Auto-Generating Google Blockly Visual Programming Elements for Peripheral Hardware

Ioana Culic
Faculty of Engineering in Foreign Languages
"Politehnica" University of Bucharest
Bucharest, Romania
ioana.culic@wyliodrin.com

Laura Mihaela Vasilescu
Department of Computer Engineering
"Politehnica" University of Bucharest
Bucharest, Romania
laura.vasilescu@cs.pub.ro

Alexandru RADOVICI
Department of Computer Engineering
"Politehnica" University of Bucharest
Bucharest, Romania
alexandru.radovici@cs.pub.ro

***Abstract*— There are plenty of tools that allow users to create their own smart gadgets. Wyliodrin allows kids and inexperienced people to start taking Internet of Things classes by offering support for visual programming with the help of the Google Blockly platform. This paper presents a novel approach to automatically generate Blockly elements in order to keep up with the emerging hardware market for IoT.**

***Keywords* — Internet of Things, embedded, visual programming, translator, Google Blockly, parser**

I. Introduction

Internet of Things (IoT) is a concept in which common objects are connected to the Internet and communicate with each other. IoT systems can be used to simplify one's life: imagine a system where by setting up the alarm clock you automatically trigger the coffee machine so that when you wake up, a nice and warm coffee will wait for you in the kitchen. Such devices are more and more frequent and everybody starts thinking of the endless possibilities they bring.

Internet of Things has become a trending technology and there are several facts in the market that support this: the explosion of sensors and the decrease of their prices, the expansion of WiFi coverage and IPv6 reliance, the emergence of wearable market, smart homes and connected cars.

The huge expansion of the IoT market also creates a need for specialists in electronics, software programming and data science experts.

Educational institutions are in front of an attractive opportunity to fill this gap but the road to it has serious barriers like complexity and advanced skills in software development. Even though IoT might seem an entertaining and easy activity, the reality is that in order to program such devices one needs to have complex knowledge about embedded computers, controllers, sensors and low-level programming skills. It is not easy to develop a curriculum to instruct people without any previous experience. The challenges range from the difficulty of connecting to the embedded board via a terminal, to the diversity of sensors that increase the complexity of the applications and also the advanced programming skills required (experience of using tools like Linux compilers, editors etc.). This is why both teachers and students either give up or lose part of their enthusiasm [13].

We propose a solution to overcome these problems. Wyliodrin is a service that allows remote programming and control of embedded boards such as the Raspberry Pi. It is a software platform designed to facilitate teaching IoT for people with no, to little experience and hardware background. It provides a graphical interface for both teachers and students to register their development boards and sensors, create code for IoT applications and finally design and use control dashboards for the IoT projects [3].

By using Wyliodrin, an education institution can overcome the previously described barriers and quickly start providing IoT courses with no advanced skills required.

The software industry is growing exponentially and countries like United Kingdom have started teaching programming classes since elementary school. However, because at such a young age pupils cannot comprehend easily how coding works, there were designed plenty of visual programming languages in order to introduce them to the programming environment [5].

Visual programming is a programming language that allows creating programs with the help of visual elements such as blocks, arrows or symbols. Writing code is replaced with connecting and combining such elements.

The platform created by us, Wyliodrin, supports multiple programming languages. In order to assist elementary school pupils to start taking IoT classes, we also added support for visual programming IoT devices. The visual editor represents an integration of the Open Source platform, Google Blockly. With all these elements at their disposal, pupils can easily learn programming and electronics.

978-1-4673-8180-2/15/$31.00 ©2015 IEEE

This paper describes our solution which optimizes the process of creating visual programming elements. The project we created offers Wyliodrin and other similar platforms, the possibility to provide users new, up to date visual programming blocks at a constant rate.

The main objective of the proposed project is to provide Wyliodrin with an additional platform that automatically generates Blockly elements.

