Multi-platform and open source development tool for automation

David Delfieu, Maurice Comlan, Narcisse Assogba

Multi-Platform and Open Source Development Tool for Automation

David Delfieu

Laboratoire des Sciences du Numérique de Nantes,
Nantes University, France
E-mail: delfieu-d@univ-nantes.fr

Maurice Comlan

Ecole Nationale d'Economie Appliqueé et de Management,
Université d'Abomey-Calavi, