Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Visual Programming and Orchestration in the Internet-of-Things: A Systematic Literature Review

Margarida Silva[1] | João Pedro Dias*[1,2] | André Restivo[1,3] | Hugo Sereno Ferreira[1,2]

[1]DEI, Faculty of Engineering, University of Porto, Porto, Portugal
[2]INESC TEC, Porto, Portugal
[3]LIACC, Porto, Portugal

Correspondence
*João Pedro Dias, DEI - FEUP, Rua Dr. Roberto Frias, Porto, Portugal Email: jpmdias@fe.up.pt

Funding Information
This research was supported by the Portuguese Foundation for Science and Technology (FCT), Grant Number: SFRH/BD/144612/2019

Summary

Internet-of-Things (IoT) systems are considered one of the most notable examples of complex, large-scale systems. Some authors have proposed visual programming solutions to address part of this inherent complexity, but most of these systems depend on a centralized unit to carry out most – if not all – the orchestration between devices and system components, hindering the degree of scalability and distribution that can be attained. In this work, we carry out a systematic literature review of the current solutions that provide visual and decentralized orchestration to define and operate IoT systems. Our work reflects upon a total of 29 proposals that address these issues up to a certain degree. We provide an in-depth discussion of these works, and find out that only four of these solutions attempt to tackle this issue as a whole, though still leaving a set of open research challenges. We finally argue that this challenges, if addressed, could make IoT systems more fault-tolerant, with impact on their dependability, performance, and scalability.

KEYWORDS:
Internet-of-Things, Orchestration, Visual Programming, Decentralized Computation, Large-Scale Systems

1 | INTRODUCTION

The Internet-of-Things (IoT) is composed of a myriad of devices, having a wide range of capabilities that are connected to the Internet directly or indirectly, allowing to transfer, integrate, analyze and act according to data generated by themselves[1]. IoT systems agglomerate both sensing and actuating devices at an unprecedented scale[†], with applications ranging from mission-critical systems to entertainment and commodity solutions[3].

The widespread usage of IoT across application domains, designed and built by different and, even, competitive, manufacturers, led to a mostly uncontrollable and ever-growing heterogeneity of devices, differing in several aspects such as computational power, protocols, and architectures. Additionally, the large-scale and distributed (geographically and logically) nature of IoT makes them highly-complex systems. These factors lead to several issues in developing these systems, as well as guaranteeing their scalability, maintainability, security, and dependability[4]. These factors prompted the need for tools allowing users that have reduced technical knowledge to configure and adapt their systems to their needs (from manufacturing floor automation to *smart home* system customization), leading to the birth of several different *low-code* solutions, that try to reduce the inherent complexity of programming and configuring these systems[5]. Visual Programming Languages (VPL) and similar visual programming

[†]According to Siemens, 26 billion physical devices are part of the Internet as of 2020, and predictions are pointing at 75 billion in 2025[2].

approaches, either model-based or mashup-based[6], allow the user to configure the system by using and arranging visual elements that and then translate them into code[7]. It provides the user with an intuitive and straightforward interface for coding at the possible cost of losing expressive power. Different languages have different focuses, such as education, video game development, 3D construction, system design, and, of course, Internet-of-Things development and configuration[8]. As an example of one of the latter, we have Node-RED[‡], which is one of the most used open-source visual programming tools [9], providing both a visual editor and a runtime environment for Internet-of-Things systems.

Most mainstream visual programming solutions focused on Internet-of-Things, Node-RED included, have a centralized approach (which can be *on-premises* or cloud-based), where the main component executes most of the computation on data provided by edge (*i.e.*, sensors and actuators) and fog devices, and other third-party services. There are several consequences of this approach: (1) it introduces a single point of failure, (2) local data is being transferred across boundaries (*e.g.*, private, technological, and political) either without a need (privacy) or even in violation of legal constraints (*e.g.*, General Data Protection Regulation)[10,11], and (3) computation capabilities of the lower-tier — edge and fog — might be being wasted. There has been an increasing research effort put in *Fog Computing* and *Edge Computing*[12]. Both concepts focus on using the resources available in lower-tier devices to improve the overall dependability, performance, and scalability. However, these efforts are still in their early stages, and manifestos such as the Local-First[13] (i.e., data and logic should reside locally, independent of third-party services faults and errors) and NoCloud[14] (i.e., on-device and local computation should be given priority over cloud service computation) have not yet received enough attention from researchers.

In this paper, we present a systematic review of the current literature on visual programming solutions for IoT with a particular focus on the ones which take into account orchestration of multiple system parts. Our initial search yielded a total of 2698 results across three different scientific databases. These results were selected, according to our defined *inclusion* and *exclusion* criteria, leaving us with 21 papers. We proceeded to enhance our selection by a mixture of snowballing, taking into account previous (mostly non-systematic) surveys and adding eight solutions not found during the search process. Our result provided a total of 28 solutions, presented across 22 papers. We proceeded to compare them regarding a selection of characteristics, including scope, architecture, scalability, and visual programming paradigms. We then carried out an in-depth analysis on the subset of the solutions that provided mechanisms for decentralized orchestration.

This remaining of this paper is structured as follows: Section 2 gives an overview of some key background concepts, Section 3 presents the methodology used in this research, Section 4 presents the results of the literature review, followed by an in-depth analysis of the current alternative for visual IoT decentralized orchestration in Section 5. An overview of the current issues and research challenges, along with some final remarks, is given in Section 6.

2 | BACKGROUND

2.1 | Internet-of-Things

Internet-of-Things paradigm is defined by the committee of the International Organization for Standardization and the International Electrotechnical Commission[15] 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."*

IoT systems are, mostly, networks of heterogeneous devices attempting to bridge the gap between people and their surroundings. According to Buuya[16], the applications of IoT systems can be divided into four categories: (i) *Home* at the scale of a few individuals or domestic scenarios, (ii) *Enterprise* at the scale of a community or larger environments, (iii) *Utilities* at a national or regional scale and (iv) *Mobile*, which is spread across domains due to its large scale in connectivity and scale.

One might think that IoT only relates to machines and interactions between them. Most of the devices we use in our day-today — *e.g.*, mobile phones, security cameras, watches, coffee machines — are now computation capable of making moderately complex tasks and are continually generating and sending information. This relates to the *human-in-the-loop* concept, where humans and machines have a symbiotic relationship[17].

[‡]Node-RED, https://nodered.org/

2.2 | IoT architectures

Internet-of-Things systems deal with big amounts of data from different sources and have to process it in an efficient and fast fashion. Typical IoT systems are composed of three tiers, which are:

Cloud Tier mostly composed of data centers and servers, normally running remotely. It is characterized by having high computation power and latency.

Fog Tier composed of gateways and devices that are normally between the cloud servers and the edge devices. This tier has less latency than the cloud, more heterogeneity, and, typically, is more geographically distributed.

Edge Tier composed of all the peripheral devices (*e.g.*, sensors, embedded systems, light sources and air conditioners). These devices have several limitations in computational capabilities, but have less latency.

Complementary to these tiers, we can also partition IoT systems into an Application Layer, a Network Layer, and a Perception Layer[18]. At first sight, these might seem compatible with the tiers mentioned above (in the same order); however, not all devices in each tier map to their respective layer. One example is a third-party service that gives readings. It can be contained in the Perceptive Layer, but it is not included in the Edge Tier.

New paradigms of computing appeared related to each of these tiers. The majority of IoT systems use a Cloud Computing architecture, taking 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 (1) high latency, and (2) high use of bandwidth, due to the need to send the data generated from the sensors to the centralized unit[19]. Systems that only use cloud computing face several challenges[20], especially real-time applications, which are sensitive to increased latency. With the increasing computation capabilities of edge devices and the requirement of reduced latency, two new paradigms appeared: Fog Computing and Edge Computing.

2.2.1 | Fog Computing

With the improvement of wireless technologies and the increasing computational power (and reducing costs) of lower-tier devices (*i.e.*, fog and edge), it became possible to improve the computational execution of IoT systems. By not depending so much on the cloud tier, communication and resource sharing between devices can occur with lower latency. The central coordinator (on-premises or cloud-based), which in Cloud Computing was responsible for all the computation, now serves as a scheduler and state manager of the communication between devices, occasionally providing necessary resources. This new paradigm, where fog and edge devices are leveraged as computational entities (and not only merely sensing, actuating and gateway devices in the network) is called Fog Computing, which aims to bring computing closer to the perception tier, bringing the computation nearer to the edge of the network[21]. It focuses on distributing data throughout the IoT system, from the cloud to the edge devices, making the system distributed.

