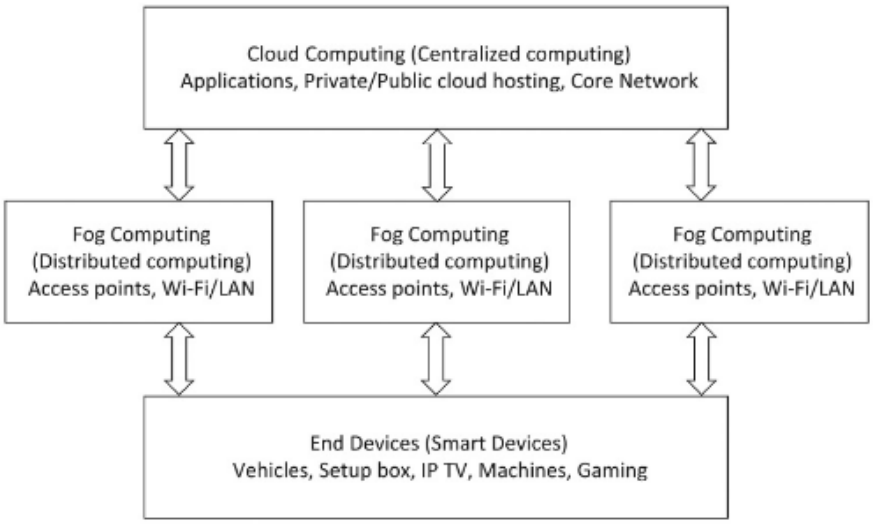
Nowadays, with the improvement of wireless networks and the hardware and software of mobile devices, there is a possibility to take advantage of this variables in the computational execution of IoT systems. This will allow for devices in the network to communicate and share resources between them, reducing 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 [37]. 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 [39, 52].

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 [51].

2.2 Visual Programming Languages

Visual Programming, as defined by Shu, consists of using meaningful graphical representations in the process of programming [53]. 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 [48].

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.

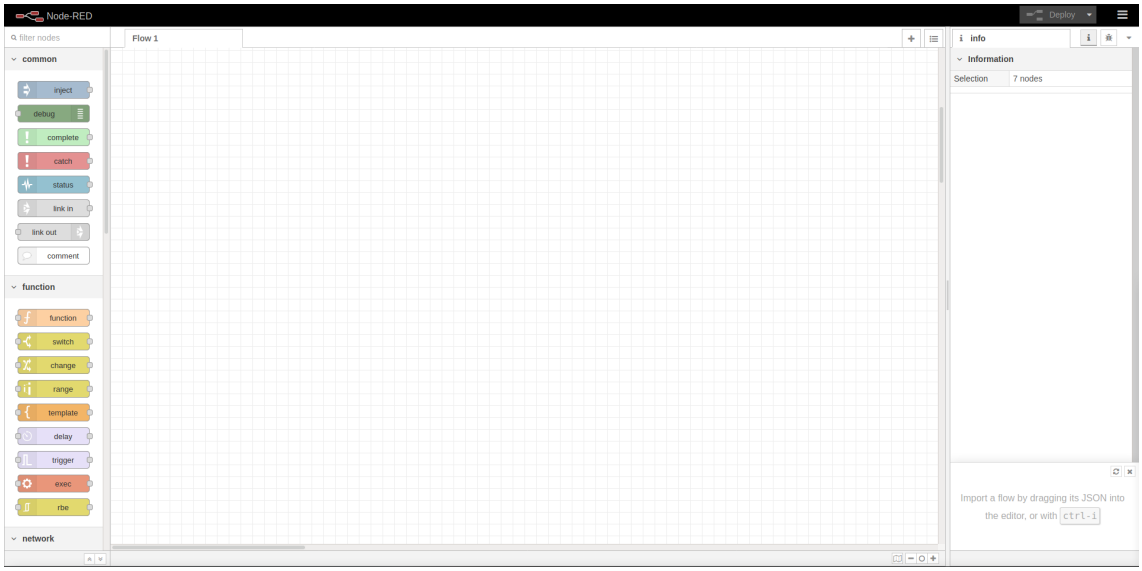
2.3 Node-RED

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 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

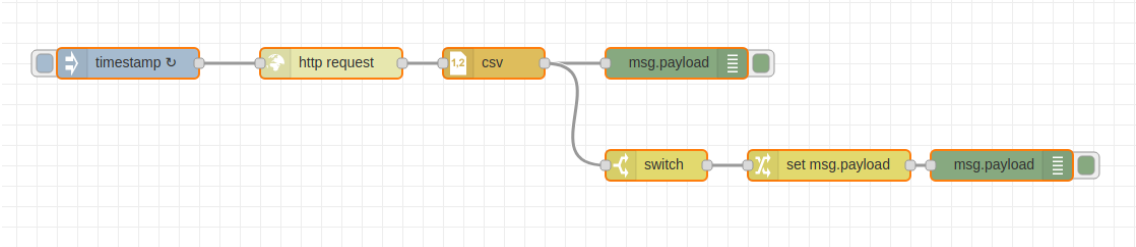


One example of a *flow* can be seen in picture 2.3, where a request is being made in intervals of 5 minutes to a HTTP URL that returns a CSV with the feed of significant earthquakes in the last 7 days. The data from the CSV is then printed to the debug console and, if the magnitude is equal or bigger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

[1] https://nodered.org/

Figure 2.3: Example of a Node-RED flow



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 new nodes and improvements to the tool. It is the most popular open-source visual programming tool for IoT, with more than 9,300 stars on Github.

