Flow-based Programming in Distributed Heterogeneous Environments

Hristo Filaretov
Technische Universität Berlin
Berlin, Germany

***Abstract*—The rapid spread of Internet-enabled devices has lead to a massive increase in data generation in distributed systems and specifically within the IoT domain. Additionally, many of the devices have limited communication interfaces and thus create complex distributed systems. The rising complexity of distributed systems coupled with the increased focus on handling large and varied data sources creates the need for a better programming paradigm, which allows the application developer to more rapidly create reliable and robust systems. In this paper the Flow-based programming paradigm is analyzed in the context of an Internet of Things/Fog Computing use case, where the dynamic and distributed nature of the system necessitates adaptability and robustness to the changing availability of resources and participating devices. Several prominent Flow-Based programming frameworks and libraries are evaluated in order to determine how they might simplify the development process and maintenance of a IoT/Fog Computing system or offer improvements when compared to conventional programming paradigms. Applications and systems with different needs are taken into account when evaluating the frameworks.**

I. Introduction

The proliferation of inter-connected devices is an ongoing trend, which has little indication of stopping. Sensors, actuators of all kinds and other kinds of resource-constrained, highly-specialized devices, which can communicate via different interfaces with the outside world have become commonplace and are being used in a plethora of different domains and use-cases [1].

The Fog computing paradigm is defined by the use of resource-constrained, heterogeneous devices on the edge of a cloud network. Its main goal is to offload some of the processing tasks the system has to execute from the main server to the participating edge devices, in order to improve the Quality of Service (QoS) and user experience of the application and make full use of the available resources on the devices [2].

Because Fog computing networks basically comprise Internet of Things (IoT) systems connected to bigger cloud-based applications [3], the term IoT will commonly be used throughout the rest of this paper in order to signify the kinds of systems that are relevant to the topic, while Fog computing will be mentioned and further explained where necessary.

The number of IoT devices is staggering and is constantly increasing. Furthermore, they create a heterogeneous environment - their communication interfaces and computer architectures vary, which presents a challenge to the system developer of an IoT application. The data generated by the devices is also different - many application rely on varied informational sources and types, such as humidity, sound and light sensors and many others. And while many applications rely on more and more data to function properly, due to the constrained nature of IoT devices, no feature-full application may be run on them. Instead, only some microservices or processes may be offloaded to the participating devices.

Taken together, all these factors lead to the need for a suitable programming paradigm for distributed heterogeneous environments, which decrease the tremendous complexity levied on the programmer. Flow-based programming (FBP) addresses the issues posed by the IoT paradigm by allowing the developer to structurally define the informational flow within a system - how the data should travel through the processes and what should be done with it, while separating the question of where each specific process should physically be executed. The FBP approach may leverage the Fog Computing paradigm by appropriately distributing the processes across the participating devices, according to their computational power and available resources.

In this paper, some of the most prominent

FBP programming tools are evaluated according to the requirements of the described IoT use case.

In the following chapter, Flow-based programming is defined in more detail and its applicability in IoT is further elaborated.

In section III, different IoT use cases and requirements are discussed.

In section IV, the evaluation criteria are outlined and explained, while in section V the frameworks and libraries are listed and described.

Afterwards, in section VI, all tools are compared and categorized according to their available features. Finally, a conclusion based on the evaluation is drawn.

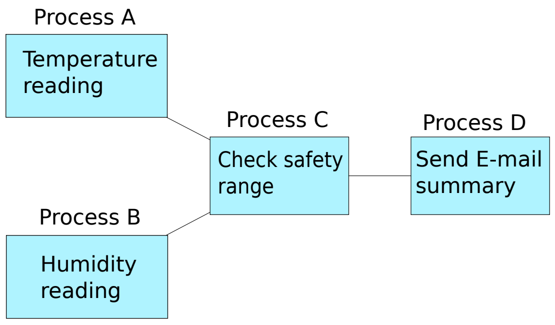


Fig. 1. Example FBP application

II. Flow-based programming

A. *Definition*

The flow-based programming paradigm was invented and defined by J. Paul Morrison in the 1960s while working for IBM and has been developed and further improved since [4], [5].

Flow-based programming is a dataflow programming method. It defines systems as directed graphs, with autonomous execution units serving as vertices and data connections as edges. All execution units are ”black box” processes, which can consume and create data. Every black box module is connected with other modules in order to form a coherent network.

Modules can be re-arranged and composed within a system without changing their internal functionality. This allows the developer of the application to ignore the exact implementation details of the module and focus on the higher-level structural representation of the system - how the data travels through the components, how it is modified and ultimately consumed.

In FBP jargon, data is transferred in ”Information Packets” (IPs) between processes. According to classic FBP, IPs have a defined lifetime and unique ownership - they either belong to a single process or are in the state of being transferred and thus owned by no process. This ensures that all processes can be executed in parallel, according to their local availability of data - once a process receives an IP, it can process it and output the result without ever having to wait or interact with another process in the meantime. This aspect is highly beneficial to Fog Computing systems where many different devices are constantly exchanging information - inter-dependent processes should be kept to a minimum and analyzed unscrupulously in order to minimize potential delays and bottlenecks.

