A preliminary study of open-source IoT development frameworks

Zeineb Baba-Cheikh
École de technologie supérieure
Montreal, Canada

Ghizlane El-Boussaidi
École de technologie supérieure
Montreal, Canada

Julien Gascon-Samson
École de technologie supérieure
Montreal, Canada

Hafedh Mili
Université du Québec à Montréal
Montreal, Canada

Yann-Gael Guéhéneuc
Concordia University
Montreal, Canada

ABSTRACT

The Internet of Things (IoT) market is growing fast with an increasing number of connected devices. This led many software companies to shift their focus to develop and provide IoT solutions. IoT development has its own challenges as typical IoT solutions are composed of heterogeneous devices, protocols and software. To cope with these challenges, many frameworks are available to help developers to build IoT applications. Some of these frameworks are open source and might be of great interest for small and medium-sized companies wishing to build IoT solutions at a lower cost. In this paper, we present the results of a preliminary study of four open source IoT development frameworks. In particular, we used these frameworks to implement a sample of three IoT applications and we analyze them against a minimal set of IoT requirements. We focus in our study on the IoT development for Raspberry PI as it is a very low-cost and popular platform.

KEYWORDS

Internet of Things, open-source IoT development frameworks, IoT applications, Raspberry Pi

ACM Reference Format:
Zeineb Baba-Cheikh, Ghizlane El-Boussaidi, Julien Gascon-Samson, Hafedh Mili, and Yann-Gael Guéhéneuc. 2020. A preliminary study of open-source IoT development frameworks. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3392198

1 INTRODUCTION

The Internet of Things (IoT) is a network that interconnects a plethora of physical devices [2]. These devices typically fall under two categories: *sensors*, which collect data from the physical world, and *actuators*, which perform actions to alter the state of the physical world. The IoT landscape has grown at a considerable rate over the past few years. According to recent industrial studies and reports, 20,6 billion devices are now connected [3] and 5 quintillions bytes of data are produced daily by IoT devices [15]. This led software companies to shift their focus to building IoT solutions.

Unlike traditional software applications which rely on traditional computing infrastructures (e.g., cloud, servers, desktop and laptop computers), IoT applications rely on devices that are widely heterogeneous. For instance, devices on the low end of the spectrum are made of simple micro-controllers, while higher-end devices (e.g., Raspberry Pi, Beaglebone) can have multi-core processors, several gigabytes of memory and can execute full operating systems. Similarly, the IoT ecosystem also features a *high heterogeneity of applications* that span across several application domains and industries. This drives the need for approriate software development tools, practises and frameworks that can efficiently cope with the software and hardware diversity of the devices and application domains. To bridge the gap, various software development frameworks have been proposed, many of which are open-source. However, they target different goals and application domains, and they provide different support for developing IoT aplications. Determining which framework to use for designing and implementing a given IoT application may be challenging.

This paper presents the results of a preliminary study of IoT development frameworks. In particular, we studied a sample of four open-source IoT frameworks (i.e. Eclipse Vorto [18], ThingML [4], Node-RED [11] and OpenHab [12]). We used these frameworks to implement a selection of three IoT applications spanning different domains. We also compared the four frameworks against a set of criteria corresponding to a minimal set of requirements of IoT applications.

The paper is structured as follows. Section 2 discusses related work. We describe the design of the study in Section 3. Section 4 summarizes the work achieved to implement each application using each framework while Section 5 presents the results of our analysis of the frameworks. We conclude in Section 6.

2 RELATED WORK

There are several studies that survey IoT platforms and frameworks. [13] proposes a general methodology to classify various functional and non-functional requirements of IoT applications and devices, as well as a taxonomy to classify IoT frameworks. Whereas other works (e.g. [8], [14] and [19]) focus on the deployment aspect of IoT applications (e.g. cloud and service based). [8] provides a set of criteria (i.e. functional, non-functional and business-related) for selecting a cloud platform, and succinctly compares IoT platforms for a subset of the criteria. Similarly, [14] surveys a selection of the main cloud-based IoT platforms, and proposes a methodology to assist in selecting the most suitable platform for a given use case. [19] surveys a set of 9 generic frameworks, and outlines their capabilities along four different dimensions: data storage in the cloud, the availability of a REST service, live logging and security.

Other works (e.g. [16], [10] and [5]) present more general studies. [16] presents a holistic survey of different IoT platforms, communication technologies and applications, and presents an outline of the main architectural components of the IoT landscape (i.e., edge, fog, cloud layers). Similarly, [10] provides a definition of the main components of an IoT platform, surveys popular communication protocols used in the IoT landscape, discusses factors that should be considered for selecting an IoT platform, and provides a summary comparison of 20 platforms against few high-level criteria. [5] provides a methodology to analyse the suitability of a large amount (63) of IoT frameworks, for several criteria along the different dimensions of the IoT application development lifecycle (e.g., design, modelling, implementation, test, deployment)

In contrast to the works listed above, our study focus on the software development frameworks; we study tools that help implement and deploy software on devices. Moreover, our work provides a thorough and practical study; while we chose to evaluate a limited set of open-source IoT frameworks, we implemented three applications that model representative use cases from IoT applications using each of the studied frameworks, and we evaluate the frameworks over these applications and a curated set of criteria.