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

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 contains the results of the Systematic Literature Review, as well as their categorization. Section 3.1.3 contains the additional tools found in a survey and their analysis. The discussion and analysis of the tools found as well as the answering of the research questions made previously is made in Section 3.1.4. The Systematic Literature Review conclusions is presented in Section 3.1.5. Lastly, Section 3.2 contains the state of the art of visual programming tools applied to IoT that implement a decentralized architecture.

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 [46].

3.1.1 Methodology

During this Systematic Literature Review, a specific methodology was followed to reduce bias and produce the best results [46]. 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:

SRQ1: What relevant VPLs applied to IoT orchestration exist? Internet of Things is a paradigm with several years, and its integration with visual programming languages makes their development easier for the end-user. The tools that integrate these two paradigms are useful and reduce the overhead of programming or prototyping IoT systems.

SRQ2: What is the tier and architecture of the tools found in RQ1? IoT systems can belong to one or more of tiers - Cloud, Fog and Edge as well as implement a centralized or decentralized architecture. A visual programming tool applied to IoT orchestration can be used to facilitate the development of systems that operate on these tiers. Each tier and type of architecture offers vantages and disadvantages, which are important in order to understand the usages and characteristics of a system.

SRQ3: What was the evolution of VPLs applied to IoT orchestration along the years? In order to understand the field of visual programming tools applied to IoT, more specifically its orchestration, it is important to perceive its evolution.

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. Presents a tool or framework that doesn't support orchestration of multiple devices;
4. 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 3.3.

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)	22	43

3.1.2 Results

After analyzing the 22 publications, we organized them by categories. [48] is a survey and the remaining 21 were frameworks or tools.

Regarding the survey, publication [48] makes an in-dept 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.

The remaining 21 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. **Belsa et al** [8], a solution for connecting devices from different IoT platforms, using Flow Based Programming with Node-RED. 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** [23] 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 [28], 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** [59], 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** [45] 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** [20], 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** [26] 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** [31] 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** [57], 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. **DDFlow** [43] 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.

13. The tool proposed by **Kefalakis et al.** [34] 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.

14. The approach presented by **Eterovic et al.** [24] 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.

15. **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.

16. **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.

17. An IoT-based GUI, proposed by **Besari et al.** [50] [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.

18. **CharIoT** [56] 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.

19. A new technology, proposed by **Desolda et al.** [21], 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.

20. An End User Development (EUD) tool, proposed by **Eun et al.** [25], 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.

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. In this case we considered scalability in terms of number and different type of devices supported. 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: Small circles (•) mean *yes*, hyphens (-) means *no information available*, empty means *no* and asterisk (*) means more than one. The ⋆ symbol represents certainty in the evaluation made.

Tool Certainty	**Scope** ⋆⋆⋆⋆	**Architecture** ⋆⋆⋆	**License** ⋆⋆⋆⋆⋆	**Tier** ⋆⋆⋆	**Scalability** ⋆⋆⋆	**Programming** ⋆⋆⋆⋆	**Web-based** ⋆⋆⋆⋆⋆
Belsa et al.[8]	*	Centralized	-	Cloud	High	Hybrid text and visual system	•
Ivy [23]	*	Centralized	-	Cloud	Medium	Purely visual language	
Ghiani et al. [28]	Home Automation	Centralized	-	Cloud	-	Form-based programming	•
ViSiT [2]	*	Centralized	-	Cloud	High	Hybrid text and visual systems	•
Valsamakis and Savidis [59]	Ambient Assisted Living	Centralized	-	Cloud	-	Hybrid text and visual system	•
WireMe [45]	Education, Home Automation	Centralized	-	Cloud	-	Hybrid text and visual system	
VIPLE [20]	Education	Centralized	-	Cloud	-	Hybrid text and visual system	
Smart Block [7]	Home Automation	Centralized	-	Cloud	-	Hybrid text and visual system	•
PWCT [26]	*	Centralized	GNU GPL v2.0	-[1]	High	Hybrid text and visual system	
DDF [31]	N/A	Decentralized	Apache 2.0	Fog	High	Hybrid text and visual system	•
GIMLE [57]	Industry	Centralized	-	Cloud	High	Hybrid text and visual system	•
DDFlow [43]	Security	Decentralized	-	Fog and Edge	-	Hybrid text and visual system	•
Kefalakis et al. [34]	-	Centralized	LGPL V3.0[3]	Cloud	-	Hybrid text and visual system	
Eterovic et al. [24]	Home Automation	-[4]	-	-	-	Hybrid text and visual system	-
FRED [11]	*	Centralized	-[5]	Cloud	High	Hybrid text and visual system	•
WoTFlow [10]	-	Decentralized	-	Fog and Edge	-	Hybrid text and visual system	•
Besari et al. [9] [50]	Education	Centralized	-	Cloud	-	Hybrid text and visual system	
CharIoT [56]	Home Automation	Centralized[6]	-	Cloud and Edge[6]	High[6]	Form-based programming	•
Desolda et al. [21]	Smart Museums	-	-	-	-	Hybrid text and visual system	
Eun et al. [25]	Home Automation	Centralized	-	-	-	Form-based programming	•