Figure 1 offers a very simple example of a FBP application. In this example, two sensors are read periodically and the result is checked for validity. The final output of the system is an E-mail summary sent to a pre-specified address. This example is of course trivial, but it serves the purpose of showing the basics of FBP.

B. *Comparison to other programming paradigms*

This is similar to concepts in other programming paradigms. For example, in functional programming, functions are also composed in a similar manner to processes in FBP - this kind of composition clearly describes how the data is moved along the pipeline and is easier to reason about. The independence between processes (in FBP) and functions (in functional programming) reduces the amount of errors that developers are prone to make when using components with side-effects and timing issues. Nevertheless, FBP strays away from the purely functional paradigm by explicitly allowing processes to store stateful information between invocations.

Object-oriented programming (OOP) also shares many similarities with FBP. In OOP, objects of the same type may be instantiated and used, similar to how many FBP processes of the same type may be used in the same application. OOP objects and FBP processes also store their own state. The differences arise in the metaphors the two paradigms use for reasoning and developing applications. In OOP, objects usually support inheritance and are highly flexible, but do not carry any kind of implicit features that allows them to always interact, nor is the system defined in any other way. In FBP on the other hand, every process in the application has input and output ports and same kind of homogeneity between the processes is assured. Another big difference lies in the fact that FBP explicitly encourages visual programming where applicable, which is easier to understand for beginner programmers and very intuitive. Even though visual programming may become a crutch in more complicated and larger systems, it remains a very useful tool for quickly understanding existing code and systems [6], [7].

C. *Applicability in IoT*

The FBP paradigm describes the constituent parts of a system as a graph. It is useful to consider the exact meaning of this definition as applied to an IoT application. It might be tempting to imagine the nodes in the graph as the individual participating devices, however this kind of representation is fairly limiting - due to the fact that it ties hardware with the software component running on top of it, it is best suited for static constellations, where the devices are known before-hand and are expected to be available throughout the application lifetime. Much more useful (and natural to FBP) is describing the processes the system has to execute independently of the actual hardware involved, excluding any specific hardware interfaces and actuators that are explicitly needed, of course. This representation is much more useful and flexible, as it separates the questions of what the application should do and where it should be executed in order to achieve that.

Furthermore, it enables the programmer to reason about the application in a much more useful way - by describing the application in the high-level graph representation and then letting the FBP framework manage the runtime process and the exact execution location of each part. This move the decision-making process of where to run the processes to the on-line phase of the application and allows flexibility and robustness against failing devices or device over-exhaustion.

Another key point is that the frameworks reviewed in this paper will be evaluated not only based on their handling of FBP and how well a programmer can develop systems using them, but also by their additional features, which might simplify certain tasks, such as keeping the system running in the presence of errors or uncertainty, automatically adjusting for a change in resource availability or usage and other aspects discussed later.

III. Requirements of Distributed Heterogeneous Systems

Before analyzing the frameworks, the requirements of a system need to be clearly defined and examined. IoT systems exist in many different flavors and constellations with different requirements and properties. When considering the right tool for the job, some basic application characteristics should be defined.

The following aspects of a system are mainly related to distributed heterogeneous systems. Many other considerations and issues have to be considered when designing an application and choosing the correct tools, however those are outside the scope of this work.

A. Heterogeneity

First and foremost, IoT applications are usually composed of heterogeneous devices, or rather, heterogeneity is to be expected and always handled. Therefore, regardless what other properties the system exhibits, heterogeneity should always be accounted for. Many of the usual participating devices are resource-constrained or lack certain common communication interfaces, that modern server and desktop computers usually have available. Some of these devices are sensors, which only have a single type of interface, common ones include I^2C (for example many Adafruit sensors [8]) or WiFi (for example the ESP8266 [9]).

The available resources of every device also have to be taken into account - most servers for example are far more powerful than the end-device they are serving. Similarly, the processes that compose a system require different computational power. A good match between process requirements and device capability is paramount to any distributed IoT application. This is indeed one of the main tasks of the Fog computing paradigm and one that could be abstracted away from the programmer through a FBP framework.

In order to create a proper abstraction layer, the FBP framework may handle communication between heterogeneous devices transparently, in order to allow the developer to not only ignore the exact implementation of the underlying communication (while working on the structural abstraction level), but to entirely forgo thinking about where each process is executed. In order to do this, the FBP tool should handle heterogeneity in such a way, that it can not only establish a communication channel between two different devices, but also distribute the processes to the devices according to the available computational resources.

To summarize, heterogeneity entails two main considerations - communication between devices with different interfaces and the difference in available computational resources and hardware. Both tasks may be handled by a full-featured FBP framework and would lead to less cognitive load and fewer possibilities for mistakes from the side of the programmers.

B. System responsiveness

The application domain decides what kind of responsiveness the system should exhibit. Industrial grade robotics and manufacturing often require hard real-time systems which operate at 1 kHz, where delays and system exceptions are very hard or even impossible to work around. Nevertheless, with the advent of the Industry 4.0 paradigm, the Internet of Things has become a standard tool, even in the context of high-requirement systems. In use cases with high real-time requirements, network delays or slow information processing could seriously hinder the production process or lead to dangerous, undefined behavior. On the other hand, applications where there are no hard real-time requirements might not stop working if delay is introduced into the system, but their QoS would decrease significantly and their value would thus be diminished.