3 DESIGN OF THE STUDY

The goal of this preliminary study is to investigate the support provided by open source frameworks to develop IoT applications. In particular, our study aims to answer the following research question:

- To what extent do open source frameworks support a minimum set of IoT application requirements?

To answer this question, we selected some IoT development frameworks and used them to implement and deploy examples of IoT applications. For the deployment of these applications, we targeted the Raspberry Pi as it is one of the most popular and low-cost platforms. To cover a wide spectrum of IoT applications, we studied different IoT applications from the literature (e.g. [17]) and their classifications (e.g. [20], [10]). At a very high-level, we identified three categories of applications and we implemented one application of each category using each of the frameworks. To systematically evaluate the studied frameworks, we identified a set of criteria that is based on a minimum set of features needed to implement IoT applications.

3.1 Studied IoT development frameworks

For this preliminary study, we selected a sample of four open-source IoT development frameworks, namely Eclipse Vorto [18], ThingML [4], OpenHab [12] and Node-RED [11]. Except for Node-RED, these frameworks are all based on the Eclipse development environment. Initially, we focused our study on frameworks that were part of the Eclipse IoT project, i.e. Eclipse Vorto and OpenHab. To diversify the studied frameworks, we also studied ThingML which is an Eclipse plugin and Node-RED which is a very popular programming tool that makes it easy to wire hardware devices. Each of these frameworks is described in more details in Section 4. We used version v0.10.0 M6 of Vorto, version v1.0.0.4 of ThingML, version v0.19.5 of Node-RED and version 2.4 of OpenHab.

3.2 Identification of a sample of IoT applications

To experiment on a representative sample of IoT applications, we rely on the basic classification of existing IoT applications that was introduced in [20]. Thus, based on their complexity, we distinguish between three categories of IoT applications:

- Tracking devices: These applications aim at localizing devices through some identification and tracking technology (e.g. RFID tags). The goal is to efficiently manage the tracked devices which can be equipment parts, assets or products. A typical example of such applications is the inventory management of a store's products.
- Monitoring the real-time states of devices: Such applications enable monitoring the status of some device. Typical examples of such applications include weather monitoring systems and traffic monitoring systems. These applications rely on various sensors (e.g. thermometer, camera, GPS) to capture the status of the monitored devices.
- Controlling states of devices: An application of this category captures the state of some device and makes decisions to act on it (or on other devices) accordingly. Thus, these applications use various sensors to monitor the state of devices but they also use actuators to turn the decisions made into physical actions (e.g. turn on a heater, move a piston, etc.). A simple example of such applications is a smart heating system.

We choose to implement an application of each category using each of the studied frameworks. Table 1 lists the examples of applications we choose for each category.

Category	Selected example
Tracking devices	inventory management system
Monitoring the real-time states of devices	Weather monitoring system
Controlling states of devices	Smart heating system

Table 1: Categories and corresponding selected IoT applications

3.3 Deployment platform: Raspberry Pi

For our study, we used the Raspberry Pi platform [7] which is a low-cost device with computing capabilities. The Raspberry Pi has been commonly used to learn programming and design and build various applications. Mainly, Raspberry Pi has been used in home automation, automated traffic signalling, surveillance, smart agriculture, etc. The Raspberry Pi can be seen as a computer that runs an operating system and provides a set of pins to interface with other devices. These pins are called GPIO (general purpose input/output) pins. The GPIO pins are the means to control and interact with IoT devices.

Several Raspberry Pi models were released by the Raspberry Pi foundation [6]. We used the Raspberry Pi 3 model B+ for our experiments. The model B+ is a 64-bit quad-core processor running at 1.4GHz. It also has 1GB LPDDR2 SDRAM, 40 GPIO pins and 4 USB 2.0 ports. To implement the three examples of applications in our sample, we used additional devices. For the inventory management system, we used RFID tags and an RFID tag reader to identify and track objects. The tag reader was plugged into the Raspberry Pi via a USB port (see Figure 1.a). For the weather monitoring system, we used a basic temperature and humidity sensor. Specifically we used the DHT11 sensor which is a popular and very cheap sensor that can be easily connected to the Raspberry Pi (see Figure 1.b). The sampling rate for the DHT11 is 1Hz, i.e. the sensor performs a reading every second. For the smart heating system (Figure 1.c), we used the DHT11 sensor to monitor the temperature, and a LED to simulate the heater; i.e., the led is switched on or off depending on the difference between the ambient temperature (detected by the sensor) and the targeted one.