[1] Used for several purposes, didn't specify the tier it is located in regarding IoT.
[2] Since it uses Node-RED, this information was based on its architecture.
[3] Under the same license of OpenIoT.
[4] No information given regarding the architecture of the environment created, only the VPL.
[5] No information about license is given, but further research discovered that it has paid plans and no source code available.
[6] CharIoT uses the Giotto stack, https://iotexpedition.org/about.html, from where we retrieved this information.

3.1.3 Expanded Search

The results of the Systematic Literature Review are disclosed in Section 3.1.2. However, there are tools that were found by non-systematic surveys [48] that are not present in the results mentioned. Possible reasons for this divergence may consist of:

1. tools not having academic publications associated to them, making it impossible to be returned as results of searches to the publication databases mentioned in Section 3.1.1.2. One example is *Node-RED* [27].

3.1.3.1 Expanded Results

The results from the found survey [48] were analyzed. The retrieved tools were assessed against the evaluation process defined in Section 3.1.1.7 and characterized with the categories mentioned in Section 3.1.2. Using the methodology described, the results are:

1. **Node-RED** [27] is a visual programming environment applied to the IoT paradigm. It makes use of a flow-based development and supports a wide range of devices and APIs. Due to being open-source and extendable, its large community contributes with features that enrich the tool, some of them talked about in Section 3.1.2 (e.g. *FRED* [11] and *DDF* [31]).

2. **NETLab Toolkit** [58] is a visual environment that makes use of drag and drop actions to allows its users to build IoT applications. It provides a web interface to connect sensors, actuators and others for the development of quick prototypes.

3. **NooDL** [42] is a platform that provides a visual programming interface for prototyping applications. It allows for the creation of interfaces, using live data and supporting several types of hardware. Although its not specific to IoT, NooDL covers the programming of IoT systems. It makes use of MQTT broker agents for connecting devices and visual paradigms such as *nodes*, *connections* and *hierarchies* to allow the user to build its system.

4. **DGLux5** [22] for DSA is a *drag-and-drop* visual language and environment that allows its users to build applications tailored for Distributed Services Architecture (DSA) IoT middleware. It provides a dashboard for analyzing and controlling device data in real-time, as well as building the system only using visual elements.

5. **AT&T Flow Designer** [6] is a visual tool incorporated in a cloud development environment, applied to the development of IoT systems. Its visual paradigm is similar to Node-RED, with the notion of *nodes* and *wires*. This tool provides an easy iteration and improvement of a product, as well as an easy deployment.

6. **GraspIO** [38] is a Graphical Smart Program for Inputs and Outputs that contains a block *drag-and-drop* visual paradigm that allows its users to build applications for the *Cloudio* hardware. It offers a Cloud Service that connects and manages all *Cloudio* devices, making them available at the user mobile device.

7. **Wyliodrin** [60] is a browser-based visual programming environment that allows development of IoT systems of several devices, such as Raspberry Pi, Arduino, Intel Galileo, Intel Edison and others. It provides a *drag-and-drop* environment, as well as support for text-based languages. A dashboard for visualizing the data collected is provided.

8. **Zenodys** [15] provides a *drag-and-drop* interface to build application backends as well as a user interfaces. Its computing engine can run in several types of devices, from the cloud to chips, devices and distributed computers. Zenodys contains a visual debugger as well as support for text-based programming and code generation.

Table 3.5: Small circles (•) mean *yes*, hyphens (-) means *no information available*, empty means *no* and asterisk (*) means more than one. The ⋆ symbol represents certainty in the evaluation made.