This problem is well-known and the difference between hard real-time and soft real-time systems is well documented. Nevertheless, every FBP tool should clearly express what kind of responsiveness and reliability it guarantees. This is especially important for safety-relevant real-time systems, but is also an important consideration when building a delay-sensitive user application. Not every IoT systems has strict real-time requirements and therefore not every FBP framework has to guarantee a low response time, but the reliability of the tool should nevertheless be clearly known to the developer.

C. System dynamicity and scalability

Another very important aspect of any system is its degree of hardware dynamicity, its expected rate of change of the participating devices. In the aforementioned example, industrial IoT systems are likely to rarely change in terms of their participating devices, thus making them easier to predict and design for. On the other hand, highly dynamic Fog computing applications, where participating devices are expected to often join or leave the network, are highly dynamic and difficult to plan for.

A further consideration is the expected amount of participating devices. Service scalability is an important aspect of an application with a varying amount of users and its importance cannot be overlooked especially when dealing with IoT use cases, where the amount of generated data can be quite staggering.

In order to best deal with both dynamicity and scalability, an application should always distribute the available workload properly across participating devices - this not only makes the best use of the available resources, but more importantly preserves the QoS guarantees of an application.

A FBP framework may solve dynamicity and scalability issues by simply using the participating devices as available resources for the processes of the system. This abstraction is certainly not trivial to implement, but would allow for a very powerful programming paradigm. Several issues have to be solved in order for this to be possible: devices should be able to leave the system at any point in time and the application should still function normally afterwards, i.e. no loss of data or other errors should occur; processes should be easily scalable and distributable across devices - the FBP paradigm makes this far easier, because every process is a black box with input and outputs and therefore a process may simply be cloned and the data split between all available similar processes. The communication overhead and data integrity would ideally be handled by the FBP framework.

D. Example use cases

1) Warehouse environment monitor: A very common IoT use case is monitoring warehouse conditions. Many warehouses have to maintain certain humidity, temperature, sound, light and other conditions. The system itself is quite simple - an appropriate number of sensors are set up in a warehouse and connected to a device, which collects their readings, parses the result and acts/warns based on the outcome. Such applications are usually not very dynamic - once some sensors have been connected and a central communication "hub" device (like a Raspberry Pi or other microcomputer) is activated, little to no change is expected and the system doesn't need to scale. System responsiveness is largely not an issue - whether a condition reading is executed a few seconds earlier or later is of little concern (for conventional warehouse needs). The only source of heterogeneity is the difference between the used sensors and the hub device and is of limited concern, because it is static in nature and thus easier to work around.

This is an example of a relatively trivial system, which can adequately be programmed with conventional programming tools, but would be far more intuitive and quicker to develop with visual FBP-based tools.

2) Public park environment monitoring: This example is similar to the last, except the humidity and temperature conditions of a park are monitored based on the personal smartphones of people walking in a park (assuming their participation is possible). The responsiveness of the system is once again not an issue - the exact timing of sensor readings is of little concern. However, the system is far more dynamic, because people are constantly entering and exiting the confines of the park and their exact number at any point in time is not fixed and scalability should also be accounted for. Additionally, the devices are somewhat heterogeneous in nature.

This example is more difficult to develop than the previous one, even based solely on the dynamicity of the system. Nevertheless, a good FBP framework would let a developer structure the informational flow between the participating smartphones and the database that collects the information.

3) Communication between neighboring smart vehicles: Assuming a scenario where smart vehicles on the road can communicate with each other and share vital information about the traffic, as well as receive information from other actors on the road, like traffic lights, an example may be conceived where vehicles might share crucial, time-sensitive information with each other, such as whether an emergency stop is required or whether the weather conditions are worse than anticipated.

In this fictional example, system response is vital - if an emergency stop is issued, all affected devices should immediately stop. Dynamicity and scalability are also large issues - vehicles would constantly be in flux when communicating with each other. Heterogeneity may be an issue because of the communication interfaces between the vehicles and traffic lights. Furthermore, a resource-heavy task that simulates different maneuvers and tries to find an optimal set of actions for all vehicles would be a good example of a process that should be executed on a powerful server instead of on a embedded device in a family car, for example.

This scenario may also be easily expressed as a flow graph, by modeling the communication between a vehicle, nearby traffic lights, neighboring vehicles and the ”decision-making” process.

These scenarios only serve as examples of how different systems might be modeled with the FBP paradigm and how it simplifies the associated development process.

IV. Evaluation criteria

Evaluation of the reviewed frameworks is mainly based on how they handle and deal with task decomposition, distribution and orchestration. The (QoS) aspect is taken into consideration for each of the evaluation criteria. Notable features of every framework or library are also explained and highlighted.

A. Task decomposition

The exact way in which the application is separated into its constituent parts. As earlier described, a FBP application is a network of connected processes and every process must be defined. This is usually manually done by the programmer and depending on their design choices, the granularity of the application is defined.

This process could possibly be improved by letting the FBP framework analyze the components the programmer has used in the application and merging or separating them further. For example, a sequence of three components that filter data might be run on three separate devices or simply be abstracted by representing the filters as a single black box with the inputs of the first filter and the outputs of the third. This could also be achieved by other means during the runtime of the application (discussed later).

