Tiny Thing Blocks: Integrating Everyday Objects into IoT Context

Lucas Amorim, Marcio Alencar and Raimundo Barreto
Institute of Computing
Federal University of Amazonas
Manaus, Brazil
{lsda, macalencar, rbarreto}@icomp.ufam.edu.br

***Resumo*—The evolution of smart things technologies caused the growth in the popularity of concepts such as smart homes and industry 4.0. The Internet of Things (IoT) is the paradigm that encompasses and give a base for these topics. The development of devices that are used in this paradigm requires knowledge of subjects such as programming, embedded cyber-physical systems, web protocols, networking and others. This paper proposes a method to make it easier for people who do not have this knowledge to create smart IoT devices. To achieve this goal, we decide to create a visual language based on blocks that automatically generate code to Internet of Things devices. This language gives support to design the behavior of devices, which is represented by a model of a finite state machine. This model is generated using a graph generator called Graphviz. We created a compiler for this language using the compiler generator Coco/r. The compiler translates the block code into the C language which is one of the programming language recognised by the Arduino IDE. We advocate that this process is more intuitive than the normal development process after conducting tests with users. We did an experiment with CS students and the result is that the proposed method is promising.**

***Index Terms*—IoT, VLP, Internet of Things, Visual programming language**

I. Introduction

Internet of Things (IoT) is an emerging paradigm focused on problem solving through collaboration between Internet standards and compute-capable objects. Because of this, it is possible to create a smart device-connected environment that both work and make decisions without human intervention. The adoption of smart devices is very useful in several kinds of applications. In industry, for instance, they can reduce the labor cost and improve the machine's supervision to prevent failures or defects in its operation. Another example of good use to the connected devices is the patient monitoring at hospitals or even at home. This paradigm turned the human into a connected thing, optimizing the generation and use of data to improve the humans quality of life [2], [8], [10].

However, there is a large number of protocols and communication standards used in the IoT development. In addition, there are a variety of devices used to build these environments, where each uses different hardware and software to communicate with the Internet. At this point, there are two parts to be observed: the communication hardware and the devices. The communication between devices is indispensable to IoT environments. However, the protocols used to support this communication are suitable for specific contexts. NFC, Wi-Fi, Zigbee, Bluetooth are examples of these protocols, but they do not always talk to each other. Thus, part of the problem is to create a way to make this variety of protocols do not interfere with the data getting and sharing of these device [2].

Another viewpoint to discuss is the development of hardware and software used to create IoT devices since they have a module responsible for communication, and another for collecting data or perform mechanical activities. The development of these devices requires knowledge about embedded systems, web protocols, wireless transmission, electronic circuits, and other subjects. These points could be a setback at creating IoT devices, since people in general usually do not have much knowledge about these subjects [2], [3].

IoT research has the main focus on industry technology and health care systems. However, some research has focused on creating an easy and cheap way to include things in the IoT context. An example is the module Tiny Thing [1], which proposed a specific hardware responsible for brokering the connection of electronic devices and the Web. However, this is just part of the solution since the user still needs to understand how to program the module. To solve this setback, we propose a visual programming language based on blocks that helps the user to understand the behavior of the device and to generate the code to it.

II. Related Work

The IoT code development has usually been done with several tools and using different approaches.

An example is the Node-RED, which is a flow-based tool that belongs to the JS Foundation. The flow-based programming is a method to describe the application's behavior as a network of nodes. Therefore, each node has its data to handle and represent. The flow of these data between nodes depends on the network. The visual representation of data is interesting since it turns the tool more accessible to beginners [9].

Another tool is Visuino, which uses blocks and drag-and-drop paradigm to create a visual programming language to develop software to Arduino boards. This tool represents the hardware actions using blocks, thus it only requires the user to understand what the hardware should do and translate these actions to code [11].

Visualino is another example of a drag-and-drop based visual language to generate software for Arduino boards. Visualino is different from Visuino in the sense that it focus on the direct development of code. Visualino provides blocks to represent the structures of the language used by Arduino [5].

Node-Red is the more recognized and used by developers and teachers on the Internet of Things subject. According to Google trends, it maintained the relevance of the researches since 2018. The following tool is the Visuino, although its relevance of the researchers has been inconstant.