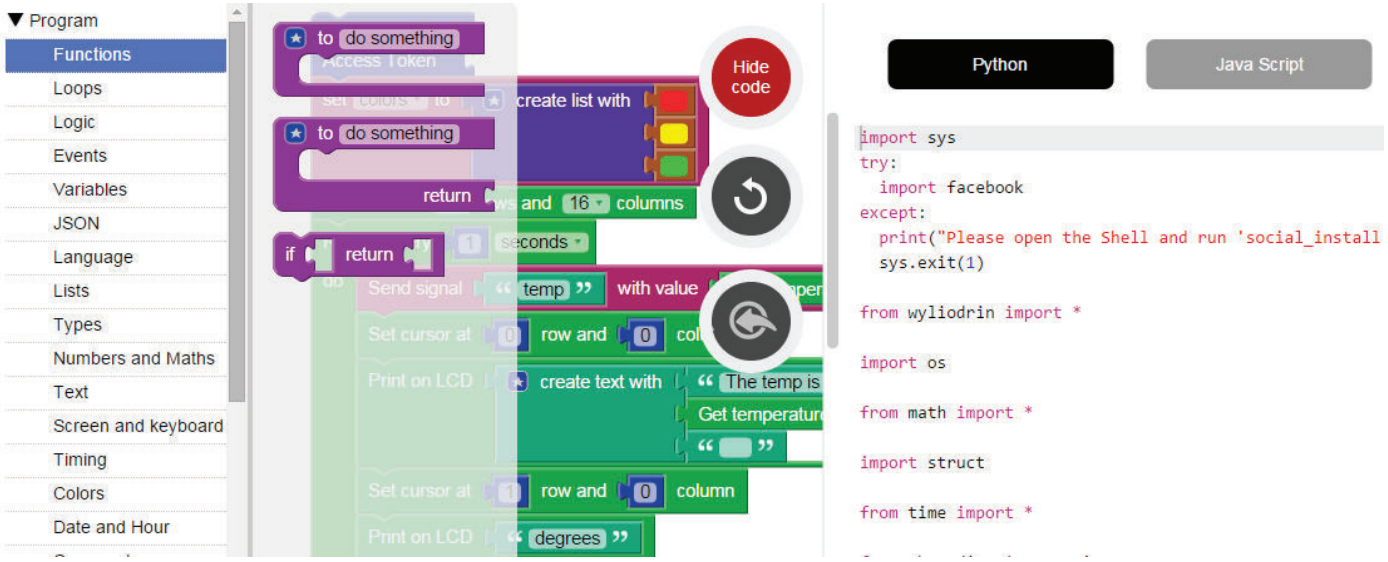

Fig. 1. Integration of Google Blockly.

II. Visual Programming Platforms' Issues in IoT

Wyliodrin is a cloud-based platform that allows remote programming and monitor of the embedded devices just by using a visual programming language [4].

In order to implement a visual programming module, it integrates the Google Blockly platform. Blockly is a visual editor that can be integrated with different platforms. It is an open source project that was inspired by Scratch. The editor allows users to drag and drop blocks and to connect them in order to create an application [11].

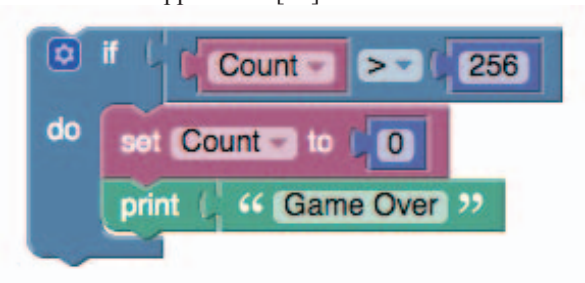

Fig. 2. Example of Google Blockly Code.

Behind the graphical interface, Blockly generates Python, JavaScript or Dart code. The generated code can also be visualized, which is of great use for people willing to learn programming [6].

However, platforms such as Wyliodrin are difficult to be kept up to date as the number of peripherals is continuously increasing. Each hardware producer that creates its own sensors provides its own way of interacting with the respective peripherals. Thus, the number of necessary visual blocks required is increasing at the same rate as the number of peripherals created, which is huge.

This is why there is a need of creating such blocks in a different way that the one provided by the Google Blockly platform. The classical way implies that somebody uses the existing Google platform in order to create each block separately and then integrate the generated element into the IoT platform which is using it.

The first step when integrating Blockly into a platform is to create a web view that will contain the editor. The editor consists of a toolbox where all the blocks are listed and a dashboard where users can drag and drop the elements. Apart from this basic setup, the interface can be customized. There is the possibility of displaying the generated code in one of the possible programming languages.