According to Buuya[1], Fog Computing has several advantages: (1) reduction of network traffic by having edge devices filtering and analyzing the data generated and sending data to the cloud only if necessary, (2) reduced communication distance by having the devices communicate between them without using the cloud as a middleman, (3) low-latency by moving the processing closer to the data source rather than communicating all the data to the cloud for it to be processed, and (4) scalability by reducing the burden on the cloud, which could be a bottleneck for the system.

Despite all the advantages, Fog Computing has several requirements and difficulties. 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 more significant than Cloud Computing since it needs to work with heterogeneous devices with different capacities.

2.2.2 | Edge Computing

Edge Computing, also known as Mist 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 tier, which contains the devices closer to the end-user, such as smartphones, TVs and sensors. The goal of this paradigm 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[22,23].

In this paradigm, each device serves both as a data producer and a data consumer. Since each device is constrained in 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 of the lack of easy-to-use tools and frameworks to build cloud-edge systems, non-existent standards regarding the naming of edge devices and the lack of security edge devices have against outside threats such as hackers[24].

There is some confusion in the research community regarding the concepts of Fog and Edge computing. The publication from Iorga et al.[25] was used to inspire the definitions of these terms. Edge Computing focuses on executing applications in constrained devices, without worrying about storage or state preservation. On the other hand, Fog Computing is hierarchical and includes devices with more capabilities, capable of control activities, storage, and orchestration.

2.3 | Visual Programming Languages

Visual Programming, as defined by Shu[26], consists of using meaningful graphical representations in the process of programming. 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[27], visual programming languages are constructed 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[8].

Visual programming languages have several characteristics, such as a concrete process and depiction of the program, immediate visual feedback and require the knowledge of fewer programming concepts[27].

VPLs can be categorized by their visual paradigms and architecture[28]:

Purely Visual Languages where the system is developed 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 a 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 behaviour in reality.

Form-based systems which are based on the architecture and behaviour of spreadsheets.

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

3 | SYSTEMATIC LITERATURE REVIEW METHODOLOGY

This work follows a Systematic Literature Review (SLR) methodology to gather information on the state of the art of visual programming applied to the Internet-of-Things paradigm, with a special emphasis on orchestration concerns. The goal of a systematic literature review is to synthesize evidence with emphasis on the quality of it[29].

During this SLR, a specific methodology was followed to reduce bias and produce the best results[29]. We started by defining the research questions to be answered as well as choosing data sources to search for publications.

3.1 | Research Questions

To reveal the current practice, research and studies related to orchestration in the Internet-of-Things that leverage visual approaches, which enable us to find the current, and pending research challenges, we outline the following research questions (RQ):

RQ1 *What relevant visual programming solutions 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.

RQ2 *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 essential to understand the usages and characteristics of a system.

RQ3 *What was the evolution of visual programming solutions applied to IoT orchestration over the years?* To understand the field of visual programming applied to IoT, more specifically, its orchestration, it is essential to perceive its evolution.

Answering these questions will provide insights that can be valuable for both practitioners — in terms of summarizing what the current practices on the usage of visual programming methodologies for IoT orchestration are — and researchers — showing current challenges and issues that can be further researched.

3.2 | Candidate Searching and Filtering

Our systematic literature review protocol followed the inclusion and exclusion criteria detailed in Table 1 and is outlined in Figure 1. To find the most relevant works for this study, three scientific databases were used, namely: *IEEE Xplore*, *ACM Digital Library* and *Scopus*. These electronic databases contain some of the most relevant digital literature for studies in the area of Computer Science, thus being considered reliable sources of information.

TABLE 1 Inclusion and exclusion criteria.

I/E	ID	Criterion
Exclusion	EC1	Not written in English.
	EC2	Presents just ideas, tutorials, integration experimentation, magazine publications, interviews or discussion papers.
	EC3	Presents a tool, framework or approach that does not support the orchestration of multiple devices.
	EC4	Has less than two (non-self) citations when more than five years old.
	EC5	Duplicated articles.
	EC6	Articles in a format other than camera-ready (PDF).
Inclusion	IC1	Must be on the topic of visual programming in Internet-of-Things.
	IC2	Contributions, challenges and limitations are presented and discussed in detail.
	IC3	Research findings include sufficient explanation on how the approach works.
	IC4	Publication year in the range between 2008 and 2019.
	IC5	Is a survey that focus visual programming in IoT or

We begun our search in these data sources using a query that captured the most probable keywords to appear in our target candidates, namely *visual programming*, *node-red*, *dataflow*, and *Internet-of-Things*. This led us to specify variants of the following query that are understood by the mentioned databases: `((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)`. This search was performed in October of 2019 and the number of results produced can be seen in the first step of Figure 1.

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

1. **Automatic Search**: Run the query string in the different scientific databases and gather results;

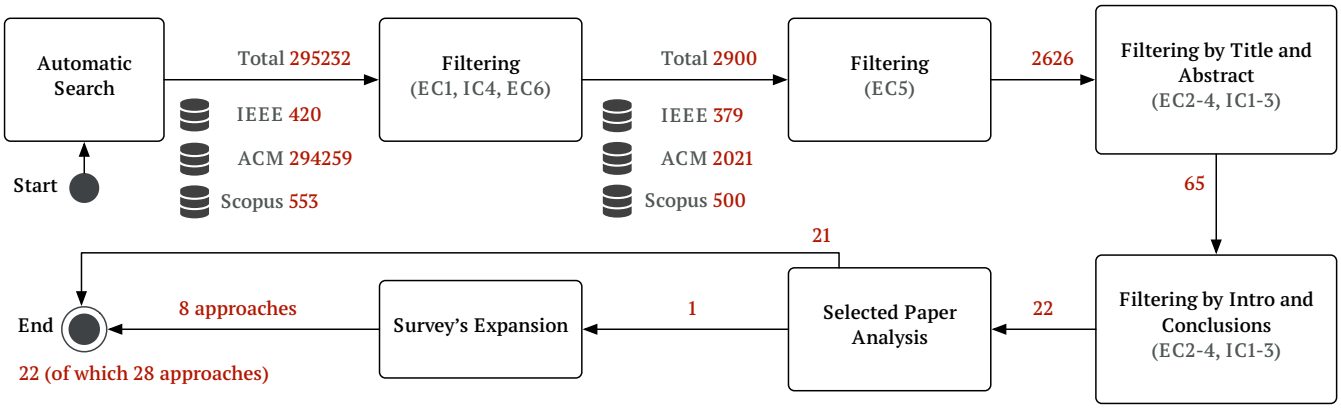

FIGURE 1 Pipeline overview of the SLR Protocol.

2. **Filtering** (*EC1*, *IC4*, and *EC6*): Publications are selected regarding its (1) language, being limited to the ones written in English language, (2) publication date, being limited to the ones published between 2008 and 2019, and (3) publication status, being selected only the ones that are published in their final versions (camera-ready PDF format);

3. **Filtering to remove duplicates** (*EC5*): The selected papers are filtered to remove duplicated entries;

4. **Filtering by *Title* and *Abstract*** (*EC2–EC4*, and *IC1–IC3*): Selected papers are revised by taking into account their *Title* and *Abstract*, by observing the (1) stage of the research, only selecting papers that present approaches with sufficient explanation, some experimental results and discussion on the paper contributions, challenges and limitations, (2) contextualization with recent literature, filtering papers that have less that two (non-self) citations when more than five years old, and (3) leverages the use of visual notations for orchestrating and operating multi-device systems.

5. **Filtering by *Introduction* and *Conclusions*** (*EC2–EC4*, and *IC1–IC3*): The same procedure of the previous point is followed but taking into consideration the *Introduction* and *Conclusion* sections of the papers;

6. **Selected Papers Analysis**: Selected papers are grouped, and surveys are separated; their content is analyzed in detail.