We used Blockly [7] which is a Javascript library used in many online code generation tools such as App Inventor, CODE, Microsoft MakeCode, and others. Using Blockly we created an online tool that generates the model through a visual language based on blocks, and translates this visual language to a C-based language.

Combined to Blockly we used the Graphviz tool, which is an open source graph visualization software that uses the DOT language to create a finite state machine [3], which represents the models behavior. Comparing to the other tools quoted previously, Tiny Thing Blocks focus on receiving a behavior model and generate code from that. This tool works using the same paradigm of Visuino and Visualino, the drag-and-drop model. Nevertheless, the blocks used at this tool represents structures and state transitions.

Another key to develop this tool is the Tiny Thing module, which is a device responsible for doing the communication between an object and the internet. To do this activity the module has two parts in its composition, the first is to describe, control and to monitor the object state. The second is to turn the object accessible through the internet and then control it. The module was built using an ATTiny85 microcontroller and a Wi-Fi board ESP8266 ESP-01 [1].

III. METHODS

The proposed tool consists of a web service that uses a visual programming language to provide a means to generate code for the Tiny Thing module. This service remains on two phases, the first performed by the user and the second by the tool. The first phase is when the user creates the finite state machine to his device using the block language, and with this language, he generates a model based on graphs. These graphs are created using the Graphviz.

The second phase begins when the user compiles the created model. The activities performed here are server-side. The first phase generates a file in the format dot, and this file has the FSM created by the user. This file is compiled using the compiler generated with the Coco/r tool, which is a compiler generator, which takes an attributed grammar of a source language and generates a scanner and a parser for this language. Finally, it returns code in the format “ino"(Arduino) that is portable to the Tiny Thing module.

A. Modeling

The modeling process was designed to be simpler and quicker than code development. To achieve this was necessary to use a computational concept of modeling that can carry the needed data for generating code. The finite state machine is a mathematical model that is present in several things that operate a designed sequence of actions depending on a series of input cases. This concept fits on the behaviors of the IoT device and can represent the devices used with the module.

The next step was to design how to create and visualize this FSM. The chosen tool for this job was the Graphviz. Its support for creating and visualizing graphs provided a way to solve the needings of the modeling phase. Once there is a way to handle the modeling, the next requirement was to create the standards to it.

The standards should represent the main pieces of the device’s operating besides presenting information necessary for the construction of the devices using the module. These standards cover the pieces of information about the type of device (sensor or actuator), the states that the device has and the transition among each one, the pins which it uses to connect with the Tiny Thing. These standards were used to generate a new modeling language based on the DOT format used by Graphviz.

The first point of these standards is about how to categorize the types of devices. There are two types: the actuators and the sensors. Actuators do mechanical actions once they get an electric or automated stimulus. By using this kind of device combined with the Tiny Thing module provide a way to make smart objects such as Smart LEDs and Smart Blinds. Figure 1 shows an example of an actuator servo motor, which can be used with Arduino. Sensors are devices that get an outside physical or chemical incentive and return it to a system responsible for handle this data (see Figure 2). Like the actuators, sensors can use the module to create Smart Objects such as automated monitoring systems of heat or presence. However, sensors are monitoring devices and have different behavior.

Figura 1. Servo motor, an example of an actuator.

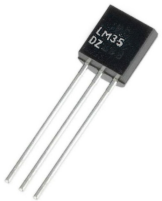


Figura 2. LM35, an example of a sensor.

These concepts of types are the base of modeling the devices. They were used to create the transition's rules for each device. However, the primary step of shaping a new object is to indicate its type and name. Figure 3 shows the syntax used to declare the expected information about the thing. This line will generate the image shown in Figure 4.

```
type[shape="box",label="actuator Led"]
```

Figura 3. Declare type.

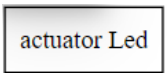


Figura 4. Example of the input "...label="actuator Led"]".

Since the name and type are informed, the next step is to handle the connection between the device and the module. The module has three available pins to connect with things, and there's a value for each one of then (0, 2 and 3). The module needs to know which pin will receive or send data to the objects, so these values need to be declared. Figure 5 represents the way to inform this.

```
pin[shape="box",label="[pinA::2]"]
```

Figura 5. Pin type.

This declaration creates a new block on the model that shows the names and the values of declared pins (Figure 6).

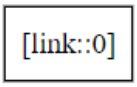


Figura 6. Example of the input "...label="link::0"]".