Figure 1 depicts how a Blockly integrated editor usually looks like [12]. On the left is the toolbox, in which blocks are divided in categories, while on the right is the dashboard where the blocks get connected. In order to create the custom blocks, Google Blockly offers an online editor that allows anybody to create their own blocks.

Wyliodrin integrated the editor while also creating a wide set of custom electronics blocks. With such blocks, students can make a robot move by just using a few visual elements instead of writing the whole code.

By using this editor, the blocks' design is created by also using Blockly elements. Users need to drag and drop special blocks such as Input and Field in order to describe how the desired visual element should look like. Once this is done, the Google platform generates the code which must be integrated within the main platform in order to add this specific element to the toolbox.

After the visual element in created, it is of no use until it can also generate some code. The code depends on the inputs and outputs of the block. This is why Google Blockly is able to generate a skeleton of what the code should do; mainly it sets some variables to have the values got from the user. Further on it is the developer's job to write the rest of the code. Basically, what has to be done is to write the exact code that must be generated, code that gets some values from the block's inputs.

For each of the supported programming language, this code must be created. Lastly, the code has to be copied to some files in the main platform and this is how a new block is added.

As the Blockly editor is a great acquisition for any educational programming environment, it is quite difficult to keep such a system up to date. This is due to the fact that custom blocks need to be continuously created.

III. Automatic Blocks Generation for Google Blockly

There are thousands of different existent sensors and actuators that can be connected to the existing embedded boards. As we discuss in the previous section, the field is in a continuous expansion and there is no standard way in which cards and peripherals can be connected yet. This implies that for each group of existing I/O devices or embedded cards, one must use specific functions and communication protocols.

libmraa is a C/C++ library created by Intel that offers a structured API in order to control the input/output ports of Intel Galileo, Intel Edison and other embedded devices [7]. The library also has Python and JavaScript bindings which make it easier to use.

As a result, it is easier for peripherals manufacturers to map their sensors on top of the supported hardware while users see a generic set of function they need to use in order to control the hardware.

By using this library, the number of functions that need to be implemented has shrunk dramatically. Wyliodrin uses this library in order to minimize the number of blocks that need to be implemented.

This paper presents a new way in which Wyliodrin can generate new and up to date visual blocks.

The generator consists of a script that pulls all libmraa files from the git repository and parses them, generating blocks correspondent to each described function. The script generates both visual elements and Python and JavaScript code. All these are stored in a database owned by Wyliodrin. Blocks are automatically retrieved from the database so that all this process is transparent for Wyliodrin users who simply see a complete and up to date toolbox.

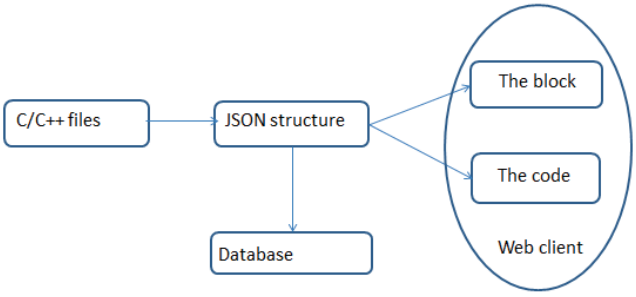

Fig. 3. Architecture of the proposed solution [8].

Once blocks were automatically generated, we noticed that 10% of them were not intuitive. Not always do the blocks explain their purpose in the simplest way (figure 4).

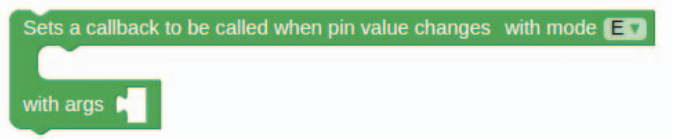

Fig. 4. Example of unintuitive auto-generated block.

From this arose another need, to offer a visual way that allows somebody to slightly modify the blocks generated by the script.

This is why we created a web platform which allows developers to search for a certain block and modify them. The existent blocks are represented under JSON format and the one bringing adjustments simply has to modify certain fields. As the JSON format is simple and intuitive, the one doing this task doesn't event require advanced programming skills.

IV. The Main Components of the Solution

A. The libmraa grammar

The whole generator is based on the libmraa library which brought us to the idea that once some generic standard functions exist, it is easy and efficient to create an automatic blocks generator.