B. Task distribution

This is the act of transferring the components to the participating devices. Once again the distinction between the processes of the application and the underlying hardware should be made - the participating devices are viewed as resources assigned to processes. It is through this abstraction and viewpoint that optimization and intelligent distribution is possible. Potential decision factors in this step include the expected memory usage of a process, its required hardware components, CPU intensity of its task, physical distance and network delay to other components/users.

C. Task orchestration

Finally, once the task has been initialized and distributed, the execution begins and the data can flow through the application. During the lifetime of the application, errors have to be handled, changes in the available resources have to be taken into account and compensated, joining and leaving devices have to be considered. Additionally, a certain QoS may be guaranteed or targeted by the framework.

D. Other considerations

Not without merit is the popularity of a project - this usually corresponds to a more active community and more available documentation and support for the framework, which in term allows for a more rapid development cycle and easier navigation through the framework - whether this means learning how to use it or avoiding pitfalls. All of the frameworks and libraries reviewed in this work are open-source, thus ensuring that proper inspection and understanding can be reached, if desired.

Furthermore, it is very important to consider whether a framework or library can improve the performance of the system - critical aspects such as latency (which impacts user experience and is a main consideration for IoT systems) or robustness to errors are common issues, which could be targeted by specific tools and alleviated as problems.

The application domain of every tool is also examined, i.e. what kind of use case it is best suited for (web development, IoT, etc.). Of course, all reviewed tools may be used in an IoT context, nonetheless not all of them are tailored for this specific use case.

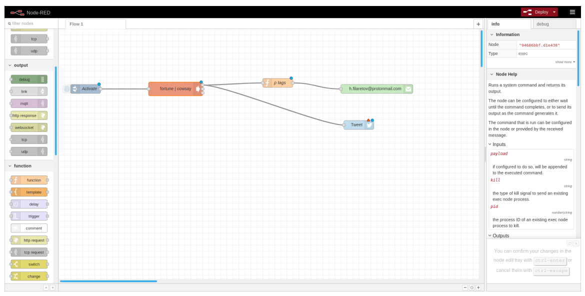
V. Reviewed Frameworks and Libraries

A. Node-RED

Node-RED is an FBP runtime and IDE. It was developed by IBM and subsequently released as free software under the Apache 2.0 License. It is based on NodeJS [10].

Essentially, Node-RED is a NodeJS server that executes and manages the programmed "flow". A web-based IDE, which by default is automatically started when running Node-RED is the main tool to develop and interact with the system (see fig. 2).

Node-RED is one of the most prominent FBP tools featured in scientific literature (see [6], [11]–[17] and others). It is also widespread and common among hobbyists and independent developers as well (with over 6000 "stars" and 1400 forks on Github [18]). Its popularity also contributes to the large amount of nodes already available and usable through the npm package



Node "94686bbf.d1e438"
Type exec

Runs a system command and returns its output.

The node can be configured to either wait until the command completes, or to send its output as the command generates it.

The command that is run can be configured in the node or provided by the received message.

payload string
if configured to do so, will be appended to the executed command.

kill string
the type of kill signal to send an existing exec node process.

pid number|string
the process ID of an existing exec node process to kill.

You can confirm your changes in the node edit tray with ctrl-enter or cancel them with ctrl-escape

Fig. 2. Screenshot of the Node-RED Web IDE

manager - including IoT relevant nodes such as I^{2}C interface nodes.

Node-RED's popularity is understandable - it is a very intuitive and simple tool. Many of the aforementioned papers describe setting up systems with different constellations and requirements with Node-RED and their experience learning and using the tool in production, however most of their applications are static in nature - the participating devices don't change and no intelligent task distribution is required. Many IoT systems are indeed static in nature and for those cases, assumptions of long-running devices and low to none dynamicity are appropriate, however the limitations of this approach have to be considered and understood - Node-RED does not allow a developer to run tasks within the flow on different machines by itself, this kind of functionality must either be recreated by hosting external processes on the participating devices and then communicating with them by MQTT nodes (for example), or by using extensions for Node-RED, which allow this kind of behavior. The most well-documented such extension is Distributed Node-RED.

B. Distributed Node-RED

Distributed Node-RED [19], [20] is an extension for "vanilla" Node-RED that provides the ability to execute a Node-RED flow on different machines.

The features that Distributed Node-RED (DNR) adds to the vanilla version are noteworthy enough to warrant their own section in this paper.

The first iteration of DNR [20] simply add the ability to assign a device ID to a node in a Node-RED flow in order to execute the node on a specific device. This already marks a great leap forward in terms of distributed capabilities for Node-RED, however no additional guarantees of efficiency are given to the developer - manual assignment means that all optimizations and considerations in terms of task distribution must be handled by the developer and are static in nature - no additional changes during the runtime of the application are supported.

Following Blackstock's initial work, Giang et al. have further developed DNR, granting it additional features, which match the requirements outlined earlier in this paper [19]. Support for migration of tasks has been added, as well as constraint-satisfaction based initial distribution of tasks.