7. **Surveys Expansion**: For the survey papers found, the enumerated solutions are analyzed and filtered taking into account their scope and checking if they are not duplicates of the current selected papers.

8. **Wrapping**: Approaches and solutions gathered from the *Selected Papers Analysis* (individual papers) and from the *Survey Expansion* are presented and discussed.

The total number of publications was 2698, and, after the evaluation process, 22 publications were selected as can be seen in Figure 1. From those, one was a survey and the others presented approaches relevant to our research questions.

4 | LITERATURE REVIEW RESULTS

After analyzing the 22 publications, we organized them by categories; of these, one was a survey[8], and the remaining 21 were papers describing papers that address our research questions. In that survey, the authors make an in-depth review of 13 visual programming languages in the field of IoT, comparing them using four attributes: (1) programming environment, (2) license, (3) project repository and (4) platform support We used this survey to complement our research in Section 4.2.

4.1 | SLR Results

The selected 21 articles described approaches that use visual programming in the IoT context having orchestration considerations. One of the tools is described in two papers, which showcases its evolution. The 20 unique solutions are:

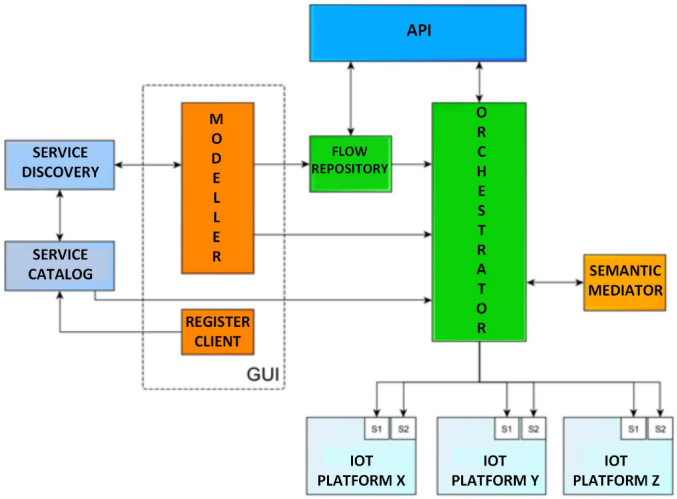

FIGURE 2 Belsa et al.[30] solution architecture. The `Modeller` is a Node-RED flow editor canvas were new flows can be created by connecting nodes that correspond to the available services (`Service Catalog and Discovery`) which are then stored in the `Flow Repository`. The `Orchestrator` is responsible managing and running the specified flows as required (by running several instances of the Node-RED runtime) and converting Node-RED calls to the different IoT Platforms *native calls* (aided by the `Semantic Mediator`).

- **Belsa et al.**[30] present a solution for connecting devices from different IoT platforms, using Flow-Based Programming with Node-RED, depicted in Fig. 2. Its motivation is based on the limitation imposed by the IoT platform on communication between components and extensibility, which limits the possibility to interact with other platforms' services. To validate their solution, they implemented a use case in the domain of transportation and logistics, with a service that uses 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.

- **Ivy**[31] proposes the next step toward visualization applied to IoT with a visual programming tool that allows its users to link devices, inject logic, and visualize real-time data flows using immersive virtual reality. 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 colour, which facilitates the understanding of the system being built or debugger for the user. The experiences made to validate the prototype were positive, with the participants being receptive to Ivy and proposing new use cases.

- **Ghiani et al.**[32] proposition is to build a collection of tools that allow non-developer users to customize their Web IoT applications using trigger-actions rules. The proposed solution provides a web-based tool that allows users to specify their trigger-action rules using *IFTTT*, as well as a context manager middleware that can adapt to the context and events of the devices and apply rules to the system. To validate the developed tool, an example home automation application that displays sensor values and directly controls appliances were built. The results were, for the most part, positive, and the issues found are related to usability and visual clues.

- **ViSiT**[33] uses the jigsaw puzzle metaphor[34] to allow its end-users to implement a system of connected IoT devices. It provides a web-based visual tool connected with a web-service that, given a jigsaw representation, generates an executable implementation. Their goal is achievable by adapting model transformations used by software developers into intuitive metaphors for non-developers to use. They validated the developed tool with a usability evaluation, which was overall positive, with a significant percentage considering the tool useful and providing real-life scenarios where they could implement it.

- **Valsamakis and Savidis**[35] propose a framework for Ambient Assisted Living (AAL) using IoT technologies, which allows for customized automation. It uses visual programming languages to facilitate their end-users - carers, family, friends, elderly - to build and modify automation. 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 smartphones and tablets and is built in Javascript, allowing it to run in browsers. Their future work focuses on integrating different visual programming paradigms to accomplish the requirements of the end-user fully.

- **WireMe**[36] is a solution for building, deploying, and monitoring IoT systems, built with non-developer end-users in mind but also extensible for advanced users to build 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 device's status to the dashboard via MQTT, which is programmable using their visual interface and Lua programming language. Their tool was validated in an empirical study with students around 16 years old and engineering students without programming experience. The results were not positive, with some students not being able to create the required simple logic. Future work consists of improving programming blocks to become more intuitive.

- **VIPLE**[37], Visual IoT/Robotics Programming Language Environment, is a new visual programming language and environment. It provides an introduction to topics such as computing and engineering and tools for more technical domains like software integration and service-oriented computing. 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 has been tested and used in several universities since 2015 due to its large set of features and use cases.

- **Smart Block**[38] is a block-based visual programming language and visual programming environment applied to IoT systems, that allows non-developer users to build their systems quickly. 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 a client-side Javascript library called Blockly, which allows for the creation of visual block languages. Future work for this project consists of supporting device grouping and security by expanding custom blocks, as well as extending the tool for other domains besides home automation.

- **PWCT**[39] is a visual programming language applied to build IoT, Data Computing, and Cloud Computing systems. Its goal consists of reducing the cost of development of these types of systems by providing a comfortable and more productive development tool. The language was meant to compete with text-based languages such as Java and C/C++. It makes use of graphical elements to replace code. It has three main layers: (1) the VPL layer, composed of graphical elements, (2) the middleware layers, responsible for connecting the VPL layer to 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.

- **DDF**[40] 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, by design, 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 significant 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 alternative for designing and developing distributed IoT systems, despite having some open issues such as not having a distributed discovery of devices and networks.

- **GIMLE**[41], Graphical Installation Modelling Language for IoT Ecosystems, is a visual language that uses visual elements to model domain knowledge using significant ontological requirements. The goal of this language is to fill the gap of modeling requirements on the physical properties of IoT installations by proposing a new process for configuring industrial installations. It makes use of flow-based and domain-based visual programming to isolate the requirements' logical flow from their details. The developed tool supports reuse within the models, which is valuable due to the repetitive nature of industrial installations. However, it still needs to clarify its scope within the current practice and its use in production settings.

- **DDFlow**[42] is a macro-programming abstraction that aims to provide efficient means to program high quality distributed apps for IoT. The authors point to a lack of solutions for complex IoT systems programming, causing developers to build their 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 different programmers created them. Developers use Node-Red to specify the application functionalities, and DDFlow handles scalability and deployment. The authors describe DDFlow's goal 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 has custom logic if the available nodes are not enough for some tasks.

- **Kefalakis et al.**[43] proposition consists of a visual environment that operates over the OpenIoT architecture and allows for the development of IoT applications with reduced 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 have its code generated and performed 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 open the possibilities of applications from developers and (3) it is open source.

- **Eterovic et al.**[44] propose 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 non-technical person, with no programming knowledge. Attaining the issues that this can create in the future, it is crucial to create 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.