The first step was to extract from the library's files the information required in order to obtain some intuitive visual elements. This is why the blocks generator relies mainly on the comments of every function, not on the function itself. However, the library is written in C++ and the header files ("*.hpp") contain both the functions' declarations and their implementation. As a result, a whole C++ file had to be parsed even though the implementations of functions were of no interest.

Once the format of libmraa's files was clear, the next step was to generate the file parser. For this, we used a parser generator, Jison [9]. Any input will be converted into the abstract syntax tree, which is, in fact, a JSON structure [2].

After the basic structure is extracted from the source file, the next step is the semantic analysis. This is required as parsing only verifies that the program consists of tokens arranged in a syntactically valid combination. The semantic analysis checks whether the combination translates into a feasible set of instructions in that specific programming language [1].

In this case, the semantic analysis consists of adapting the basic JSON structure to one that contains all the relevant information for generating a block. However, before all this can be accomplished, the comments need to be parsed. Each comment is parsed line by line, extracting the first part of its description and each parameter together with its details.

After everything is parsed, the next step is to do the semantic analysis. First of all, the function's name must be separated from its type. Also, in case a function argument is of an "enum" type, it should be tied to that enum structure.

After this step, the basic information for auto-generating the blocks is available.. Nevertheless, the resulted block would be not much different from the function it represents.

The purpose of Google Blockly is to replace code by using more intuitive elements. This is why it is important to adjust this structure in order to make it more suitable for this type of programming language. This is an optimization phase and consists of creating a well-built block. What was done, in these terms, is that the description of each function, which is basically the main text on the block, was, where possible, interpolated with the arguments.

In order to achieve this, the function's description was parsed word by word and in case a word was similar to the name of the argument, the description was divided around the word. This is how parameters got interpolated with the block's text.

In the end, the result is a new JSON structure that contains all the relevant information required in order to correctly generate a complete Blockly element (figure 5).

Parting from all this information, all the rest of the required details can be inferred.

This is why the next step is to create a more complex structure from which the visual elements can be easily generated.

```
{
"class":"class_name"(optional)
"name":"function_name",
"returnType":"any_type",
"parameters":
[{
"type":"any_type",
"order":"number of order in function",
"text":"corresponding parameter text",
"name":"parameter name",
"defaultValue":"the default value" (optional)
},
{
"type":"function",
"order":"number of order in function",
"text":"corresponding parameter text",
"name":"parameter name",
"arguments":[arg0,arg1,...]
},
{
"type":"enum",
"order":"number of order in function",
"text":"corresponding parameter text",
"name":"parameter name",
"defaultValue":"the default value" (optional),
"enums":[val1, val2,...]
},
{
"type":"dummy",
"text":"corresponding parameter text"
```

Fig. 5. The intermediate JSON structure.

During this process, each parameter in converted into an input together with the appropriate fields. For instance, any boolean parameter is transformed into a checkbox while any enumerator translates to a dropdown list. What is more, basic variables types such as int or char* are converted to Number, respectively String type.

In the end, results a new JSON structure that represents in an intuitive way the correspondence it has to the created block.

B. *API for Administrating Visual Programming Elements*

Starting with the block generator, we created an API that allows creation, manipulation and storage of the block structure. The API has two parts, one is for use by the Web server to generate blocks and change the JSON structure, while the other part is to be used in a Web client to display the block and insert its code in the page.

The API server part includes the generator, because it exposes the functions that generate the JSON structure from the C/C++ functions. In addition, the API also manages the storage if the blocks. It communicates with a database API which stores data structures and also alters them.

On the other hand, the web client API allows the blocks to be displayed in a browser. The module exposes a function that analyzes the JSON structure and calls the Blockly functions that build the block. Basically, this function replaces the initialization function of Blockly.

The API also consists of the code-generating function. This function analyzes the JSON structure and generates a skeleton code correspond to the block.