These two features alone significantly contribute to DNR's applicability in IoT use cases. Constraint-satisfaction problems are well-understood and allow the developer to carefully consider the limitations of each participating device, while letting the framework itself decide where each node should be executed. Any kind of error-proneness of human decision-making is thus alleviated (even though the developer is free to assign a task to a specific device as well - this however nullifies the aforementioned benefits).

Live migration is handled in the following manner: each participating device is sent the whole flow and a local parser is responsible for deciding whether a node should be run locally on the machine or whether its inputs and outputs should be connected to external processes. This way, no distribution of nodes is required during the runtime of the application, as every machine has the necessary information to run every node (however, if a node has requirements, which cannot be fulfilled by the machine, it still cannot be executed, of course).

C. Calvin

Calvin was developed by Ericsson Research [21] as a runtime specification meant for IoT systems. In contrast to the other frameworks reviewed in this paper, Calvin is primarily a specification - while an implementation in Python is available [22], the runtime may be implemented in any language. Indeed, the authors of the paper mention Erlang as a good candidate, possibly because of its close parallels to FBP and intrinsic distributed nature [23], as well as Scala, in order to use the ubiquitous Java Virtual Machine.

D. Calvin Constrained

Calvin Constrained [24] is an implementation of Calvin developed for constrained devices. It is a proof of concept of how different Calvin implementation may operate together thanks to the common interface.

Calvin Constrained is written in the C programming language and support both C actors as well as MicroPython actors.

E. NoFlo

NoFlo is a FBP NodeJS framework available as free software under the MIT License [25]. It is currently under the umbrella of Flowhub [26].

NoFlo is very similar to NodeRED in many ways: they are both implemented on top of the Javascript based NodeJS framework, both inherit their concurrency from it, both have a web-based GUI for visual programming and have community built around them. However, NoFlo seems to be used far more often for web-related tasks, as evidenced by the available modules on their web page and based on the available scientific literature - there is a distinctive lack of mention of NoFlo in papers related to the IoT and distributed systems.

Nevertheless, Flowhub also host and maintain MsgFlo [27] with the goal of allowing "polyglot FBP systems", however the project is currently used only in hobbyist applications and no extensive scientific examination of the library is available. As such, it is ill-suited to be used in production environments.

NoFlo is also usable as a standard Javascript library within a project. While this certainly allows the programmer to use a subset of the framework and bypass the visual programming IDE, the use of Javascript callbacks might be a hindrance for more complex concurrent and distributed applications.

F. Related works

The frameworks described above define how a system should be structured and what kind of technologies it should use (to a varying degree). However, there do exist scenarios where the feature-set of these frameworks is too large and involved and a leaner solution might be required. Such use cases include advanced scenarios, where the programmer might want to implement many components of the system herself or otherwise wield more control over the application. On the other hand, hobbyist and home-automation scenarios, where simpler requirements are set may also fall into this category (in this scenario the system is rarely highly dynamic, the relationships between processes are relatively trivial and the participating devices are known beforehand). This also includes educational or academic use cases, where the applicability of FBP might be tested with a simpler scenario.

In order to accommodate such requirements, smaller and simpler libraries are available. Some are listed below.

1) GoFlow: GoFlow [28] is a FBP library written in the Go programming language. It is a generic FBP library, which implements many of the features classical FBP defines. The library itself does not handle distributed communication or other more advanced features, it simply allows the user to define black box units and connect them in an application graph.

2) Flowex: Flowex [29] is a FBP library written in the Elixir programming language. Similarly to GoFlow, it is a generic library aimed at giving the developer the opportunity to model a system in an FBP-inspired manner. Elixir runs on the Erlang virtual machine and as noted earlier, Erlang is well suited for systems where processes communicate between each other. Another feature of Elixir well-suited for FBP style applications is the concept of back-pressure, which allows a system to dynamically adapt the workload of processes according to how heavily they're being used in a certain moment. Should this feature be extended or applied in a distributed system, it would certainly be worthy of examination and further research.

3) JavaFBP: JavaFBP is a FBP library written by John Cowan and currently maintained by J. Paul Morrison [30]. If compatibility with the classic FBP definition is of any importance to a developer, they would certainly be well accommodated by one of the libraries hosted on J. Paul Morrison's own GitHub page [31].

4) IFTTT: If This Then That [32], [33] is a service/application which allows users to connect triggering events and actions in order to create the kind of "informational flow" typical to FBP. This is a very simple service, applicable mostly to hobbyist and everyday users who wish to automate certain events, such as reacting to E-mails, Twitter posts [34] and other popular services. IFTTT relies on public APIs to connect services and only services connected to IFTTT by the developers can be used. This makes it fairly limiting for developers, but very simple and user-friendly for the casual user. This is the only non-open-source tool reviewed in this paper and is excluded from the final comparison.

VI. Comparison

All of the tools reviewed in this paper require manual task decomposition - the programmer is responsible for defining the execution units of the application.

A. Applicable domain

Table I offers a brief overview of the domain each tool is best suited for. Distributed Node-RED and Calvin are the best fits for the IoT use case that concerns this paper, however other tools are simpler to use and can also be reasonably applied in scenarios with less demanding requirements. A very important decision criterion when evaluating tools is how much upfront work and effort is required to set it up and to maintain it afterwards. While the ideal case of choosing the most feature-full tool regardless of difficulty might be appealing, it often leads to complications which compromise the quality of the application.