After these declarations, the model represents the type and the connections acting at this system. These details are useful for the code generation process. Nevertheless, they are nott relevant to design the behavior of a device. The model should have nodes for each state, and these nodes have arrows representing the transition between them. These data are the principal purpose of the process. Figure 7 gives an example of these states and transitions declaration.

Figure 8 shows the output from these lines (Figure 7). Figure 9 represents the transition between two states (on and off).

These examples cover the base of modeling devices. However, sensors need more details for their models. The behavior presented by this type has roots in valued inputs that cause the transitions. There are two ways to get those inputs using sensors one is doing only one reading and return the value. The other way is doing a sequence of readings and return the average of these readings.

The transition's behavior using these functions depends on the range values of each state. The transition between states

```
states[shape="box",label="N=2:[ON,OFF]"]
ON[shape="diamond",label="ON"]
OFF[shape="diamond",label="OFF"]
action_ON[shape="box",label="action_ON -> ON"]
OFF -> ON[label="action_ON -> ON"]
```

Figura 7. State and transition declaration

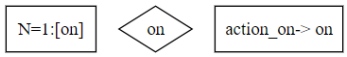


Figura 8. Visualization of State and transition declaration

happens when the input or the average is inside of a different range.

These rules were used to create an Extended Backus-Naur Form (EBNF) that express the grammar for this new language. The EBNF is a notation of metalanguage used to express context-free grammar. Programming languages could use this notation to do formally describe themselves. Finally, using the EBNF notation, we build the syntax of this new language. Since EBNF is defined, it can be used to create a compiler to translate this language to code for the Tiny Thing module.

B. Code generation

Since the new language was defined, the next step in the process is to produce code for the module using the model. To perform this job is necessary to create a translator from the new language to language used by the Tiny Thing module, in this case, a compiler. The translator recognizes the grammar of this new language and translates the structures of this language to a ".ino"file to be used by the module. To create this compiler we used the compiler generator Coco/r. That tool receives the grammar of one language and generates a scanner and a parser for it.

The scanner created by the Coco/r works as a deterministic finite automaton (DFA), this implies that the scanner accepts or rejects strings of symbols (or tokens) of the grammar. Finally, it produces or runs a single computation of the automaton for each input. The EBNF grammar is essential to describe the tokens which the scanner recognizes.

The parser generated with Coco/r recognizes LL1 grammar. It makes a parsing decision based on a multi-symbol

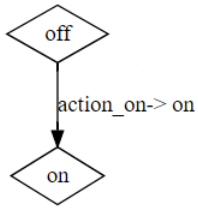


Figura 9. Transition of states

```
HOT -> NORMAL[label="avg.read: ? < 0 & ? <0"]
HOT -> NORMAL[label="read: ? < 0 & ? <0"]
```

Figura 10. Read and avg declaration

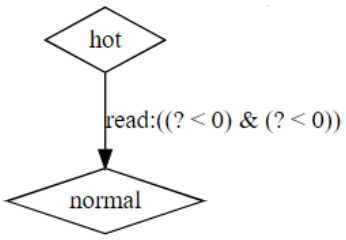


Figura 11. Transition read

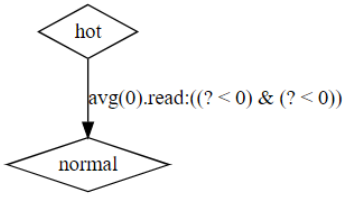


Figura 12. Transition average

lookahead or, semantic information. The EBNF productions with attributes and semantic actions specify the parser. The Coco/R translates the structures into an efficient recursive descent parser.

In the Coco/r, the semantic actions are a piece of code written in the target language, in this case, Java. The parser executes these actions at the point they were specified at the grammar. The semantic actions can contain declarations of variables which will be at the final code. These concepts were necessary to create the EBNF of the new language using the Coco/r.

C. Visual programming language

The result of the previous steps is a new language based on the Graphviz format. Like any other language, this one has a curve of learning and requires of its user knowledge about programming languages. So, we decided to create a more accessible approach to develop using this language. This method is a visual language that represents the structures using blocks. According to S. Chang and M. Erwing, a visual programming language (VLP) is one language that allows the users to develop programs using graphic elements in the place of specifying them textually. To built this language we used the library Blockly, which allowed us to create a visual language based on blocks [4], [6].