Tool Certainty	**Scope** ⋆⋆⋆⋆	**Architecture** ⋆⋆⋆	**License** ⋆⋆⋆⋆⋆	**Tier** ⋆⋆⋆	**Scalability** ⋆⋆⋆	**Programming** ⋆⋆⋆⋆	**Web-based** ⋆⋆⋆⋆⋆
Node-Red [27]	*	Centralized	Apache 2.0	Cloud and Edge	High	Hybrid text and visual system	•
NETLab Toolkit [58]	-	-	GNU GPL	-	-	Hybrid text and visual system	•
NooDL [42]	*	-	NooDL End User License[1]	-	-	Hybrid text and visual system	
DGLux5 [42]	*	-	DGLux Engineering License	-	High	Purely visual language	
AT&T Flow Designer [6]	*	-	GNU GPL3	Cloud	High	Hybrid text and visual system	•
GraspIO [38]	Education	-	BSD	Cloud	-	Purely visual language	
Wyliodrin [60]	*	-	GNU GPL3	-	-	Hybrid text and visual system	•
Zenodys [15]	*	-	GNU GPL3	-	High	Hybrid text and visual system	•

[1] Available at https://www.noodl.net/eula

3.1.4 Analysis and Discussion

The tools presented in this Systematic Literature Review passed the evaluation process defined in Section 3.1.1.7. Tools that only supported one device were left out, as well as tools that extended a VPL applied to IoT.

3.1.4.1 Evolution Analysis

Add Graphic with the evolution of nr of articles in the years; try to get a pattern

3.1.4.2 Result Analysis

REVIEW

Scope Most of the tools found have several scopes, such as education, industry or home automation. From the 28 tools, 6 were specific to Home Automation, 4 to Education, 3 to specific domains, 1 for industry and the remain had a wide range of use cases.

Architecture From the 28 tools found, 16 tools have a centralized architecture, 3 are decentralized and the remaining 9 didn't have enough information to reach a conclusion.

License Most of the tools didn't mention a license and the ones who did were in its majority open source (e.g. GNU GPL2, GNU GPL3, Apache 2.0 and LGPL3).

Tier From the 28 tools, 9 didn't specify the tier they are situated in. From the remaining 19, 14 are situated in the Cloud layer, 3 are in the Fog and/or Edge and the remaining 2 are work both in the Cloud and Edge tiers. **CHECK THIS**

Scalability The majority of tools analyzed don't have scalability metrics analyzed, more specifically the number of devices supported by them. The ones that do have high scalability, which concludes that that result is analyzed when the tool has support for it.

Programming From the 28 analyzed tools, 22 employ a hybrid text and visual system visual programming paradigm, while 3 use a purely visual and the other 3 a form-based one.

Web-based The majority of tools analyzed are web-based, being accessible with the use of a browser. Only one tool didn't provide an environment, only a specification of a visual programming language.

3.1.4.3 Research Questions

Responder às research questions com os resultados TODO

SRQ1: What relevant VPLs applied to IoT orchestration exist?

SRQ2: What is the tier and architecture of the tools found in RQ1?

SRQ3: What was the evolution of VPLs applied to IoT orchestration along the years?

3.1.5 Conclusions

TODO

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

Section 3.1 mentions some tools that aim to offer a decentralized solution to visual programming environments applied to Internet of Things systems [31] [43] [10].

The work made by in WoTFlow [10], DDF [31] and subsequent works [29] [30] consists of a system built on Node-RED framework and focused on the use case of Smart Cities. Their goal is to make a tool more suitable for the development of fog-based applications that are dependent on the context of the edge devices they operate on.

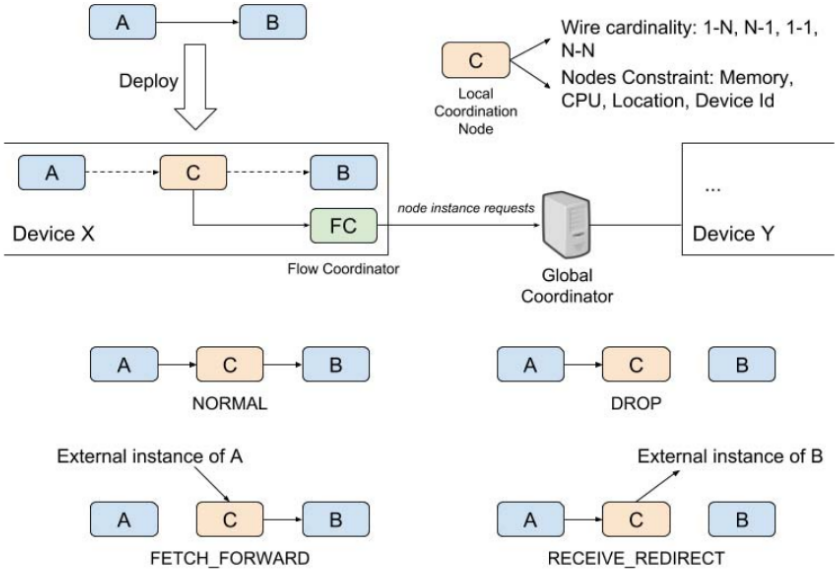
In DDF [31], they started by extending Node-RED and implementing D-NR (Distributed Node-RED), which contains processes that are able to run across devices in local networks and servers in the Cloud. The application, called flow, is built in the visual programming environment, which is running in a development server. All the other devices running D-NR subscribe to a MQTT topic that contains the status of the flow. When a flow is deployed, all devices running D-NR are notified and subsequently analyze the given flow. Based on a set of constrains, they decide which nodes they may need to deploy locally and which sub-flow (parts of a flow) must be shared with other devices. Each device has a set of characteristics, from its computational resources such as bandwidth, available storage to its location. The developer can insert constraints into the flow, by specifying which device a sub-flow must be deployed in or the computational resources needed. However, these constraints and the characteristics of each device must be manually inserted in the system by developers or system operators.