3.4 Criteria of analysis

We used a set of criteria to analyze the support provided by the studied IoT development frameworks to build IoT applications. To define these criteria, we proceeded in two steps. First, we surveyed a number of works discussing IoT applications (e.g. [10], [20]) and we identified an initial list of features that must be provided by the frameworks to ease the implementation of IoT applications. Second, we studied the ISO IoT-RA standard [2] which provides a standardized IoT reference architecture and discusses IoT industry best practices. We complemented our initial list of features through the analysis of the ISO IoT-RA. We focused on features related to the software development activities. The identified features include: 1) the support provided to specify the application's functions (i.e. specification of the connected devices), 2) the support provided for non functional requirements (e.g. security, fault tolerance, heterogeneity management), 3) the support for analysis of IoT applications (e.g., visualization, simulation), and 4) the support for complete code production.

Thus, the final set of criteria that we used in our study includes:

- Specification of the interface of the device: The ability of the framework to support the developer in defining the inputs /outputs of a device.
- Specification of the behavior of the device: The ability of the framework to support the developer in modeling the behavior of a device.
- Specification of the properties of the device: The ability of the framework to support the developer in defining the hardware characteristics of a device (e.g. memory).
- Security: The support provided by framework to the developer to build secure applications. In particular, we focus on common security concerns, namely authentication, data encryption, and data integrity.
- Heterogeneity management: The ability to build systems of heterogeneous connected devices.
- Fault Tolerance: This refers to the ability of the framework to: 1) support the specifications of errors, and 2) enable their detection during the execution of the application.
- Integration of COTS components: The ability of the framework to support the integration of components available on public repositories (e.g. Eclipse Marketplace).
- Discoverability: The ability to ease the addition of new devices or services to an exiting IoT application; i.e. to support the identification and description of a new device in a way that it can be discovered by existing ones.
- Complete code production: The ability to support the production of a complete code that is ready to run.
- Data storage: The ability of the framework to support data storage (e.g. captured values, logs of errors).
- Visualization: The ability of the framework to support the visualization of the connected devices, i.e. displaying the states of the devices or the data exchanged between devices.
- Simulation: The ability of the framework to simulate the operation of the IoT application in order to evaluate the application against its requirements.

Since we are using the Raspberry Pi as a target platform, we defined two additional criteria related to the deployment and execution of the source code generated by the framework on the Pi:

- Easy deployment on the Raspberry Pi: The ability to easily deploy the code developed using a framework on the Pi.
- Execution on the Raspberry Pi: The ability to execute the code on the Pi.

4 STUDY EXECUTION

In this section we briefly introduce each of the studied frameworks and we summarize the work achieved to implement each of the IoT applications using each of the frameworks.

4.1 Eclipse Vorto

4.1.1 Overview. Eclipse Vorto [18] is an open source project that is part of the Eclipse IoT project. The goal of Vorto is to provide support for developers to: 1) describe device capabilities, 2) share devices descriptions through a common repository, and 3) use these descriptions to integrate them with various IoT platforms.

To describe devices, Vorto relies on its own DSL (domain-specific language) which was developed based on other programming languages like Java. Vorto DSL specifies the capabilities and functionality of a device as an Information Model. An Information Model is composed of a set of abstract and technology-agnostic Function Blocks. Information models can be shared with other the Vorto community through the Vorto Repository. Vorto relies on some code generators to generate the application code from the Information model. The developer may choose one of the available code generators (e.g. Bosch IoT suite, Eclipse Ditto) depending on the target platform. The Vorto version we used for this study (i.e, Vorto v0.10.0 M6) didn't support complete code generation.

4.1.2 Implementation of our sample of IoT applications. We implemented the three examples of IoT applications using Vorto. Their implementation was similar; i.e. we used Vorto's DSL to describe the information model and function blocks for each of the devices

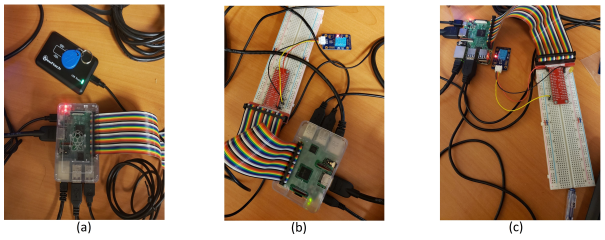
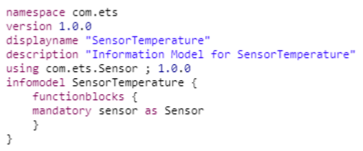
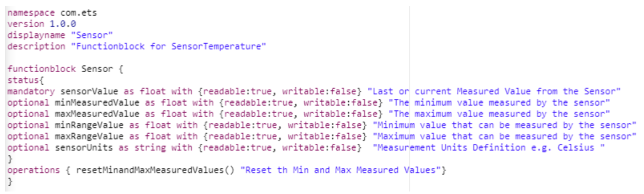


Figure 1: Experimental setup



```
namespace com.ets
version 1.0.0
displayname "SensorTemperature"
description "Information Model for SensorTemperature"
using com.ets.Sensor ; 1.0.0
infomodel SensorTemperature {
    functionblocks {
    mandatory sensor as Sensor
    }
}
```