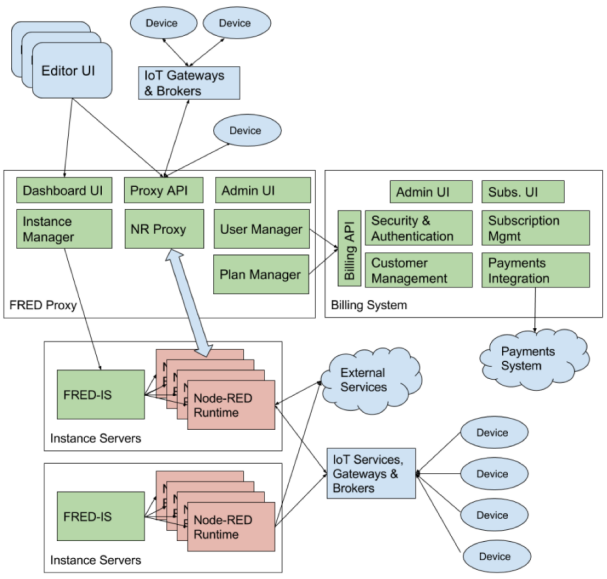

FIGURE 3 FRED[45] high-level architecture. FRED is based on Node-RED and addresses the limitation of running several flows in parallel (multiple runtimes) by orchestrating several instances of Node-RED (`FRED-IS`) using their `FRED Proxy`.

- **FRED**[45] is a frontend for Node-RED, a development tool that makes it possible to host multiple Node-RED runtimes (Fig. **??**). It can be used to connect devices to services in the cloud, manage communication between devices, create new web app applications, APIs and event-integrated services. To provide all these features, FRED allows the running of flows for multiple users, in which all flows get fair access to resources such as CPU, memory, storage, as well as secure access to flow editors and the flow runtime. The authors concluded that FRED is useful for users learning about Node-RED and allows users to prototype cloud-hosted applications rapidly.

- **WoTFlow**[46] 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.

- **Besari et al.**[47,48] proposes an IoT-based GUI that aims to control sensors and actuators in an IoT system using an android application, in which the users use a visual programming language to configure and interact with the IoT system. The system was tested with a Pybot, a robot that is programmable like 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.

- **CharIoT**[49] is a programming environment that promises its end-users a solution that unifies and supports 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. Two types of virtual sensors were developed to capture higher-level events. The programmed virtual sensor provides more accessible and understandable abstractions (defining that a room is "cold" if the temperature is below 20°C). The demonstrated virtual sensors are more complex, requiring the user to provide a demonstration of the occurrence and lack of 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.

- **Desolda et al.**[50] proposition uses a tangible programming language that allows non-programmers to configure smart objects' behaviour to create and customize smart environments. The main goal was to create, with the developed technology, a scenario of a smart museum. The authors defend that the synchronization of smart devices cannot limit the personalization of a smart environment, and it may require experts to build their narrative, much like a museum said. With this in mind, they introduced custom attributes to assign semantics to connected objects to empower and simplify the creation of event-condition-action rules. This is ongoing research focused on developing new technology with an interaction paradigm that supports the input of domain experts in the creation of smart environments. The fact that this technology uses expensive material (tabletop surface as a digital workspace) does not allow a regular user to use it, as stated in the introduction.

- **Eun et al.**[51] proposes an End-User Development (EUD) tool that allows users to develop their applications. It uses the dataflow approach, which allows for a more generalized programming experience as well as 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 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 similar, focusing on consumer development, IoT, and home environments, with Zapier focusing on business environments. Both Zapier and *IFTTT* use the Trigger-Action paradigm (TAP), which differs from the dataflow paradigm used in this paper's tool.

4.2 | Complementary Results

The results of the Systematic Literature Review were disclosed in the previous section. However, some tools were found in a non-systematic survey[8] that are not present in the selected papers. We consider that this divergence may result from tools that have no academic publications associated with them, thus not being present in the publication databases mentioned in Section 3.2. One famous example is *Node-RED*[52]. The results from the survey[8] were analyzed, the described tools were assessed against the

evaluation process defined in Section 3 and characterized by the categories mentioned in Subsection 4.1. Using the methodology described, the results are:

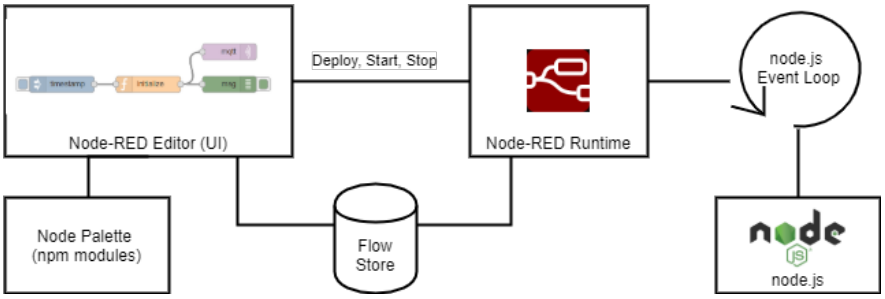

FIGURE 4 Node-RED[52] high-level architecture, identifying its development interface, runtime and `node.js` dependency. The *flows* can be versioned and organized in projects and new modules (*i.e.*, *nodes*) can be added using the `node.js` dependency manager tool (*i.e.*, `npm`).

- **Node-RED**[52] is a visual programming environment applied to the IoT paradigm. It makes use of flow-based development (connecting communication and computation *nodes* in *flows*), supporting a wide range of devices and APIs. It has two main modules: (1) a development interface which consists of a flow drawing canvas and a *node* palette, and (2) a runtime module that leverages the Node.JS event-loop to pass messages between the different *nodes*? . Due to being open-source and extendable, its large community contributes with features that enrich the tool, some of them talked about in Subsection 4.1 (*e.g.*, FRED[45] and DDF[40]).
- **NETLab Toolkit**[53] is a visual environment that makes use of *drag-and-drop* actions to allow users to build IoT applications. It provides a web interface to connect sensors, actuators, and others for quick prototypes.
- **NooDL**[54] 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 it is 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.
- **DGLux5**[55] 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 and builds the system only using visual elements.
- **AT&T Flow Designer**[56] 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.
- **GraspIO**[57] 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's mobile device.
- **Wyliodrin**[58] is a browser-based visual programming environment that allows the 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.
- **Zenodys**[59] provides a *drag-and-drop* interface to build application backends as well as 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.

4.3 | Results Categorization

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

1. **Scope.** Some tools have specific use cases in mind. 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 *Several* if there is more than one.

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

3. **License.** The license of 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.

4. **Tier.** IoT systems, as explained in Section 2.2 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.

5. **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 types of devices supported. Possible values are *low*, *medium*, *high* or N/A, in case there is no sufficient information.

6. **Programming.** According to Downes and Boshernitsan[28] and also mentioned in Section 2.3, 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 are not 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.

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

The resulting categorization of the SLR results is depicted in Table 2. Some key take-ways are easily observable, namely: (1) most tools use a centralized architecture, (2) the hybrid visual-textual programming paradigm is predominant, and (3) most of the tools are web-based. The extended search findings and their categorization is presented in Table 3, following the same previously defined categories.

4.4 | Analysis and Discussion

The tools presented in this Systematic Literature Review passed the evaluation process defined in Section 3. Tools that only supported one device were left out, as well as tools that extended a VPL by applying it to IoT.

4.4.1 | Evolution Analysis

To understand the evolution of visual programming languages applied to IoT, the publication years of the tools found in Section 4, as well as the launch years of the survey tools of Section 4.2, were analyzed. Figure 5 contains the their evolution, where we can observe that there was a more substantial amount of work related to this topic in the years 2017 and 2018. The year 2019 still does not have conclusive data.

4.4.2 | Result Analysis

Scope Most of the tools found have several scopes, such as education, industry or home automation. From the 28 tools, six were specific to home automation, 4 to education, 3 to specific domains, and 1 for the industry; the remainder 14 had a wide range of use cases.