TABLE I
COMPARISON OF FRAMEWORK DOMAIN

Tool	Domain	Type	Language
Node-RED	IoT/Web	Framework	Javascript
DNR	IoT	Framework	Javascript
NoFlo	Web	Framework	Javascript
Calvin	IoT	Framework	Python
GoFlow	Generic	Library	Go
Flowex	Generic	Library	Elixir
JavaFBP	Generic	Library	Java

B. *Communication and distribution*

In table II the method of communication between devices and instances of the tool and distribution aspect of each tool is listed. Characteristically, all of the frameworks have some type of predefined communication preferences, while the libraries lack such features and are thus subject to extension by the programmer (all tools with N/A as their communication method may indeed communicate with other devices, but only if this behavior is explicitly programmed by the developer). This is a pivotal difference when choosing the correct framework. Should the predefined choice of the framework (such as MQTT for DNR) meet the requirements of the project, then the work of the programmer is simplified and taken care of. Otherwise, trying to work around the framework's native choice might prove more error-prone than developing the communication interface from the ground up. In Calvin's case, the freedom to choose the exact implementation somewhat alleviates this problem, but the burden of correct implementation falls once again on the programmer. Differences like this should not be underestimated when choosing the tool, because they also define the abstraction layer the programmer will be working on and as noted by Giang et al. in [19], FBP tools are most valuable by guaranteeing correct behavior below the graph-level representation of the system and allowing the programmer to reason and define the system on a higher abstraction level.

Distribution is a simpler choice in this case - whether it is needed or not should be relatively clear in the design phase of an application and how crucial it is to automate it and guarantee the best possible distribution would be the leading decision-making factors. DNR and Calvin allow for Constraint Satisfaction Problem (CSP) solving methods of task distribution, where every process is sent to the most appropriate device according the available resources, such as computing power and hardware components. Even this method has a manual component - the programmer is currently responsible for describing the requirements of each process and resources of every device and is therefore also prone to human-enacted errors. It is not infeasible to automate this process in the future, however this is still not available.

TABLE II
COMPARISON OF FRAMEWORK COMMUNICATION BETWEEN INSTANCES AND DISTRIBUTION

Tool	Communication	Distribution
Node-RED	N/A	None
DNR	MQTT	CSP
NoFlo	N/A	Manual
Calvin	Adjustable	CSP
GoFlow	N/A	None
Flowex	N/A	None
JavaFBP	N/A	None

C. *Orchestration and QoS*

The most tantalizing aspect of a FBP framework is adaptability and robustness in the presence of a dynamic system - the ability to properly re-distribute execution units to new devices as old ones leave and new ones join the system, as well as reactively adapting the workloads. Such features are difficult to implement but would greatly increase the usefulness of IoT systems and specifically Fog computing networks. This aspect of FBP systems is called orchestration. Only two of the reviewed tools implement some kind of advanced orchestration - Distributed Node-RED and Calvin. In DNR, nodes may be switched on and off on specific devices during runtime, while Calvin allows for live migration of execution units. Switching processes on and off on devices and migrating them across devices are basic requirements for further improvements in this area. Both systems also allow for load-balancing using this method, but this has to be implemented by the programmer.

Orchestration also deals with the QoS of a system. As earlier defined, this is the ability of an application to guarantee certain performance benchmarks, such as maximum delay the user may experience, affordable failures and how to recover from them and other features which influence the perceived quality by the user. Flow-based programming tools can only indirectly improve the QoS aspect of an application, because the underlying technology and software used by them is the same as the ones a programmer would utilize when writing the application manually. The advantage comes in the form of correctness guarantee - by using pre-made solutions (for example, nodes in Node-RED or, more generally, the NodeJS server implementation of Node-RED), the programmer avoids the cognitive load of correctly implementing the same solutions, which is usually necessary when writing an application with conventional programming methods. This is a classic trend in programming and engineering in general - the best practice techniques are encapsulated in the form of predefined methods or software libraries and subsequently used where appropriate. This ensures correct behavior and simplifies the development of any application. Similarly, FBP allows the programmer to concentrate on the structural aspect of a system, as opposed to the lower-level implementation of communication sockets, processing nodes and similar.

This improvement of QoS may be obtained by using a framework which has a thriving library of already written and maintained components (a precondition for this is the active maintenance of the project, as well as its popularity and usual application domain - both of which are discussed in this chapter) or which strictly abides by certain rules in order to guarantee correct and optimal behavior.

Few of the evaluated tools exhibit such features. Node-RED is by far the most popular framework and is being very actively maintained, and while offering many different nodes, offers no guarantees about the temporal behavior of the system, nor does it clearly define how to handle errors and whether some kind of QoS guarantee can be given. Nevertheless, a programmer is far less likely to introduce software errors in the system when using pre-written modules. This is a benefit worthy of consideration when choosing which tool to use for any given application development process.

Distributed Node-RED can adapt the execution of its flow live and uses the MQTT protocol to communicate between instances. Because of this reliance on MQTT, some kind of QoS measure regarding the latency of the system may be estimated. MQTT is a very popular messaging protocol, especially in the IoT sphere and has been extensively tested [35]. Therefore, some approximation of its performance may be given depending on the MQTT QoS level used in the system.