Subsequent work to the previously mentioned tool focused on support for Smart Cities domain. In the publication of 2018 [29], the problems addressed were the deployment of multiple instances of devices running the same sub-flow was a developed feature, as well as the support

for more complex deployment constraints of the application flow. With this, the developer can specify requirements for each node on device identification, computing resources needed (CPU and memory) and physical location. In addition to these improvements, the coordination between nodes in the fog was tackled by introducing a coordinator node. This node is responsible for synchronizing the context of the device with the one given by the centralized coordinator. In Figure 3.1 it is possible to see the four possible states of a coordinator node: (1) NORMAL, where the node passes the data to its output, (2) DROP, in which the node does not pass the data to other node and instead drops it, (3) FETCH_FORWARD, where the node gets the input from an external instance of its supposed input and (4) RECEIVE_REDIRECT in which the node sends the data to an external instance of its output node.

In more recent work [30], support for CPSCN (Cyber Physical Social Computing and Networking) was implemented, making it possible to facilitate the development of large scale CPSCN applications. To make this possible, the contextual data and application data were separated, so that the application data is only used for computation activities and the contextual data is used to coordinate the communication between those activities.

Figure 3.1: Coordination between nodes in D-NR [29]

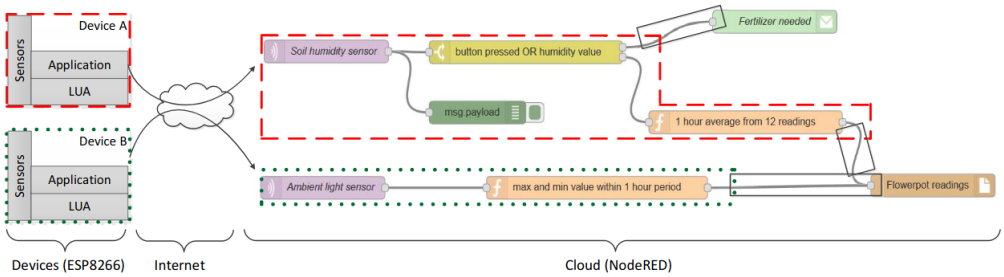


Other approach was made in the publication from Szydlo et al. [55], where they focused on the transformation and decomposition of data flow. Parts of the flow can be translated into the executable parts, such as Lua code. Their contribution includes the concepts of data flow transformation, a new runtime environment called *uFlow* that can be executed on a variety of resource-constrained embedded devices and the integration with the Node-RED platform.

The solution found consisted in the transformation of a given data flow, where the developer

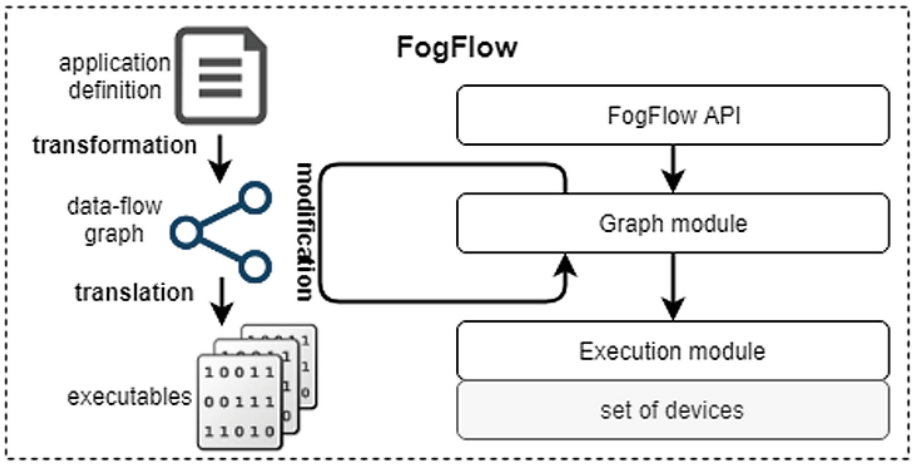
chooses the computing operations that will be run on the devices. The operations that will run directly on the devices are implemented in the form of embedded software, using the developed framework *uFlow*, which allows for parts of the flow to be run on heterogenous devices. All this is integrated with NodeRED. The communication between the devices is made only through the cloud, with no support for peer-to-device communication. The results were promising, with the decrease in the number of measurements made by the sensors. However, there was room for improvements, with the automation of the decomposition and partitioning of the initial flow, the detection bottlenecks which will move computations accordingly from the cloud to the fog. Figure 3.2 represents a situation of partitioning and assignment of tasks. There are two IoT devices and a Node-RED instance running in the Cloud. The system goal is to measure soil humidity and ambient light. If a button is pressed or fertilizer is needed, an e-mail is sent to the gardener. The partition of computation is made in the assumption that the closer a selected process it to the source of data, the higher the amount of data transmitted between computing operations. After parts of the flow are assigned to specific devices, they are altered in order to be executed by *uFlow* and Node-RED. It is possible to observe in Figure 3.2 the results of the transformation process, where the parts of the flow surrounded by a color are executed in the device with the respective color.