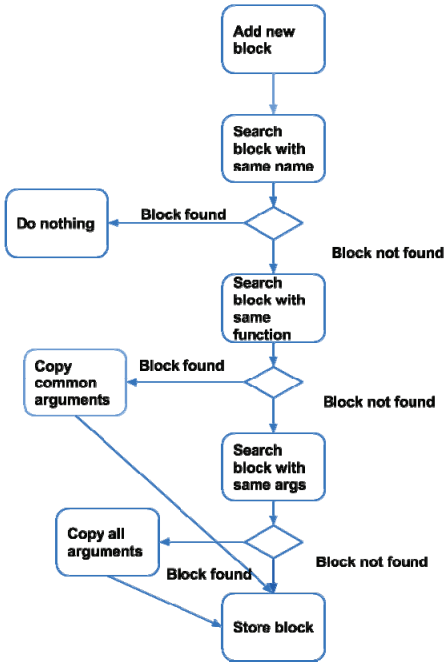

Fig. 6. Steps for auto-generating visual programming elements.

C. *The auto-generation system*

Now that the representation of each block is complete, there is required a script which would pull all libmraa's files, generate the final JSON structure and finally, store it in a database. To accomplish these tasks, a simple Node.js scrip is enough.

First of all, libmraa is cloned from the github repository. Once the files are retrieved, the code parser is used to obtain the JSON structures. Next, the structures are saved in a database.

The script should reflect the fact that the generated blocks can be modified by a user. Accordingly, in the case where the

script is rerun, it would not be recommended to overwrite the changes previously made by the user. On the other hand, it is possible to have blocks that are slightly different. For these blocks, we can take common elements from other blocks, already edited to make the whole process even more efficient (figure 6).

D. Other components

Besides the described components, we also used a database in order to store the blocks. As the blocks are represented by a JSON, one of the obvious choices for the database is MongoDB.

As both the Python and JavaScript code for each block has to be saved. In case the code is exactly the one generated by the previously described API, there is no need to store it as the generation takes place very fast. However, in case the code has been modified by a user, it is stored as part of the block's JSON structure.

Once the blocks are automatically generated and stored, there comes the second step: creating the editor. The editor is basically a web platform that allows creating new blocks or editing existent ones.

The Web editor is a simple platform consisting of a server and a client that retrieve and display the data stored in the database and that allow altering the JSON structures [10].

V. CONCLUSIONS

In conclusion, this paper describes an extension of the visual language Google Blockly which facilitates work on the integration of new blocks for Internet of Things platforms. The platform is essentially a translator that transforms C/C++ code into visual language while also supporting human intervention.

Our novel contributions are the following:

- it automates the repetitive tasks of generating simple visual elements;
- it automatically integrates the generated content with the main platform;
- it is intuitive and also reduces costs;

ACKNOWLEDGMENTS

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/187/1.5/S/155420.

REFERENCES

[1] A. Aho, M. Lam, R.Seth and J. Ullman, "Compilers : Principles, Techniques, and Tools", 1986.

[2] D. Grune, H. Bal, "Modern Compiler Design", Springer , 2012.

[3] M. Tudor, I. Culic, A. Radovici, "Getting started guide for Intel Galileo and Intel Edison using Wyliodrin", 2015.

[4] A. Radovici, I. Culic, "Open cloud platform for programming embedded systems", Networking in Education and Research, RoEduNet,vol. 12, pp. 1-2, 2013.

[5] A. Radovici, I. Culic, I. Bocicor, A. Armean, M. Ene, " Platformă educationalã care permite programarea dispozitivelor embedded din Cloud – Wyliodrin", Conferinţa Naţională de Învăţământ Virtual, 9th edition, vol. 9, pp. 57-61, 2013.

[6] M. Tătaru, I. Culic, "Wyliodrin – sistem de programare si monitorizare a sistemelor integrate", Electronica Azi Hobby, vol. 3, pp.10-12, 2014.

[7] Intel(R).http://iotdk.intel.com/docs/master/mraa, 21.06.2015.

[8] Vasilescu, Laura. http://elf.cs.pub.ro/cpl/wiki/laboratoare/laborator-01, 26.06.2015.

[9] Carter, Zach. http://zaach.github.io/jison/docs/, 26.06.2015.

[10] https://nodejs.org, 18.06.2015.

[11] Google. https://developers.google.com/blockly, 25.06.2015.

[12] https://wyliodrin.com, 20.07.2015

[13] D. Rosner, G. Sârbu, R. Tătăroiu, R. Deaconescu – "Applied Electronics Curriculum for Computer Science Students", IEEE Conference – Eurocon 2011 – International Conference on Computer as a Tool, Lisbon, Portugal, ID 176", 2011