TABLE 2 Visual programming solutions 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.[30]	Several	Centralized	-	Cloud	High	Hybrid	●
Ivy[31]	Several	Centralized	-	Cloud	Medium[7]	Purely visual	
Ghiani et al.[32]	Home Automation	Centralized	-	Cloud	-	Form-based	●
ViSiT[33]	Several	Centralized	-	Cloud	High	Hybrids	●
Valsamakis and Savidis[35]	Ambient Assisted Living	Centralized	-	Cloud	-	Hybrid	●
WireMe[36]	Education, Home Automation	Centralized	-	Cloud	-	Hybrid	
VIPLE[37]	Education	Centralized	-	Cloud	-	Hybrid	
Smart Block[38]	Home Automation	Centralized	-	Cloud	-	Hybrid	●
PWCT[39]	Several	Centralized	GNU GPL v2.0	-[1]	High	Hybrid	
DDF[40]	-	Decentralized	Apache 2.0	Fog	High	Hybrid	●
GIMLE[41]	Industry	Centralized	-	Cloud	High	Hybrid	●
DDFlow[42]	Security	Decentralized	-	Fog and Edge	-	Hybrid	●
Kefalakis et al.[43]	-	Centralized	LGPL V3.0[3]	Cloud	-	Hybrid	
Eterovic et al.[44]	Home Automation	-[4]	-	-	-	Hybrid	-
FRED[45]	Several	Centralized	-[5]	Cloud	High	Hybrid	●
WoTFlow[46]	-	Decentralized	-	Fog and Edge	-	Hybrid	●
Besari et al.[48 47]	Education	Centralized	-	Cloud	-	Hybrid	
CharIoT[49]	Home Automation	Centralized[6]	-	Cloud and Edge[6]	High[6]	Form-based	●
Desolda et al.[50]	Smart Museums	-	-	-	-	Hybrid	
Eun et al.[51]	Home Automation	Centralized	-	-	-	Form-based	●

[1] Used for several purposes, did not 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 is given regarding the architecture of the environment created, only the VPL.
[5] No information about the license is given, but further research discovered that it had paid plans and no source code available.
[6] CharIoT uses the Giotto stack[60] from where we retrieved this information.
[7] Certainty regarding this information is low.

TABLE 3 Characterization of the visual programming solutions applied to IoT from survey[8]. 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
Node-Red[52]	Several	Centralized	Apache 2.0	Cloud and Edge	High	Hybrid	●
NETLab Toolkit[53]	-	-	GNU GPL	Edge[2]	-	Hybrid	●
NooDL[54]	Several	-	NooDL End User License[1]	Cloud[2]	-	Hybrid	
DGLux5[54]	Several	-	DGLux Engineering License	Cloud and Fog[2]	High[2]	Purely visual	
AT&T Flow Designer[56]	Several	-	GNU GPL3	Cloud[2]	High[2]	Hybrid	●
GraspIO[57]	Education	-	BSD	Cloud[2]	-	Purely visual	
Wyliodrin[58]	Several	-	GNU GPL3	All[2]	-	Hybrid	●
Zenodys[59]	Several	-	GNU GPL3	Cloud[2]	High[2]	Hybrid	●

[1] Available at https://www.noodl.net/eula
[2] Certainty regarding this information is low.

Architecture From the 28 tools found, 16 tools have a centralized architecture, three are decentralized, and the remaining nine do not present enough information to conclude on this topic.

License Most of the tools did not 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).

Scalability The majority of tools analyzed do not have scalability metrics analyzed, more specifically, the number of devices supported by them. The ones that do have high scalability, which seems to indicate that scalability is only 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 three a form-based one.

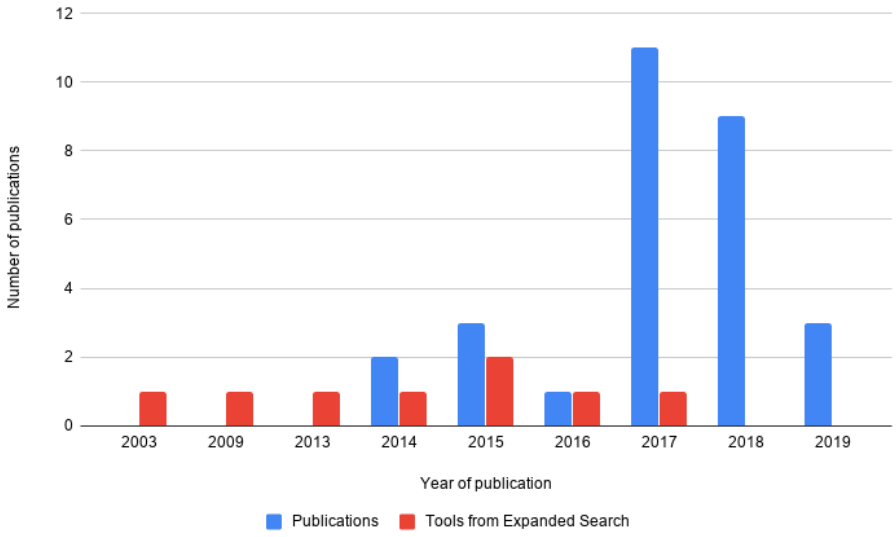

FIGURE 5 Publications and tools of VPL tools applied to IoT per year.

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

4.4.3 | Research Questions

The research questions presented in Section 3.1 served as a way of directing the research of this Systematic Literature Review and obtain answers to relevant questions regarding the available tools that apply visual programming languages to the IoT domain. These answers are:

RQ1 *What relevant visual programming solutions applied to IoT orchestration exist?* From the analyzed tools in Section 4 and 4.2, we found 28 visual programming tools applied to IoT orchestration.

RQ2 *What is the tier and architecture of the tools found in **RQ1**?* Tables 2 and 3 give an overview of the characteristics of all the tools found. In these tables and subsequent analysis in Section 4.4.2 it is concluded that the majority of the tools have a centralized architecture and work in the Cloud tier.

RQ3 *What was the evolution of visual programming solutions applied to IoT orchestration over the years?* As it can be observed in Section 4.4.1 and more specifically in Figure 5, there are visual programming tools applied to the orchestration of IoT since 2003, and in 2017 and 2018 there was a bigger number of publications with a focus on building these type of tools.

4.5 | Summary

In this Systematic Literature Review, 2698 publications were analyzed from IEEE, ACM and Scopus databases, resulting in 20 visual programming tools applied to the Internet-of-Things. A survey made on the visual programming solutions applied to IoT found during the research process resulted in 8 more tools, making a total of 28.

The results show that there is a significant number of tools that allow end-users to build IoT systems using visual programming in several different scopes. The majority of these tools have a centralized architecture and operate in the Cloud tier. Despite the considerable amount of tools, most of them do not have their source code accessible nor have a license. The results from the expanded search are more positive in this aspect, with the majority of them being open-source, such as Node-RED[52], NETLab Toolkit[53] and others. However, this poses a problem since there is an evident lack of open source tools.

In summary, the majority of tools found do not possess a license, employ a centralized architecture, operate in the Cloud tier and use a hybrid text and visual programming system. Thus, it propels the possibility of future research on designing and building

a visual programming tool applied to IoT that is (1) open-source, (2) has a decentralized architecture and (3) also operates in the Fog and/or Edge tiers.

5 | VISUAL ORCHESTRATION IN IOT

Although the substantial amount of solutions found during our systematic literature (Section 4), only a small fraction of those aim to offer a truly decentralised solution to visual orchestration for Internet-of-Things systems. These solutions are now analysed in detail, followed by a comparison and discussion.

5.1 | DDF

The work made by in WoTFlow[46], DDF[40] and subsequent works[61,62] consists of a system built on the 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 where they operate.

In DDF[40], the authors started by extending Node-RED and implementing D-NR (Distributed Node-RED), which contains processes that can 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 an MQTT topic that contains the status of the flow. When a flow is deployed, all devices running D-NR are notified and subsequently analyse the given flow. Based on a set of constraints, 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. Further, each device must be inserted manually into the system by a technician.

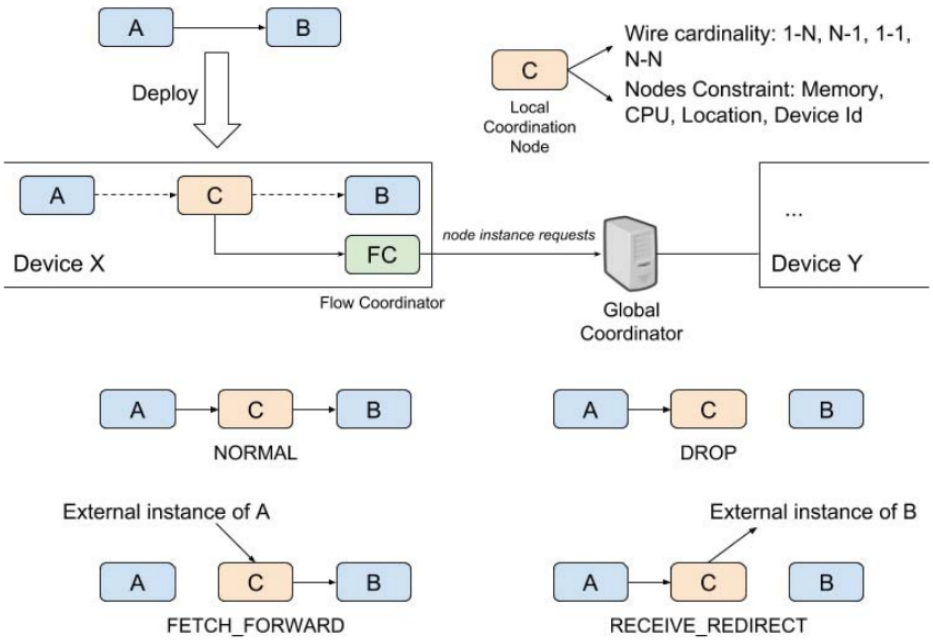

FIGURE 6 Coordination between nodes in D-NR[61].