Figure 3.2: Partition and assignment of parts of the flow [55]



In 2019, they continued their work with the publication [49], where they built the model and engine *FogFlow*, which enables the design of applications able to be decomposed onto heterogeneous IoT environments according to a chosen decomposition schema. To achieve a level of decentralization and heterogeneity, they abstract out the application definition from its architecture and rely on graph representation to provide an unambiguous, well defined model of computations. The application definition should be infrastructure-independent and contain only data processing logic, and its execution should be possible on different set of devices with different capabilities. Several algorithms for flow decomposition were mentioned [41] [33], but none were specified in terms of results. Figure 3.3 represents the *FogFlow* architecture, which is composed by three modules: (1) the *FogFlow* API, which enables the creation of the application definition, (2) the Graph Module, responsible for processing and transforming the application definition into a data flow graph and finally the (3) Execution Model, which translates the graph and generates executables ready to be run on the assigned devices.

Figure 3.3: *FogFlow* architecture [49]

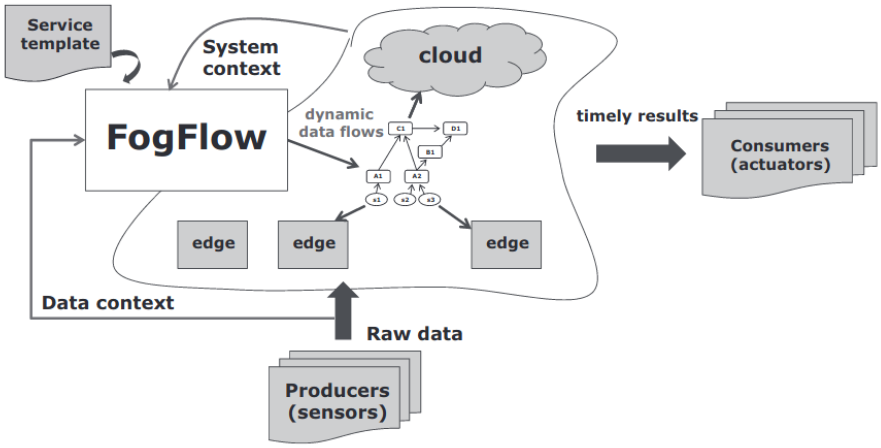


There is another tool with the same name *FogFlow*, but created by Cheng et al. [54]. In the first publication related to this tool [19], the contributions made were the implementation of a standard-base programming model for Fog Computing and a scalable context management. The first contribution consists in extending the dataflow programming model with hints, ir order to facilitate the development of fog applications. The scalable context management introduces a distributed approach, which allows to overcome the limits in a centralized context, achieving much better performance in terms of throughput, response time and scalability. The *FogFlow* framework focuses in a Smart City Platform use case, separated in three areas: (1) Service Management, normally hosted in the cloud, (2) Data Processing, present in cloud and edge devices and (3) Context Management, which is separated in an device discovery unit hosted in the cloud and IoT brokers scattered in edge and cloud.

In more recent work [18], *FogFlow* was improved in order to provide infrastructure providers an environment that allows them to build decentralized IoT systems faster, with increased stability and scalability. The architecture can be seen in Figure 3.4, where the IoT system is represented by dynamic data flows that are orchestrated between sensors (Producers) and actuators (Consumers). The application is first designed using the *FogFlow* Task Designer, a hybrid text and visual programming environment, which results in an abstraction called Service Template. This abstraction contains specifics about the resources needed for each part of the system. Once the Service Template is submitted, the framework will determine how to instantiate it using the context data available. Each task is associated with an operator, which consists of a Docker image, and its assignment is based on how many resources are available on each edge node, the location of data sources and the prediction of workload. Edge nodes are autonomous since they are able to make their own decisions based on their local context, without relying on the central cloud. Review

DDFlow [43], first mentioned in Section 3.1.2, presents another distributed approach by extending Node-RED with a system runtime that supports dynamic scaling and adaption of application deployments. The coordinator of the distributed system maintains the state and assigns tasks

Figure 3.4: *FogFlow* high level model [18]