Figure 2: Information model of DHT11 sensor



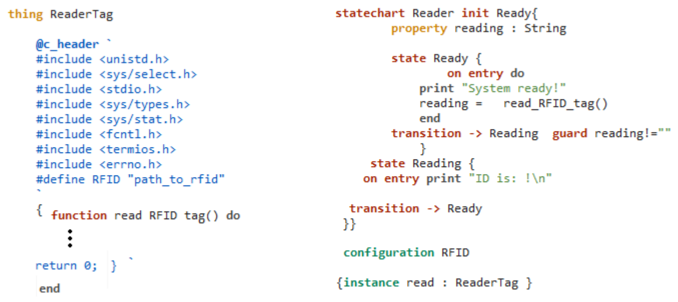
```
namespace com.ets
version 1.0.0
displayname "Sensor"
description "Functionblock for SensorTemperature"

functionblock Sensor {
status{
mandatory sensorValue as float with {readable:true, writable:false} "Last or current Measured Value from the Sensor"
optional minMeasuredValue as float with {readable:true, writable:false} "The minimum value measured by the sensor"
optional maxMeasuredValue as float with {readable:true, writable:false} "The maximum value measured by the sensor"
optional minRangeValue as float with {readable:true, writable:false} "Minimum value that can be measured by the sensor"
optional maxRangeValue as float with {readable:true, writable:false} "Maximum value that can be measured by the sensor"
optional sensorUnits as string with {readable:true, writable:false}  "Measurement Units Definition e.g. Celsius "
}
operations { resetMinandMaxMeasuredValues() "Reset th Min and Max Measured Values"}
}
```

Figure 3: FunctionBlock of DHT11 sensor

of each application, and we generated the Java code of each application using one of the generators provided by Vorto. For instance, to implement the weather monitoring system, we described the DHT11 sensor using an information model displayed by Figure 2 and its ability to sense temperature as a function block displayed by Figure 3. The function block defines a number of properties of the sensor as part of its status; e.g. the property sensorValue describing the current value captured by the sensor, the properties minMeasuredValue and maxMeasuredValue corresponding, respectively, to the lowest and the highest captured values, the properties minRangeValue and maxRangeValue defining the temperature Range of the DHT11, and the property sensorUnit describing the temperature unit. Since the generated code is incomplete (i.e. only a skeleton is generated), it needed to be completed manually before deployment and execution on the Pi.

4.2 ThingML

4.2.1 Overview. ThingML [4] is an Eclipse plug-in that targets the design and implementation of distributed reactive systems. ThingML is mainly intended for devices with limited resources. ThingML relies on a DSL that combines various modeling constructs including statecharts, components and a platform-independent action language. The DSL enables to specify the architecture of the IoT application in terms of communicating devices, and to model the complete behavior of the devices using statecharts and the action language. ThingML also supports the integration of legacy components as black boxes.



```
thing ReaderTag

    @c_header `
    #include <unistd.h>
    #include <sys/select.h>
    #include <stdio.h>
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    #include <termios.h>
    #include <errno.h>
    #define RFID "path_to_rfid"
    `
    { function read RFID tag() do
        ⋮
    return 0;  } `
     end
```

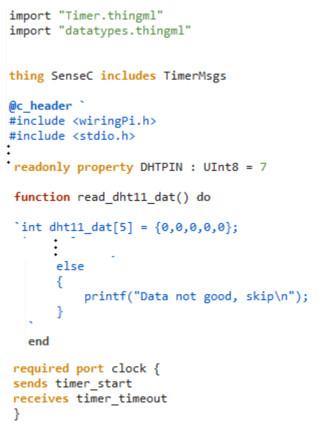
```
statechart Reader init Ready{
        property reading : String

        state Ready {
                on entry do
            print "System ready!"
            reading =   read_RFID_tag()
            end
        transition -> Reading  guard reading!=""
            }
      state Reading {
     on entry print "ID is: !\n"

   transition -> Ready
  }}

  configuration RFID

 {instance read : ReaderTag }
```

Figure 4: Specification of the tag reader using ThingML

The DSL of ThingML relies on two key structures [9]: things and configurations. The thing is the implementation unit that describes devices. The specification of a thing may include properties, functions, messages, ports and state machines. Both properties and functions are local to a thing. Ports can send and receive messages, they represent the public interface of a thing. Configurations describe the connection between things through their ports. ThingML provides a code generation framework including a set of compilers targeting different languages (e.g. Java, C/C++). The generated code is complete and ready to be compiled and executed on the target platform.

4.2.2 Implementation of the inventory management system. Using ThingML's DSL, we created a Thing named ReaderTag (see left side of Figure 4). We also had to implement a C function that listens to the USB port to which the tag reader is connected, and grabs data when a tag is detected by the reader. Data returned by this function is processed and displayed on the Raspberry Pi command line. The behavior of the tag reader is specified as a state machine using ThingML's DSL (see right side of Figure 4). Initially, the ReaderTag thing is in the "Ready" state. It transitions to the "Reading" state when a Tag ID is read.