Subsequent work to the previously mentioned tool focused on support for the Smart Cities domain. In a 2018 publication[61], the problems addressed were the deployment of multiple instances of devices running the same sub-flow, 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 synchronising the context of the device with the one given by the centralised coordinator. In Figure 6 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[62], support for CPSCN (Cyber-Physical Social Computing and Networking) was implemented, making it possible to facilitate the development of large scale CPSCN applications. Additionally, 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.

5.2 | FogFlow & uFlow

Another approach was made in the publication by Szydlo et al.[63], 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 run-time environment called *uFlow* that can be executed on a variety of resource-constrained embedded devices and the integration with the Node-RED platform.

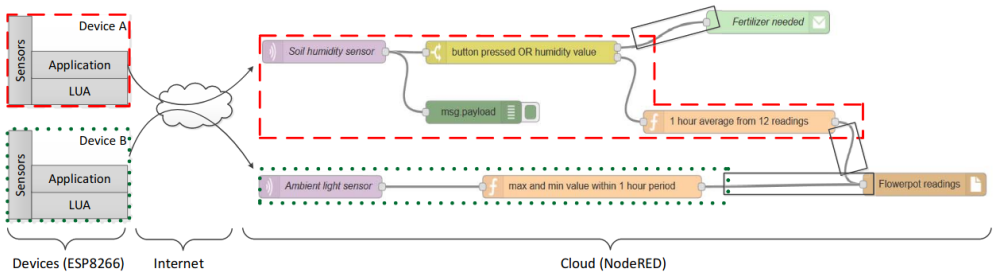

FIGURE 7 Partition and assignment of parts of the flow[63].

The solution consisted of 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 parts of the flow to be run on heterogeneous devices. All this is integrated with Node-RED. The communication between the devices is made only through the Cloud, with no support for peer-to-device communication. The results were promising, with a decrease in the number of measurements made by the sensors. However, there was room for improvement, with the automation of the decomposition and partitioning of the initial flow. Another improvement would be the detection bottlenecks which will move computations accordingly from the cloud to the fog.

Figure 7 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's goal is to measure soil humidity and ambient light. If a button is pressed or fertiliser is needed, an e-mail is sent to the gardener. The partition of computation is made with the assumption that the closer a selected process is 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 to be executed by *uFlow* and Node-RED. It is possible to observe in Figure 7 the results of the transformation process, where the parts of the flow surrounded by colour are executed in the device having the same colour.

In a new publication[64], 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 decentralisation

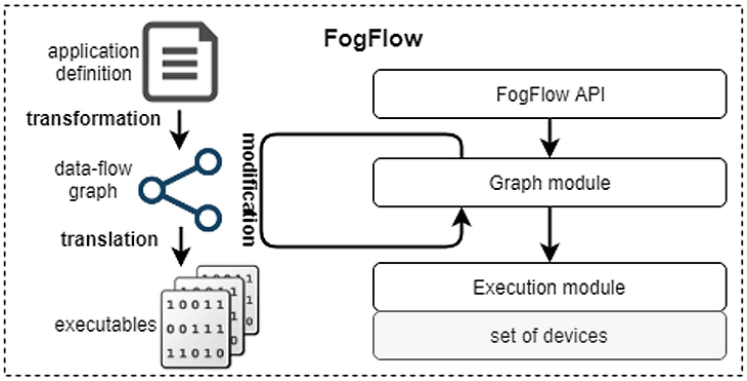

FIGURE 8 *FogFlow* architecture[64].

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 sets of devices with different capabilities. Several algorithms for flow decomposition are mentioned[65,66], but none were specified in terms of results. Figure 8 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.

5.3 | *FogFlow*

There is another tool with the same name *FogFlow* but created by Cheng et al.[67]. In the first publication related to this tool[68], the contributions made were the implementation of a standards-based programming model for Fog Computing and scalable context management. The first contribution consists in extending the dataflow programming model with hints to facilitate the development of fog applications. The scalable context management introduces a distributed approach, which allows overcoming the limits in a centralised 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, typically hosted in the Cloud, (2) Data Processing, present in cloud and edge devices and (3) Context Management, which is separated in a device discovery unit hosted in the Cloud and IoT brokers scattered in Edge and Cloud.

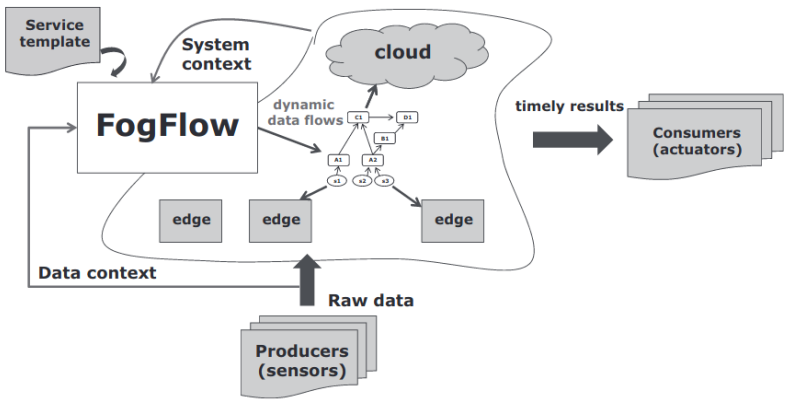

FIGURE 9 *FogFlow* high level model[69].

In more recent work[69], *FogFlow* was improved to deliver infrastructure providers with an environment that allows them to build decentralised IoT systems faster, with increased stability and scalability. The architecture can be seen in Figure 9, where dynamic data representing the IoT system 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 (a Docker image), and its assignment is based on (1) how many resources are available on each edge node, (2) the location of data sources, and (3) the prediction of workload. Edge nodes are autonomous since they can make their own decisions based on their local context, without relying on the central Cloud.

5.4 | DDFlow

DDFlow[42], first mentioned in Section 4, presents another distributed approach by extending Node-RED with a system run-time that supports dynamic scaling and adaption of application deployments. The coordinator of the distributed system maintains the state and assigns tasks 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).

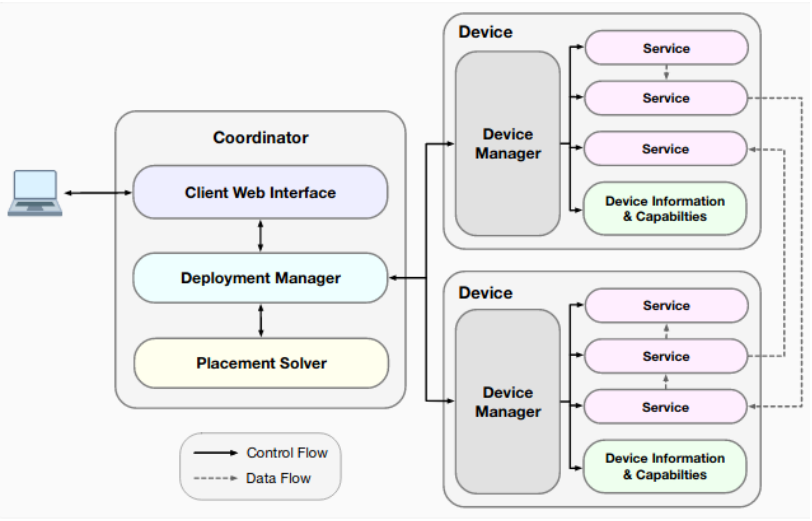

FIGURE 10 DDFlow architecture[42].