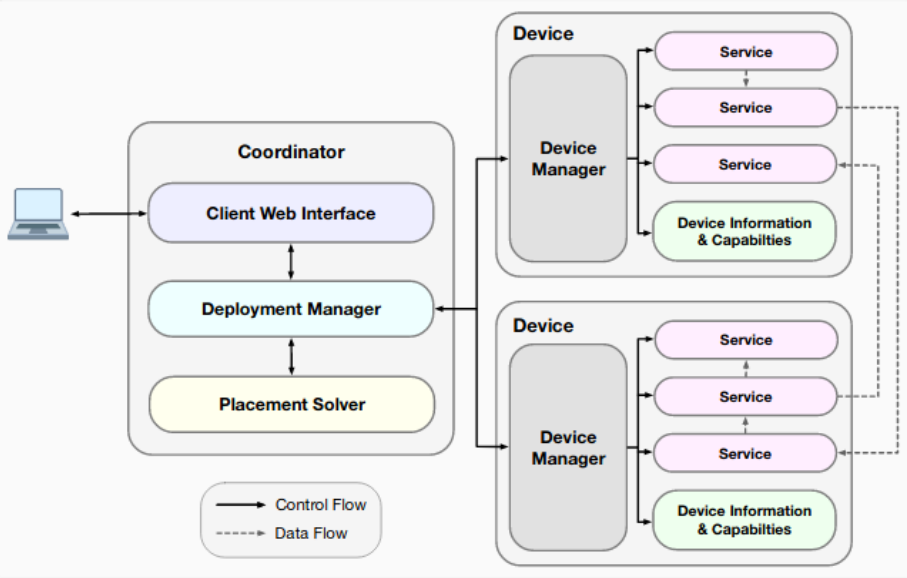
to available devices, minimizing end-to-end latency. Dataflow notions of *node* and *wire* are expanded, with a *node* in DDFlow representing an instantiation of a task that is deployed in a device, receiving inputs and generating outputs. *Nodes* can be constrained in their assignment by optional parameters, *Device* and *Region*, inserted by the developer. A *wire* connects two or more nodes and can have three types: *Stream* (one-to-one), *Broadcast* (one-to-many) and *Unite* (many-to-one).

In a DDFlow system, each device has a set of capabilities and a list of services that correspond to an implementation of a *Node*, as it can be seen in Figure 3.5. Falar mais tarde que isto é bastante limitador The devices communicate this information through their Device Manager or a proxy, if it is a constrained device. The Coordinator is a web server responsible for managing the DDFlow applications and is composed of three parts, which can be seen in Figure 3.5: (1) a visual programming environment were DDFlow application are built, (2) a Deployment Manager that communicates with the Device Managers of the devices and (3) a Placement Solver, responsible for decomposing and assigning tasks to the available devices. When an application is deployed, a network topology graph and a task graph are constructed based on the real-time information retrieved from the devices. The Coordinator proceeds with mapping tasks to devices by minimizing the task graph's end-to-end latency of the longest path. Dynamic adaptation is supported by monitoring the system and adapting to changes. If changes in the network are detected, such as the failure or disconnection of a device, adjustments in the assignment of tasks are made. In addition to this, the Coordinator can be replicated onto many devices in order to improve the reliability and fault-tolerance of the system.

In the evaluation made to DDFlow, the system is able to recover from network degradation or device overload, whereas in a centralized system this would cause its total failure.

FALAR SOBRE mobile_cloud_heterogeneous e mobile_cloud?

Figure 3.5: DDFlow architecture



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. **Communicate 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: Communicate 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 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 Main Hypothesis

Write main hypothesis

This dissertation is built around the following hypothesis:

> *"Given an IoT system with several heterogeneous devices connected, a decentralized architecture is more resilient, efficient and scalable than a centralized one."*

The attributes presented in the hypothesis will be measure against a system using the current development branch of Node-RED. These attributes consist of:

- **Resilience** means the system capability to adapt to failures and changes. It will be measured by injecting failures and measuring the Mean Time T Recover (MTTR).
- **Efficiency** how fast the system can execute the logic of the system, as well as communicate. The efficiency of a system is measured by the sum of total latency of the system.
- **Scalability** specifies how a system can grow in terms of devices supported, while maintaining its features. This attribute will be tested by increasingly adding more devices and assessing the behavior of the system.