4.2.3 Implementation of the weather monitoring system. For the application example of the second category, we created a thing "SenseC" representing the DHT11 sensor (Figure 5). The DHT11 thing uses a timer to control the frequency of measuring the temperature and humidity. We also had to develop a C function that reads the measured temperature and humidity values from the GPIO pins of the Raspberry Pi. The behavior of the sensor is specified as a state machine "PiSenceC" whose diagram is shown in Figure 6. The sensor thing starts the timer and it initially enters the "Sensing" state. When the timer's time is out, the sensor measures



```
import "Timer.thingml"
import "datatypes.thingml"


thing SenseC includes TimerMsgs

@c_header `
#include <wiringPi.h>
#include <stdio.h>
:
 readonly property DHTPIN : UInt8 = 7

 function read_dht11_dat() do

 `int dht11_dat[5] = {0,0,0,0,0};
      :
      else
      {
          printf("Data not good, skip\n");
      }
   `
   end

 required port clock {
 sends timer_start
 receives timer_timeout
 }
```

Figure 5: Specification of DHT11 sensor using ThingML

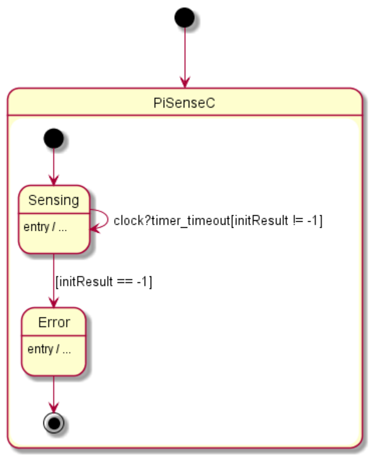


Figure 6: State machine of the DHT11

the temperature and humidity, restarts the timer and transitions back to the "Sensing" state. If the DHT11 sensor could not carry out the measurement, the thing transits to an "error" state. Using the ThingML specification of the DHT11 behavior, we generated the C executable code of the application that we deployed and executed on the Raspberry platform. The results of the sensing were displayed on the Raspberry command line.

4.2.4 Implementation of the smart heater system. For the application example of the third category, we used the same DHT11 thing specification that we defined for the previous example. The only difference is that the measured temperature is no more displayed on the command line. Instead, the temperature value is sent to another thing that represents the heater device. Thus, we created a "Heater" thing and we described its behavior as a state machine shown in Figure 7. The heater is initially in the "Stop" state. The measured temperature received from the DHT11 is compared to a threshold temperature. If the current temperature is less than the threshold temperature, the heater is turned on and it transitions to the "heating" state through the "activate" transition. Otherwise,

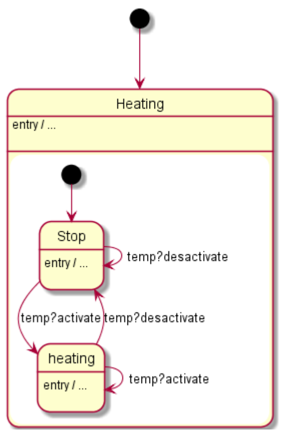


Figure 7: State machine of the Heater

it remains in the "Stop" state. When the heater is in the "heating" state and the temperature is greater than the threshold, the heater is turned off and it transitions to the "Stop" state through the "deactivate" transition. Using the specifications of the DHT11 and the heater, we generated the code of our application and we deployed and executed the code on the Raspberry Pi.

4.3 Node-RED

4.3.1 Overview. Node-RED [11] is a Flow-based programming tool. It allows the interconnection of devices, APIs, cloud environments and online services. Node-RED’s flow editor is accessible through a web browser interface and consists of 3 parts:

- The palette: it provides a set of nodes that are the basic building blocks for creating flows.
- The work environment: the work space where users can create their flows.
- The sidebar: which may be used to display various information; e.g. information about nodes, output and debugging information.

In order to design the flow diagram, one must drag and drop the nodes from the palette to the work environment, then wire them to create the data flow. The nodes are developed in JavaScript and saved in JSON format. By default, there is a set of predefined nodes; additional nodes can be easily downloaded. The palette nodes are also extensible, developers can create and save their own nodes. In fact, Node-RED has a very active community with a plethora of nodes having been developed for different devices and purposes. A notable node is the "Function" node, which allows for programming a given behavior directly using JavaScript code through the GUI.