The Tiny Thing Blocks is an open-source tool that allows the user to develop code for the Tiny Thing module without programming with a text editor. Each block was designed to represent the syntax of one structure of the new language.

The blocks represent declarations of nodes in the model, but not all of these nodes are one part of the final code. Some of the declarations of a block are used only to create the FSM. These declarations are responsible for representing the transition between states.

Figure 13 and Figure 14 show the declarations of state transitions, which is the base of creating an FSM. However,

Figura 13. example of transition for actuators devices

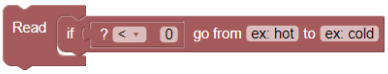


Figura 14. Example of transition for sensors devices

this model is the expected behavior of the device, and the tool does not cover the code for functions. In Section VI, we talk more about changes that would happen at the features of this tool.

There are four categories to categorize the blocks, and each one is a cluster of blocks that are directly linked.

- The first category is the “device"it is the two blocks responsible for starting and finishing the model.

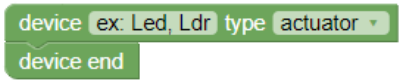


Figura 15. Device blocks

- The second is the "pin’s blocks"category. This one encapsulates the declaration of which pins will the device use.

Figura 16. Pin’s blocks

- The third category is the “state’s blocks". This category encapsulates the blocks which declare the states which the device use, the blocks of transition between states in the case of the device is an actuator and the declaration of the initial state for the model.
- Finally, the last category is the “read’s blocks", which is the group of declarations of functions and transitions that the sensors uses. The user can use these blocks to create logic expressions to build the operation of sensors.

These blocks generate the code to create models and using this model at the compiler it generates the code to modules like Tiny Thing and others that use ESP8266. The following figures show examples of models and the code generated using then.

This example uses blocks to create a model of behavior to an led, which has two states "on"and "off". The system starts

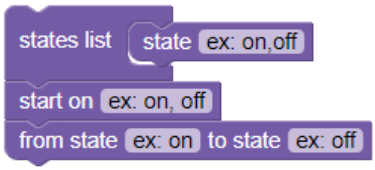


Figura 17. State's blocks

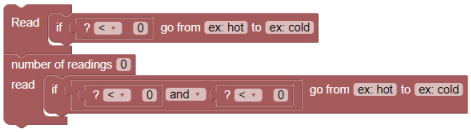


Figura 18. Read's blocks

on the state "off "and can do transitions between the two states, creating a loop.

After creating the model, the user can finally generate code at the format ".ino", this code presents the main pieces of operation to devices based on the modules which use ESP8266. Information such as the states, the connections used, the functions, the type of device and other data are present at the result code of generating phase.

The process to create models for a sensor is a bit different, since the sensor gets an input before changing the state of its device. So, after declaring the states, the user will inform the values of each case of transition between then. To do this, the user needs the set of blocks linked to reading actions, which give the support to logical structures.

IV. RESULTS AND DISCUSSION

The tests evaluated the usability and performance of the tool from people with different levels of knowledge about the development of IoT devices.

The tool was inspected using heuristic evaluations because it is a user-centered system, facilitating the application of heuristics and guidelines for evaluation. The first test was about usability. Thus, a set of heuristics and guidelines were

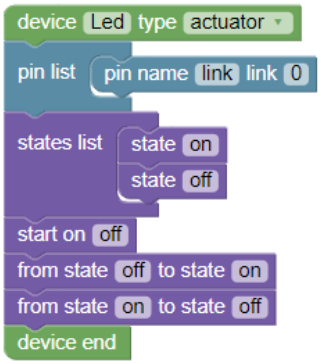


Figura 19. Example of a model for led

```
#define link 0
#include < tinything.h >
TinyThing device;
const char *states[] = {"on", "off"};
void setup(){
        Serial.begin(9600);
        device.setSerial(&Serial);
        device.setCurrentState(1);
        device.setStates(2,states);
        device.setDescription("Led controller");
        device.setType("actuator");
        pinMode(link,OUTPUT);
}
void action_on {
        device.setCurrentState(0);
}
void action_off {
        device.setCurrentState(1);
}
void loop(){
        if(device.hasMessage()){
                char *s = device.checkMessage();
                uint8_t i = device.checkMessage(s);
                switch(i){
                        case: 0 action_on(); break;
                        case: 1 action_off(); break;
                        default: break;
                }
        }
}
```