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

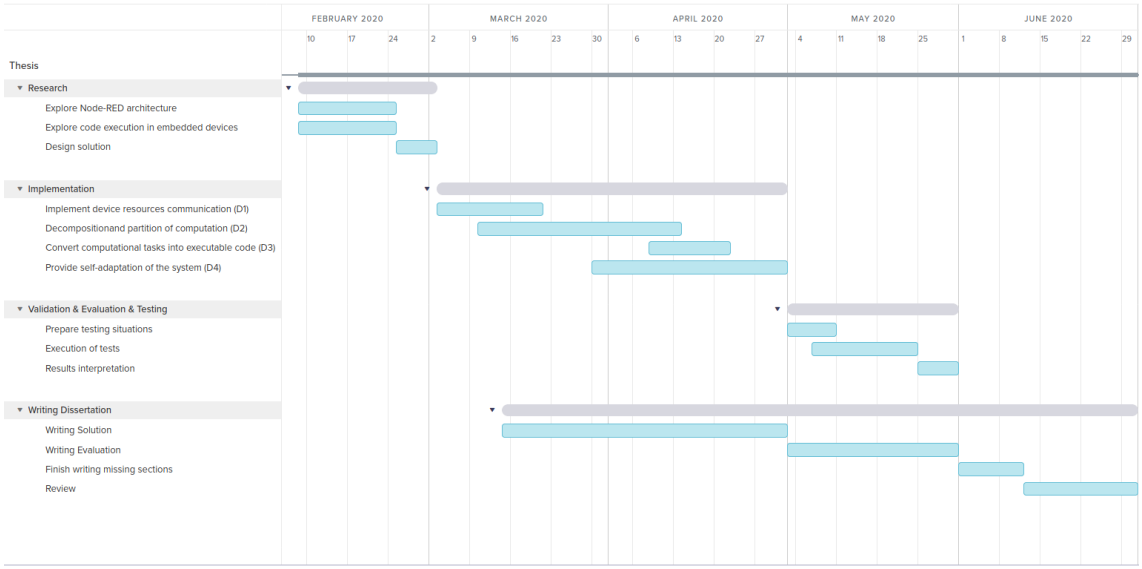
4.6 Planning

In order to help managing the amount of work needed in the next few months, a *Gantt Chart* was built. It is possible to consult it in Figure 4.1.

4.7 Summary

TODO

Figure 4.1: *Gantt Chart* for this dissertation



Chapter 5

Conclusions

TODO pls kill me

5.1 Expected Results

TODO no more thesis, much happiness

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] AT&T. AT&T Flow Designer. Available: https://flow.att.com, 2020. Last access 2020. [Online].

[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] Zenodys B.V. Zenodys. Available: https://www.zenodys.com/, 2020. Last access 2020. [Online].

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

[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] B. Cheng, E. Kovacs, A. Kitazawa, K. Terasawa, T. Hada, and M. Takeuchi. Fogflow: Orchestrating iot services over cloud and edges. *NEC Technical Journal*, 13:48–53, 11 2018.

[19] Bin Cheng, Gurkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. Fogflow: Easy programming of iot services over cloud and edges for smart cities. *IEEE Internet of Things Journal*, PP:1–1, 08 2017.

[20] 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.

[21] 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.

[22] DGLogik. DGLux5. Available: http://dglogik.com/products/dglux-for-dsa, 2020. Last access 2020. [Online].

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] OpenJS Foundation. Node-RED. Available: https://nodered.org/, 2020. Last access 2020. [Online].

[28] 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.

[29] N. K. Giang, R. Lea, M. Blackstock, and V. C. M. Leung. Fog at the edge: Experiences building an edge computing platform. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 9–16, July 2018.

[30] N. K. Giang, R. Lea, and V. C. M. Leung. Exogenous coordination for building fog-based cyber physical social computing and networking systems. *IEEE Access*, 6:31740–31749, 2018.

[31] 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.

[32] 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.

[33] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

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

[35] Martin Kleppmann, Adam Wiggins, Peter Hardenberg, and Mark McGranaghan. Local-first software: you own your data, in spite of the cloud. pages 154–178, 10 2019.

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

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

[38] Grasp IO Innovations Pvt. Ltd. GraspIO. Available: https://www.grasp.io/, 2020. Last access 2020. [Online].

[39] 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.

[40] 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.

[41] Mohammed Islam NAAS, Laurent Lemarchand, Jalil Boukhobza, and Philippe Raipin. A graph partitioning-based heuristic for runtime iot data placement strategies in a fog infrastructure. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 767–774, New York, NY, USA, 2018. Association for Computing Machinery.

[42] NooDL. NooDL. Available: https://classic.getnoodl.com/, 2020. Last access 2020. [Online].

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] Reza Rawassizadeh, Timothy Pierson, Ronald Peterson, and David Kotz. Nocloud: Exploring network disconnection through on-device data analysis. *IEEE Pervasive Computing*, 17, 03 2018.

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

[49] Joanna Sendorek, Tomasz Szydlo, Mateusz Windak, and Robert Brzoza-Woch. *FogFlow - Computation Organization for Heterogeneous Fog Computing Environments*, pages 634–647. 06 2019.

[50] 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.

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

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

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

[54] SmartFog. FogFlow. Available: https://github.com/smartfog/fogflow, 2020. Last access 2020. [Online].

[55] 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.

[56] 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.

[57] 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.

[58] NETLab Toolkit. NETLabTK: Tools for Tangible Design. Available: www.netlabtoolkit.org/, 2020. Last access 2020. [Online].

[59] 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.

[60] Wyliodrin. Wyliodrin. Available: https://wyliodrin.com/, 2020. Last access 2020. [Online].