4.3.2 Implementation of the inventory management system. We followed the steps described in [1] to implement the inventory management example using Node-RED. To be able to read from the tag reader, we needed first to register it with its vendor and device IDs. To do so, we used the `getHIDdevice` node. We then created the flow for parsing the readings of the tag reader (Figure 8). To do so, we used the `HIDdevice` node which was fed the values of the vendor and device IDs of the tag reader. The `HIDdevice` node reads the information received from the RFID reader through the Raspberry

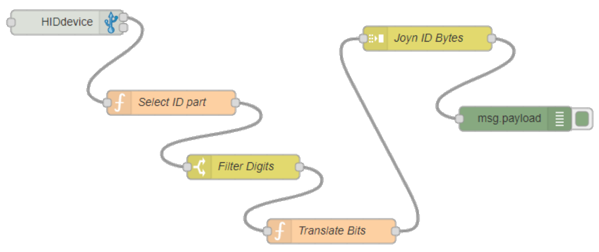


Figure 8: The flow for parsing the readings of the tag reader

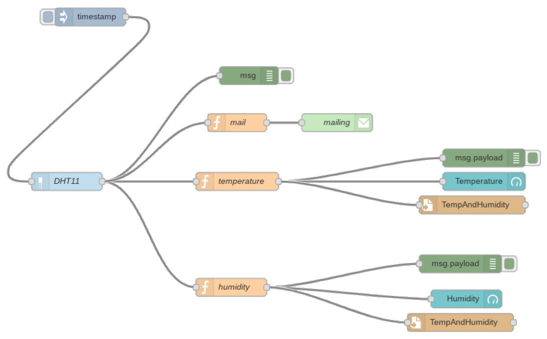


Figure 9: The flow of the weather monitoring system

USB port. The byte arrays produced by the `HIDdevice` node need to be processed in order to retrieve the tag IDs. This processing includes selecting only the part of the byte array containing the ID, eliminating empty lines and line breaks, translating bytes to real digits and merging the resulting digits into one ID.

4.3.3 Implementation of the weather monitoring system. Node-RED includes ready-to-use nodes for the Raspberry Pi to communicate with sensors and peripherals. We use the DHT11 node to implement our weather monitoring system (figure 9). The DHT11 node performs a single reading of the *temperature* and *humidity* values. The captured values are then saved to a file. We graphically display each value on a gauge using existing graphic nodes.

4.3.4 Implementation of the smart heater system. In this application, we compare the temperature measured using the DHT11 node against a predefined threshold. To do so, we used a function node to convert the temperature data obtained from the DHT11 node from a "string" type to a "real" type. We also implemented a decision function node (in JavaScript), which compares the two temperature values, and enables/disables a LED if the heater should be activated/deactivated.

4.4 OpenHab

4.4.1 Overview. OpenHab (Open Home Automation Bus) [12] is an open source software framework for smart home. The framework runs on several platforms (i.e. Linux, Windows and MacOS) and several hardware including the Raspberry Pis. A key feature of OpenHab is its ability to integrate a multitude of smart devices into a single application. OpenHab also explicitly distinguishes between two views of an IoT system: the physical view which represents the devices and their connections, and the logical view which corresponds to the information representing devices and connections in the application. The framework also features a lightweight rule engine that executes the application automation processes.

To use OpenHab for creating a smart home application, the developer needs to:

- Define things and corresponding channels and items. A thing is a software representation of a device or a service. It exposes its functions through channels. Items represent functionality (user interfaces and automation logic) that is used by the application. Things are part of the physical view while items are part of the logical view (also called the virtual layer). Channels are linked to items, and these links are what establishes the relation between the virtual layer (i.e. items) and the physical layer (i.e. things). A link connects one channel to one item, but a channel may be linked to several items and vice versa. A thing reacts to events sent to items linked to its Channels, and it sends events to these items.
- Search the OpenHab addons for each smart device that is part of the application and install the bindings for each device; a binding is an OpenHAB component that enables the interaction with a device. In particular, the bindings establish the connections between devices and things.
- Define the sitemap of the application; a sitemap is the OpenHAB generated user interface that presents information and allows for interactions.
- Define rules; a rule is used for automating processes. A rule defines a triggering condition (e.g. an item whose status has changed) and the logic (i.e. a script) that should be executed when the condition is met.

4.4.2 Implementation of the inventory management system. We were not able to implement this application. This is due to the protocol used by the RFID reader (USB communication) that is not supported by OpenHab.

4.4.3 Implementation of the weather monitoring system. To implement this application, we tried two different methods: one that follows the process described above and another that relies on the MQTT protocol to retreive the sensors readings. We only describe the first method in this paper. In this case, we defined a thing for the DHT11 sensor in a file named DHT11.things and the temperature and humidity as items. We implemented a Python script that reads the temperature and humidity values from the DHT11 thing. We also added the GPIO bindings to let OpenHab read the values captured by DHT11 through the pins of the Raspberry. Finally, we created a file sitemap for the UI of our application.

4.4.4 Implementation of the smart heater system. To implement this application, we first activate the GPIO bindings for OpenHab to enable the reading and writing through the Raspberry Pi pins. We specified the pins to which the DHT11 and the LED were plugged using a python script. We defined two things: one for the DHT11 sensor and one for the LED. We also defined the temperature and the LED state as items. We implemented a set of rules corresponding

	1st category	2nd category	3rd category
Eclipse Vorto	●	●	●
ThingML	≋	∿	∿
Node-RED	∿	→	→
OpenHab	●	→	∿

→: Easy to implement ∿: Somewhat difficult to implement
≋: Very difficult to implement ●: Not feasible

Table 2: Support provided by the frameworks for implementing our sample of IoT applications

to the heating process; e.g. if the current temperature is below the target temperature, the LED turns ON, otherwise it goes to OFF. Finally, we created a file sitemap for the UI of our application.