In a DDFlow system, each device has a set of capabilities and a list of services that correspond to an implementation of a *Node* (Fig. 10). 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 10: (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 minimising 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 to improve the reliability and fault-tolerance of the system.

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

5.5 | Comparison and Discussion

The mentioned tools were characterised based on their mentions or support for the following features and characteristics:

1. **Leverage devices.** A decentralised architecture takes advantage of the computational power of the devices in the network, assigning them tasks. However, some tools can have limitations on the type of devices, making constraints or only focusing on the devices of the Fog tier and not Edge.

2. **Capabilities communication.** The devices need to communicate to the orchestrator their capabilities so that it can make an informed decision regarding the decomposition and assignment of tasks.

3. **Open-source.** The license of software or tool is essential in terms of its usability. Open-source allows access to the code, making it possible for its analysis, improvement, and reuse.

4. **Computation decomposition.** To implement a decentralised architecture, it is important to decompose the computation of the system into independent and logical tasks that can be assigned to devices. This is made using algorithms, which can be specified or mentioned.

5. **Run-time adaptation.** A system needs to adapt to run-time changes, such as non-availability of devices or even network failure. The system notices these events and can take action to circumvent the problems and keep functioning.

TABLE 4 Small circles (•) mean *yes*, hyphens (-) means *no information available*, empty means *no* and asterisk (*) means more than one.

Tool	Leverage devices	Capabilities communication	Open-source	Computation decomposition	Run-time adaptation
DDF[40,46,61,62]	Limited[1]	•	•	Limited[2]	•
FogFlow & *uFlow*[63,64]	•	Limited[3]		Limited[2]	Limited[3]
FogFlow[67,68,69]	•	-	•	Limited[2]	•
DDFlow[42]	Limited[4]	•		Limited[2]	•

[1] Assumes that all devices run Node-RED, which limits the type of devices.
[2] Do not specify the algorithm used.
[3] Communication between devices is made through the Cloud.
[4] Assumes that all devices have a list of specific services they can provide.

From the analysis and the characteristics Table 4, we can be conclude that the current research in decentralised architectures in visual programming tools applied to IoT is incomplete. All the tools leverage the devices in the network but in different ways. DFF[40] assumes that all devices run Node-RED, which limits the type of devices that can be leveraged since it needs to have minimum resources to run it. *FogFlow* and *uFlow*[64,63] is the only tool that specifies how it truly leverages constrained devices, with the transformation of sub-flows into Lua code, with DDFlow[42] assuming that all devices have a list of specific services they can provide, that should match the node assigned to them.

Regarding the method used to decompose and assign computations to the available devices, DDFlow describes the process with the use of the longest path algorithm focused on reducing end-to-end latency between devices. *FogFlow* and *uFlow*[64,63] mention several algorithms that could be used, but do not specify which one was implemented. Both DDF[40] and *FogFlow*[68,69] do not specify the algorithm used besides some constraints but are the only tools with their source code accessible and with an open-source license. All the tools claim to have support for run-time adaptation to changes in the system, such as device failures.

6 | CONCLUSION

Several solutions are available that provide a decentralized architecture in visual programming tools applied to the Internet-of-Things paradigm. However, some of these tools solve specific problems or make assumptions regarding the scale of the system and the constraints of the devices. We can highlight the following research challenges that remain to be addressed by the research community:

1. **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.

2. **Communication of computational capabilities**: some of the current tools require the developer to manually introduce the resources of each device in the network, which is not a scalable solution. Others have a specific list of services, manually inserted that the devices can provide. Information about the computational capabilities of the devices in the network is vital for the successful distribution of computation across the devices.

3. **Detecting unavailability**: when a device fails or becomes unavailable, the system needs to realize and adapt automatically. The majority of current solutions do not possess this feature, which is vital if a system aims to adapt to changes in the environment dynamically.

4. **Code generation of sub-flows**: to truly leverage constrained devices, it is important to convert sub-flows or "tasks" into executable code. Devices that support simple firmware capable of executing code can be used to execute blocks of code, despite their limited capabilities.

5. **Provide self-adaption of the system**: devices can fail, as well as the connection between them or even the network. The system needs to discover and identify these changes and adapt to them at run-time in order to keep functioning.

Addressing these research challenges can improve the current state of the Internet-of-Things ecosystem, where systems are mostly centralized, leading to the proliferation of single point of failure (SPOF) (*e.g.*, dependency on the Cloud can render systems unusable if there is an Internet-connectivity problem). Further, high amounts of computational power are unused (*e.g.*, response times could be improved by using on-premises and already existent resources), and can address some pending security and privacy issues [14].

We should also mention that although Partha's survey [8] conclude that there exists some advantages of using visual programming languages, such as the ease of visualizing programming logic (which useful for rapid prototyping), and less burden on handling syntax error, as well as mentioning some negative aspects, such as the significant amount of time required for building simple IoT applications, we cannot find support or a rationale on how this observation results from the attributes they analysis (*e.g.*, licence and project repository).

References

1. Buyya R, Dastjerdi AV. *Internet of Things: Principles and Paradigms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 1st ed. 2016.

2. Alam T. A Reliable Communication Framework and Its Use in Internet of Things (IoT). 2018; 3.

3. Chen S, Xu H, Liu D, Hu B, Wang H. A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective. *IEEE Internet of Things Journal* 2014; 1(4): 349-359. doi: 10.1109/JIOT.2014.2337336

4. Zhang K, Han D, Feng H. Research on the complexity in internet of things. *IET Conference Publications* 2010; 2010(571 CP): 395–398. doi: 10.1049/cp.2010.0796

5. Burnett M, Kulesza T. End-User Development in Internet of Things: We the People. *In International Reports on Socio-Informatics (IRSI), Proceedings of the CHI 2015 - Workshop on End User Development in the Internet of Things Era* 2015; 12(2): 81–86.

6. Prehofer C, Chiarabini L. From IoT Mashups to Model-based IoT. *W3C Workshop on the Web of Things* 2013.

7. Chang SK. *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing Company . 2002

8. Ray PP. A Survey on Visual Programming Languages in Internet of Things. *Scientific Programming* 2017; 2017. doi: 10.1155/2017/1231430

9. Dias JP, Lima B, Faria JP, Restivo A, Ferreira HS. Visual Self-Healing Modelling for Reliable Internet-of-Things Systems. In: Springer; 2020: 27-36. To Appear.

10. Zhou Y, Zhang D, Xiong N. Post-cloud computing paradigms: A survey and comparison. *Tsinghua Science and Technology* 2017; 22(6): 714–732. doi: 10.23919/TST.2017.8195353

11. Bangui H, Rakrak S, Raghay S, Buhnova B. Moving to the edge-cloud-of-things: Recent advances and future research directions. *Electronics (Switzerland)* 2018; 7(11). doi: 10.3390/electronics7110309

12. Varshney P, Simmhan Y. Demystifying fog computing: Characterizing architectures, applications and abstractions. In: IEEE. ; 2017: 115–124.

13. Kleppmann M, Wiggins A, Hardenberg P, McGranaghan M. Local-first software: you own your data, in spite of the cloud. In: ; 2019: 154-178

14. Rawassizadeh R, Pierson T, Peterson R, Kotz D. NoCloud: Exploring Network Disconnection through On-Device Data Analysis. *IEEE Pervasive Computing* 2018; 17. doi: 10.1109/MPRV.2018.011591063

15. 1 IJ. Internet of things (iot) - preliminary report. *ISO,Tech. Rep.* 2014.

16. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems* 2012; 29. doi: 10.1016/j.future.2013.01.010

17. Nunes DS, Zhang P, Sá Silva J. A Survey on Human-in-the-Loop Applications Towards an Internet of All. *IEEE Communications Surveys Tutorials* 2015; 17(2): 944-965. doi: 10.1109/COMST.2015.2398816

18. Miao Yun , Bu Yuxin . Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid. In: ; 2010: 69-72

19. Linthicum DS. Connecting Fog and Cloud Computing. *IEEE Cloud Computing* 2017; 4(2): 18-20. doi: 10.1109/MCC.2017.37

20. Aazam M, Khan I, Alsaffar AA, Huh EN. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In: IEEE. ; 2014: 414–419.

21. Liu W, Nishio T, Shinkuma R, Takahashi T. Adaptive resource discovery in mobile cloud computing. *Computer Communications* 2014; 50: 119 - 129. Green Networkingdoi: https://doi.org/10.1016/j.comcom.2014.02.006

22. Martín Fernández C, Díaz Rodríguez M, Rubio Muñoz B. An Edge Computing Architecture in the Internet of Things. In: ; 2018: 99-102

23. Shi W, Pallis G, Xu Z. Edge Computing [Scanning the Issue]. *Proceedings of the IEEE* 2019; 107(8): 1474-1481. doi: 10.1109/JPROC.2019.2928287

24. Shi W, Dustdar S. The Promise of Edge Computing. *Computer* 2016; 49(5): 78-81. doi: 10.1109/MC.2016.145

25. Iorga M, Feldman L, Barton R, Martin MJ, Goren NS, Mahmoudi C. Fog computing conceptual model. tech. rep., 2018.

26. Shu NC. Visual Programming: Perspectives and Approaches. *IBM Syst. J.* 1999; 38(2–3): 199–221. doi: 10.1147/sj.382.0199

27. Burnett MM, Baker MJ, Bohus C, Carlson P, Yang S, Van Zee P. Scaling up visual programming languages. *Computer* 1995; 28(3): 45-54. doi: 10.1109/2.366157

28. Boshernitsan M, Downes M. Visual Programming Languages: A Survey. 1998.

29. Petersen K, Vakkalanka S, Kuzniarz L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 2015; 64: 1 - 18. doi: https://doi.org/10.1016/j.infsof.2015.03.007

30. Belsa A, Sarabia-Jacome D, Palau CE, Esteve M. Flow-based programming interoperability solution for IoT platform applications. In: ; 2018: 304–309

31. Ens B, Anderson F, Grossman T, Annett M, Irani P, Fitzmaurice G. Ivy: Exploring spatially situated visual programming for authoring and understanding intelligent environments. In: ; 2017: 156–163.

32. Ghiani G, Manca M, Paterno F, Santoro C. Personalization of context-dependent applications through trigger-action rules. *ACM Transactions on Computer-Human Interaction* 2017; 24(2): 14:1—-14:33. doi: 10.1145/3057861

33. Akiki PA, Bandara AK, Yu Y. Visual simple transformations: Empowering end-users to wire internet of things objects. *ACM Transactions on Computer-Human Interaction* 2017; 24(2): 10:1—-10:43. doi: 10.1145/3057857

34. Humble J, Crabtree A, Hemmings T, et al. "Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments. In: . 2864. ; 2003: 256-263

35. Valsamakis Y, Savidis A. Visual end-user programming of personalized AAL in the internet of things. In: . 10217 LNCS. ; 2017: 159–174

36. Pathirana D, Sonnadara S, Hettiarachchi M, Siriwardana H, Silva C. WireMe - IoT development platform for everyone. In: ; 2017: 93–98

37. De Luca G, Li Z, Mian S, Chen Y. Visual programming language environment for different IoT and robotics platforms in computer science education. *CAAI Transactions on Intelligence Technology* 2018; 3(2): 119–130. doi: 10.1049/trit.2018.0016

38. Bak N, Chang BM, Choi K. Smart Block: A Visual Programming Environment for SmartThings. In: . 2. ; 2018: 32–37

39. Fayed MS, Al-Qurishi M, Alamri A, Al-Daraiseh AA. PWCT: Visual language for IoT and cloud computing applications and systems. In: ; 2017

40. Giang NK, Blackstock M, Lea R, Leung VC. Developing IoT applications in the Fog: A Distributed Dataflow approach. In: ; 2015: 155–162

41. Tomlein M, Grønbæk K. A visual programming approach based on domain ontologies for configuring industrial IoT installations. In: ; 2017

42. Noor J, Tseng HY, Garcia L, Srivastava M. DDFlow: Visualized declarative programming for heterogeneous IoT networks. In: IoTDI '19. ACM; 2019; New York, NY, USA: 172–177

43. Kefalakis N, Soldatos J, Anagnostopoulos A, Dimitropoulos P. *A visual paradigm for IoT solutions development*. 9001 . 2015

44. Eterovic T, Kaljic E, Donko D, Salihbegovic A, Ribic S. An Internet of Things visual domain specific modeling language based on UML. In: ; 2015

45. Blackstock M, Lea R. FRED: A hosted data flow platform for the IoT. In: ; 2016

46. Blackstock M, Lea R. Toward a distributed data flow platform for the Web of Things (Distributed Node-RED). In: . 08-October. ; 2014: 34–39

47. Setiawan R, Anom Besari AR, Wibowo IK, Rizqullah MR, Agata D. Mobile visual programming apps for internet of things applications based on raspberry Pi 3 platform. In: ; 2019: 199–204

48. Besari ARA, Wobowo IK, Sukaridhoto S, Setiawan R, Rizqullah MR. Preliminary design of mobile visual programming apps for Internet of Things applications based on Raspberry Pi 3 platform. In: . 2017-Janua. ; 2017: 50–54

49. Tomlein M, Boovaraghavan S, Agarwal Y, Dey AK. CharIoT: An end-user programming environment for the IoT. In: ; 2017

50. Desolda G, Malizia A, Turchi T. A tangible-programming technology supporting end-user development of smart-environments. In: AVI '18. ACM; 2018; New York, NY, USA: 59:1—-59:3

51. Eun S, Jung J, Yun YS, So SS, Heo J, Min H. An end user development platform based on dataflow approach for IoT devices. *Journal of Intelligent and Fuzzy Systems* 2018; 35(6): 6125–6131. doi: 10.3233/JIFS-169852

52. Foundation O. Node-RED. Available: https://nodered.org/; 2020. Last access 2020. [Online].

53. Toolkit N. NETLabTK: Tools for Tangible Design. Available: www.netlabtoolkit.org/; 2020. Last access 2020. [Online].

54. NooDL . NooDL. Available: https://classic.getnoodl.com/; 2020. Last access 2020. [Online].

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

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

57. Ltd. GIIP. GraspIO. Available: https://www.grasp.io/; 2020. Last access 2020. [Online].

58. Wyliodrin . Wyliodrin. Available: https://wyliodrin.com/; 2020. Last access 2020. [Online].

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

60. Agarwal Y, Dey AK. Toward Building a Safe, Secure, and Easy-to-Use Internet of Things Infrastructure.. *IEEE Computer* 2016; 49(4): 88–91.

61. Giang NK, Lea R, Blackstock M, Leung VCM. Fog at the Edge: Experiences Building an Edge Computing Platform. In: ; 2018: 9-16

62. Giang NK, Lea R, Leung VCM. Exogenous Coordination for Building Fog-Based Cyber Physical Social Computing and Networking Systems. *IEEE Access* 2018; 6: 31740-31749. doi: 10.1109/ACCESS.2018.2844336

63. Szydlo T, Brzoza-Woch R, Sendorek J, Windak M, Gniady C. Flow-Based Programming for IoT Leveraging Fog Computing. In: ; 2017: 74-79

64. Sendorek J, Szydlo T, Windak M, Brzoza-Woch R. *FogFlow - Computation Organization for Heterogeneous Fog Computing Environments*: 634-647; 2019

65. NAAS MI, Lemarchand L, Boukhobza J, Raipin P. A Graph Partitioning-Based Heuristic for Runtime IoT Data Placement Strategies in a Fog Infrastructure. In: SAC '18. Association for Computing Machinery; 2018; New York, NY, USA: 767–774

66. Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R. 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* 2017; 47(9): 1275-1296. doi: 10.1002/spe.2509

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

68. Cheng B, Solmaz G, Cirillo F, Kovacs E, Terasawa K, Kitazawa A. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet of Things Journal* 2017; PP: 1-1. doi: 10.1109/JIOT.2017.2747214

69. Cheng B, Kovacs E, Kitazawa A, Terasawa K, Hada T, Takeuchi M. FogFlow: Orchestrating IoT services over cloud and edges. *NEC Technical Journal* 2018; 13: 48-53.