Calvin itself does not address the issue of QoS either - the behavior of an application using Calvin is predominantly dependent on its specific implementation and is thus a responsibility of the programmer. In principal, however, a communication library or component may be written for Calvin, which may guarantee or estimate a certain QoS.

NoFlo acts similarly to Node-RED - no guarantee of performance is given. The developer must test and confirm the reliability of the system herself. If software errors occur, they are at least gracefully caught and left to be handled by the programmer.

Table III offers an overview of this comparison.

TABLE III
ORCHESTRATION AND QOS SUPPORT IN THE COMPARED FRAMEWORKS

Tool	Orchestration	QoS Assurance
Node-RED	Static	Little to none
DNR	Dynamic	Some (based on MQTT)
NoFlo	Static	Little to none
Calvin	Dynamic	Implementation dependent
GoFlow	N/A	None
Flowex	N/A	None
JavaFBP	N/A	None

D. Project status and maintenance

Another important consideration for a piece of software is its maintenance - or lack thereof. Specifically, when choosing a library or framework which handles the structural integrity of the system - how the individual components are connected and how reliably they work together, and the means of communication (mainly in the case of frameworks), constant maintenance and updating of the software is crucial. Security vulnerabilities and bugs must be solved in a timely manner and quickly deployed downstream. For open-source software, this aspect is somewhat less of a worry, because a developer can inspect and modify the source code of the framework. Nevertheless, this requires additional work and understanding on the part of the programmer, which complicates the development process. Therefore, it is necessary to take into account how active a software project is before including it in an application. Figure 3 shows the frequency of Google searches for the specific terms listed in the graph. As is evident, Node-RED is the most popular of all reviewed frameworks. This is also backed up by the number of participants and active subscribers to the Github pages of each project. Most of the project seam to receive regular updates, but the

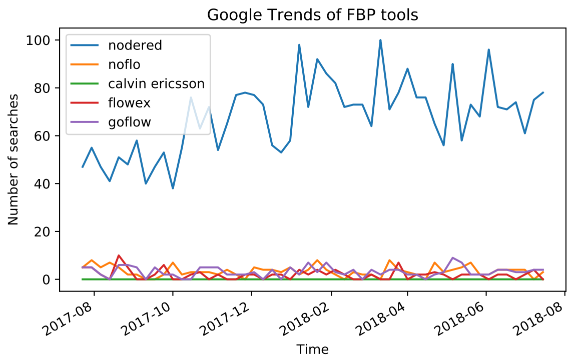


Fig. 3. Google trends plot of FBP tools

volume of users Node-RED has is an advantage - a bigger community means more real-world tests, which can help a programmer get started with a framework.

VII. FURTHER DEVELOPMENT AND RESEARCH

Many questions are left unanswered by the body of scientific literature available today and many of the features and criteria outlined in the beginning of this paper are left unfulfilled. One possible avenue for improvement lies in the algorithmic analysis of an IoT system. Chaturvedi et al. [36] attempt to optimize a flow within such a system by using a relatively simple approach. A practical application of such an algorithm is, as of the writing of this paper, lacking. Privacy and security are also often overlooked in many IoT related scientific endeavors and frameworks. Kawamoto et al. [11] offer a modification of Node-RED which better suits modern user privacy considerations and requirements.

Many of the criteria defined in the beginning of this paper are left unfulfilled, especially features regarding orchestration and QoS assurance. The difficulty of creating such a system is further exacerbated by the heterogeneity of Fog computing networks and the constrained nature of the participating devices. Nevertheless, such a unifying framework or at least idiomatic techniques when developing similar systems are crucial in order to ensure the correct operation and acceptable user experience of IoT applications.

VIII. Conclusion

The main role of FBP frameworks and libraries in the development process is simplifying the design and implementation of the application and allowing the developers to rely on tested and well-thought out solutions to common problems, such as managing dynamically changing participating devices and recovering from unexpected changes in the system behavior, thus ensuring a better functioning system when compared to one where the programmers are responsible for implementing every component.

There are many available ”classic” FBP and FBP-inspired frameworks and libraries available to developers, but only a few of those are readily applicable to the IoT and Fog Computing use case. Distributed Node-RED and Calvin hold the greatest promise in terms of features a developer might heavily rely on (such as the ones mentioned above).

For cases in which the developer might require more flexibility or the opportunity to develop their own solutions, leaner libraries exist, which serve the goal of connecting components in an FBP-like manner, but leave handling communication, distribution and orchestration to the programmer.

Acknowledgments

Thanks to Cem Akpolat for his supervision and guidance in writing this paper and the research process, and to Dr. Stefan Fricke for hosting the Intelligent Software Systems seminar.

References

[1] B. Al-Shargabi and O. Sabri, “Internet of things: An exploration study of opportunities and challenges,” in *2017 International Conference on Engineering MIS (ICEMIS)*, pp. 1–4, May 2017.

[2] S. B. Nath, H. Gupta, S. Chakraborty, and S. K. Ghosh, “A survey of fog computing and communication: Current researches and future directions,” *CoRR*, vol. abs/1804.04365, 2018.