Figura 20. Example of code generate using the previous model for led

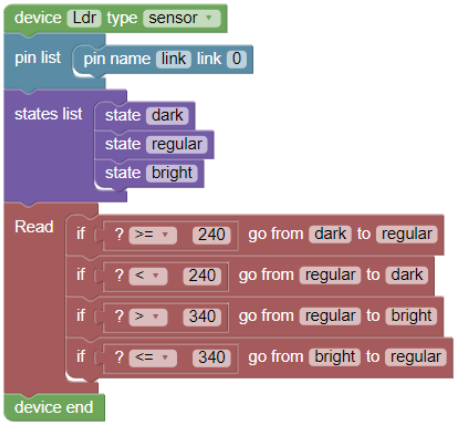
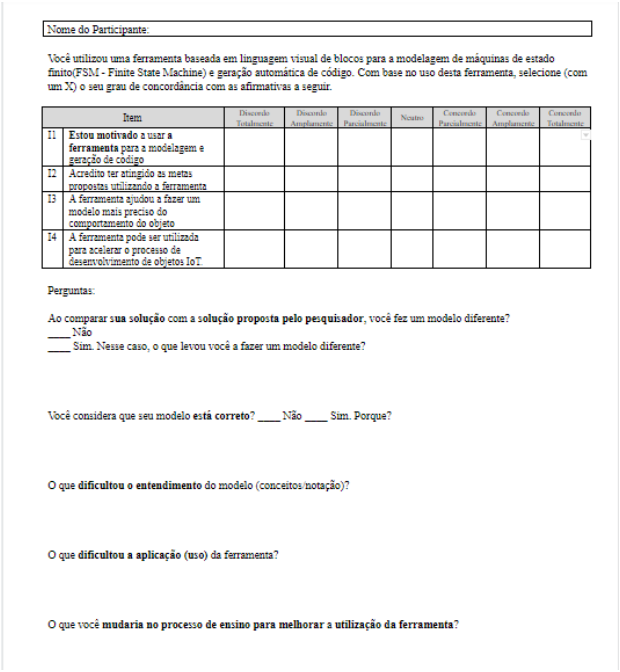


Figura 21. Example of a model for ldr

```
#define link 0
#include < tinything.h >
TinyThing device;
const char *states[] = {"dark", "regular", "bright"};
void setup(){
        Serial.begin(9600);
        device.setSerial(&Serial);
        device.setCurrentState(-1);
        device.setStates(3,states);
        device.setDescription("Ldr checker");
        device.setType("sensor");
        pinMode(link,INPUT);
}
double read(){
        return value;
}
void loop(){
        if(device.hasMessage()){
                double r =read();
                device.setValue(r);
                char *s = device.checkMessage();
                uint8_t i = device.checkMessage(s);
                switch(i){
                        default: break;
                }
        }
}
```

Figura 22. Example of code generate using the previous model for ldr

used to evaluate each part of this feature. This approach has its support in the user-centered feature of the tool. A list of heuristics was used during the operating of the system, this process should capture possible problems in the system.



Nome do Participante:

Você utilizou uma ferramenta baseada em linguagem visual de blocos para a modelagem de máquinas de estado finito(FSM - Finite State Machine) e geração automática de código. Com base no uso desta ferramenta, selecione (com um X) o seu grau de concordância com as afirmativas a seguir.

	Item	Discordo Totalmente	Discordo Amplamente	Discordo Parcialmente	Neutro	Concordo Parcialmente	Concordo Amplamente	Concordo Totalmente
I1	**Estou motivado** a usar a **ferramenta** para a modelagem e geração de código							
I2	Acredito ter atingido as metas propostas utilizando a ferramenta							
I3	A ferramenta ajudou a fazer um modelo mais preciso do comportamento do objeto							
I4	A ferramenta pode ser utilizada para acelerar o processo de desenvolvimento de objetos IoT							

Perguntas:

Ao comparar **sua solução** com a **solução proposta pelo pesquisador**, você fez um modelo diferente?
___ Não
___ Sim. Nesse caso, o que levou você a fazer um modelo diferente?

Você considera que seu modelo **está correto**? ___ Não ___ Sim. Porque?