5 RESULTS AND DISCUSSION

In this section, we first summarize our findings regarding the ability of the studied frameworks to support the implementation of each category of IoT application. We then present and discuss the results of our study in relation to the research question stated in Section 3.

5.1 Support for the selected IoT applications

Table 2 summarizes the results for each studied framework in terms of ease/difficulty of implementing each of the three IoT applications in our sample. Eclipse Vorto does not enable to build a complete application; i.e. does not support the generation of a complete code. This is due to the fact that Vorto focuses on integration of heterogeneous devices. Thus, it provides a DSL to specify devices at a very high-level. Depending on the target IoT platform, a skeleton of the source code is generated from the specification of the device enabling its integration into the platform. For instance, the code generator Eclipse Ditto generates a skeleton of code to be integrated into the Eclipse Ditto platform.

Using ThingML, we were able to implement all the three IoT applications in our sample. However, the three applications required some amount of effort for implementation. The first category was very difficult to implement because to the operating mode of the RFID reader; i.e. we needed to listen to the USB port and display a tag ID when it is read by the RFID tag reader. However, ThingML does not support this type of connection. So we had to develop a C function that reads the RFID tags ID; i.e. we took advantage of the fact that ThingML enables to integrate platform-specific code. The third category was also challenging as we had a problem exchanging information between the two things (i.e. DHT11 and LED). For all of the three categories, debugging was an issue as the ThingML IDE does not support platform-specific code debugging.

Node-RED is the framework that required the least efforts and time for learning and implementing. Both the second and third category of applications were easy to implement because of the availability of nodes corresponding to the functions of the devices included in these applications. However, the first category (i.e. the inventory management application) was a bit challenging because of some package dependencies; i.e. the implementation of this application required downloading additional packages that caused compatibility issues with our Node-RED version.

Using OpenHab, we were able to implement the second and the third category examples (i.e. the wather monitoring and the heating systems) but not the first category. This was to be expected since Openhab is a Home automation framework and the second and third examples are related to the smart home domain. Nevertheless, the third category was a bit more challenging to implement because of the number of files we had to put together to realize the application.

5.2 Analysis according to IoT requirements

Table 3 summarizes the results of our analysis of the studied frameworks according to the criteria defined in Section 3. Looking at the frameworks, none of them meets all our criteria. In particular, Eclipe Vorto is the framework that provides the least support to developers in building IoT applications. This is a bit normal since Vorto can be seen as an interface description language focusing on heterogeneity management.

Looking at the criteria, all the frameworks support heterogeneity management. This is due to the fact that IoT applications are typically composed of various devices and components and interoperability between these devices is required. Conversely, none of the frameworks enables the specification of the properties of devices (i.e. hardware characteristics). This is an issue since many IoT devices may have limited hardware capabilities (i.e. memory and process) that must be taken into consideration during deployment but also during the execution of IoT applications. OpenHab and ThingML are the frameworks that support a complete specification of the devices in terms of input/output and behavior. Although it is widely used, Node-RED does not support the specification of the interface of devices neither their behavior; i.e. not in an explicit way. In fact, Node-RED focuses on describing the data flow.

Regarding security, most of the frameworks provide some support. In Eclipse Vorto, the user has to login to the Vorto repository to gain access to the specification of devices. In Node-RED and OpenHab, there are APIs or addons that can be downloaded and installed to ensure controlled access to the things and the application being developed. Node-RED is the framework that most eases the implementation of security. For instance, Node-RED provides explicit nodes to support data-encryption. Regarding fault tolerance, Node-RED is the only framework that does not support the explicit specification of errors during the design of the application. On the other hand, Node-RED and OpenHab are the only frameworks that support detecting errors during execution; both frameworks provide means to catch errors and create logs. In addition, Node-RED and OpenHab support simulation of the application, data visualization and integrating COTS components, which makes the two frameworks very attractive to developers. As for data storage, both ThingML, Node-RED and OpenHab makes it possible to persist data; ThingML requires to manually implement a function to do so.

ThingML, Node-RED and OpenHab enable to produce a complete code. However, ThingML requires the manual implementation of several platform-specific functions and the deployment of the resulting code on the Raspberry Pi requires some efforts, whereas

Criteria	Eclipse Vorto	ThingML	Node-RED	OpenHab
Specification of the interface of the device	✓	✓	×	✓
Specification of the behavior of the device	×	✓	×	✓
Specification of the properties of the device	×	×	×	×
Security-Authentication	✓	×	✓	✓
Security-Data Encryption	×	✓	✓	✓
Security-Data Integrity	×	✓	✓	×
Heterogeneity management	✓	✓	✓	✓
Fault tolerance-Specification of errors	✓	✓	×	✓
Fault tolerance-Detection of errors during execution	×	×	✓	✓
Integration of COTS components	×	×	✓	✓
Discoverability	×	×	×	✓
Complete code production	×	✓	✓	✓
Data storage	×	✓	✓	✓
Visualization	×	×	✓	✓
Simulation	×	×	✓	✓
Easy deployment on the Raspberry Pi	×	×	✓	✓
Execution on the Raspberry Pi	×	✓	✓	✓

Table 3: Evaluation of the studied frameworks according to our criteria

Node-RED and OpenHab both run on the Raspberry Pi which makes it easy to run the resulting code on the Pi.