[3] R. Mahmud and R. Buyya, “Fog computing: A taxonomy, survey and future directions,” *CoRR*, vol. abs/1611.05539, 2016.

[4] J. P. Morrison, *Flow-based Programming, 2nd edition*. 2011.

[5] J. P. Morrison, “Flow-based programming.” http://www.jpaulmorrison.com/fbp/. Accessed: 2018-07-10.

[6] Z. Chaczko and R. Braun, “Learning data engineering: Creating iot apps using the node-red and the rpi technologies,” in *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pp. 1–8, July 2017.

[7] D. Mason and K. Dave, “Block-based versus flow-based programming for naive programmers,” in *2017 IEEE Blocks and Beyond Workshop (B B)*, pp. 25–28, Oct 2017.

[8] Adafruit, “Adafruit industries.” https://www.adafruit.com/. accessed: 2018-07-12.

[9] Espressif, “Esp8266 overview.” https://www.espressif.com/en/products/hardware/esp8266ex/overview. Accessed: 2018-07-12.

[10] Linux Foundation, “Nodejs.” https://nodejs.org/en/. accessed: 2018-07-12.

[11] J. Kawamoto, “An implementation of privacy preserving stream integration system,” in *2016 International Conference on Information Networking (ICOIN)*, pp. 57–62, Jan 2016.

[12] R. K. Kodali, S. Mandal, and S. S. Haider, “Flow based environmental monitoring for smart cities,” in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 455–460, Sept 2017.

[13] M. Leki and G. Gardaevi, “Iot sensor integration to node-red platform,” in *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–5, March 2018.

[14] A. Rajalakshmi and H. Shahnasser, “Internet of things using node-red and alexa,” in *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 1–4, Sept 2017.

[15] 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)*, pp. 74–79, June 2017.

[16] J. A. C. Yatan, A. K. Mahamad, S. Saon, M. A. B. Ahmadon, and S. Yamaguchi, “Earthquake monitoring system,” in *2017 IEEE International Symposium on Consumer Electronics (ISCE)*, pp. 61–64, Nov 2017.

[17] J. Zaman and W. D. Meuter, “Crowd sensing applications: A distributed flow-based programming approach,” in *2016 IEEE International Conference on Mobile Services (MS)*, pp. 79–86, June 2016.

[18] Node-RED, “Node-red github page.” https://github.com/node-red/node-red. accessed: 2018-07-12.

[19] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung, “Developing iot applications in the fog: A distributed

dataflow approach," in *2015 5th International Conference on the Internet of Things (IOT)*, pp. 155–162, Oct 2015.

[20] M. Blackstock and R. Lea, "Toward a distributed data flow platform for the web of things (distributed node-red)," in *Proceedings of the 5th International Workshop on Web of Things*, WoT '14, (New York, NY, USA), pp. 34–39, ACM, 2014.

[21] P. Persson and O. Angelsmark, "Calvin merging cloud and iot," *Procedia Computer Science*, vol. 52, pp. 210 – 217, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[22] Ericsson Research, "Calvin github page." https://github.com/EricssonResearch/calvin-base. Accessed: 2018-07-15.

[23] J. Armstrong, *Making reliable distributed systems in the presence of software errors*. PhD thesis, 2003.

[24] A. Mehta, R. Baddour, F. Svensson, H. Gustafsson, and E. Elmroth, "Calvin constrained; a framework for iot applications in heterogeneous environments," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1063–1073, June 2017.

[25] NoFlo, "Noflo github page." https://github.com/noflo/noflo. Accessed: 2018-07-15.

[26] Flowhub, "Flowhub ug." https://flowhub.io/about/. Accessed: 2018-07-15.

[27] MsgFlo, "MsgFlo Github page." https://github.com/msgflo/msgflo. Accessed: 2018-07-22.

[28] Vladimir Sibirov, "GoFlow github webpage." https://twitter.com/. Accessed: 2018-07-15.

[29] Anton Mishchuk, "Flowex github webpage." https://github.com/antonmi/flowex/. Accessed: 2018-07-15.

[30] J. P. Morrison, "Javafbp github page." https://github.com/jpaulm/javafbp. Accessed: 2018-07-15.

[31] J. Paul Morrison, "J. Paul Morrison's Github Page." https://github.com/jpaulm. Accessed: 2018-07-22.

[32] IFTTT, "IFTTT webpage." https://ifttt.com/. Accessed: 2018-07-15.

[33] E. Ackerman, "IFTTT: San Francisco Startup Lets Anyone Control The Internet of Things." https://www.forbes.com/sites/eliseackerman/2012/09/23/ifttt-the-san-francisco-startup-lets-anyone-control-the-internet-of-things/#1388a1d85abd, 2012. Accessed: 2018-07-15.

[34] Twitter, "Twitter webpage." https://twitter.com/. Accessed: 2018-07-15.

[35] P. Ferrari, A. Flammini, E. Sisinni, S. Rinaldi, D. Brando, and M. S. Rocha, "Delay estimation of industrial iot applications based on messaging protocols," *IEEE Transactions on Instrumentation and Measurement*, pp. 1–12, 2018.

[36] S. Chaturvedi, S. Tyagi, and Y. Simmhan, "Collaborative reuse of streaming dataflows in iot applications," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, pp. 403–412, Oct 2017.