O que **dificultou o entendimento** do modelo (conceitos/notação)?

O que **dificultou a aplicação (uso)** da ferramenta?

O que você **mudaria no processo de ensino para melhorar a utilização da ferramenta**?

Figura 23. Questionnaire about the usage of the tool

This process resulted in notes of which guidelines were being affected, why they were affecting them, and their severity level. At this activity, fifteen possible problems were defined, and six of then were real defects. Problems have been reported mainly about the various information that should be given to the user regarding interactions with the system: what did this button do? What is going on? What went wrong? None of these questions were treated by the system. After conduct these tests and fix the problems, the next step were to evaluate the performance of users.

To achieve this, we defined a sample group of twenty students from the Federal University of Amazonas, and classify then using their familiarity with the subject. At this test, the user should create a model of each type of device (actuator e sensor), the metrics of this test were the achievement of the proposed goal and the difficult to create the model. Figure 23 shows the questionnaire answered by the users after the tests.

The users who have more knowledge about programming using visual editors had not difficulties using the tool. Users who have less familiarity with this kind of tools have some difficulties learning the way to create models for the first time. However, after having an example of how to use the tool, the doubts were cleared.

V. CONCLUSION

After evaluating the performance of the users, we notice that users who have much familiarity with this kind of tool had no difficulties in understanding the way of creating models using it. Using simple concepts from both FSM and logic added to the tool, the learning of how to build IoT devices is much faster.

In addition, users have noted that the current state of the tool is quite simple and needs to cover more tasks. Although they consider this tool limited, most of then recognized it as useful for creating FSM models for devices and teaching this concept.

On the other hand, the users who have less familiarity with visual programming had a few difficulties in the beginning. However, after seeing examples of models and how the blocks work together, all of them achieved the final goal. By analyzing the behavior of these users during testing, we recognize the importance of giving examples and guides for beginners. However, the experiments are not conclusive since the tests had just a few participants.

For future work, the tool must receive new functionalities such as programming the functions generated using the model, creating accounts and projects, the capacity of saving projects to work at any time,, sample models for users and improve the usability of the tool.

VI. ACKNOWLEDGEMENTS

This research, according for in Article 48 of Decree nº 6.008/2006, was funded by Samsung Electronics of Amazonia Ltda, under the terms of Federal Law nº 8.387/1991, through agreement nº 003, signed with ICOMP/UFAM, by the Amazonas State Research Support Foundation (FAPEAM) through project 122/2018 (UNIVERSAL), and by Institutional Program of Scientific Initiation Scholarships (PROPESP/UFAM) through project PIB-E/0355/2018.

REFERÊNCIAS

[1] Márcio Alencar, Lucas Amorim, Eduardo Souto, and Raimundo Barreto. Tinything: Um módulo hardware/software de baixo custo para internet das coisas. *Relatório Técnico - RT-GISE 001/2019. Universidade Federal do Amazonas. Instituto de Computação*, pages 1–6, 2019.

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

[3] Z. Cai, A. Bourgeois, and W. Tong. Guess editorial: Special issue on internet of things. In *Tsinghua Science and Technology*, 2017.

[4] S.-K. Chang, T. Ichikawa, and P.A. Ligomenides. *Visual Languages*. Plenum Press, 1986.

[5] Visualino: A desktop version of Roboblocks. A block-based programming environment for arduino [online]. Available at https://github.com/vrruiz/visualino/ (25/06/2019).

[6] Martin Erwig, Karl Smeltzer, and Xiangyu Wang. What is a visual language? *Journal of Visual Languages Computing*, 38:9 – 17, 2017.

[7] Google for Education Blockly. [online]. Available at https://developers.google.com/blockly/guides/get-started/web (25/06/2019).

[8] D. Makoshenko and I. Enkovich. Iot development: Discovering, enabling and validation of real life iot scenarios. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 159–164, May 2017.

[9] Node-RED. [online]. Available at https://nodered.org/ (25/06/2019).

[10] Karen Rose, Scott D. Eldridge, and Lyman Chapin. THE INTERNET OF THINGS: AN OVERVIEW Understanding the Issues and Challenges of a More Connected World. Technical report, Internet Society, 10 2015.

[11] Visuino. [online]. Available at https://www.visuino.com/ (25/06/2019).