5.3 Threats to validity

Some limitations of our study are due to the number of applications we implemented using the studied frameworks and the number of devices in these applications. To mitigate these threats, we relied on existing classifications of IoT applications and we implemented one application for each IoT class. In the future, we plan to extend the study and implement additional applications with more devices. Also, the comparison between the studied frameworks may seem unfair as each framework has a specific goal or targets a specific domain. Future work includes the study of how these frameworks may complement each other.

6 CONCLUSION

This paper presents a study on open-source IoT development frameworks. We selected a sample of four frameworks (Eclispe Vorto, ThingML, Node-RED and OpenHab) that we used to implement a sample of three IoT applications covering the different categories of applications discussed in the literature. We also analyzed the studied frameworks using a minimal set of requirements that must be met by IoT applications. The results of our study show that Node-RED and OpenHab are the frameworks that offer the best support for developing our sample of applications. A combination of these two frameworks may offer a more extensive support for IoT requirements. In fact, Node-RED is now providing a set of nodes to ease the integration of OpenHab. In the short-term we plan to study additional frameworks and extend our sample of IoT applications.

REFERENCES

[1] Raphael Binks. 2018. *IOT Tutorial: Read RFID-tags with an USB RFID reader, Raspberry Pi and Node-RED from scratch.* https://medium.com/coinmonks/iot-tutorial-read-tags-from-a-usb-rfid-reader-with-raspberry-pi-and-node-red-from-scratch-4554836be127

[2] Technical comitee : ISO/IEC JTC 1/SC 41 Internet of Things and related technologies. 2018-08. ISO/IEC 30141:2018 Internet of Things Reference Architecture.

[3] Ericsson. 2018. *Internet of Things forecast.*

[4] B. Morin F. Fleurey and O. community. 2016. *Thingml source code repository.* https://github.com/sintef-9012/thingml

[5] Mahdi Fahmideh and Didar Zowghi. 2020. An exploration of IoT platform development. *Information Systems* 87 (2020).

[6] Raspberry Pi foundation. 2020. *Buy a Raspberry Pi.* https://www.raspberrypi.org/products/

[7] Raspberry Pi foundation. 2020. *Teach, Learn, and Make with Raspberry Pi.* https://www.raspberrypi.org/

[8] Pankaj Ganguly. 2016. Selecting the right IoT cloud platform. In *2016 International Conference on Internet of Things and Applications (IOTA).* IEEE, 316–320.

[9] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 2016. ThingML: a language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems.* 125–135.

[10] Hamdan Hejazi, Husam Rajab, Tibor Cinkler, and László Lengyel. 2018. Survey of platforms for massive IoT. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT).* IEEE, 1–8.

[11] Node-red. 2013. *Node-RED: Low-code programming for event-driven applications.* https://nodered.org/

[12] OpenHAB. 2014. *openHAB - empowering the smart home.* http://www.openhab.org/

[13] Leila Fatmasari Rahman, Tanir Ozcelebi, and Johan J Lukkien. 2016. Choosing your IoT programming framework: Architectural aspects. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud).* IEEE, 293–300.

[14] Amirfardad Salami and Alireza Yari. 2018. A framework for comparing quantitative and qualitative criteria of IoT platforms. In *2018 4th International Conference on Web Research (ICWR).* IEEE, 34–39.

[15] Tim Stack. 2019. *Internet of Things (IoT) Data Continues to Explode Exponentially. Who Is Using That Data and How?* 2018.

[16] Shahab Tayeb, Shahram Latifi, and Yoohwan Kim. 2017. A survey on IoT communication and computation frameworks: An industrial perspective. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference.* 1–6.

[17] Itorobong S Udoh and Gerald Kotonya. 2018. Developing IoT applications: challenges and frameworks. *IET Cyber-Physical Systems: Theory & Applications* 3, 2 (2018), 65–72.

[18] Eclipse Vorto. 2016. *Vorto introduction.* https://www.eclipse.org/vorto/documentation/overview/introduction.html

[19] Indunil Withana and Cassim Farook. 2019. IoT Generic Frameworks: What Needs to Improve. In *2019 7th International Conference on Smart Computing & Communications (ICSCC).* IEEE, 1–5.

[20] Ying Zhang. 2011. Technology framework of the Internet of Things and its application. In *2011 International Conference on Electrical and Control Engineering.* IEEE, 4109–4112.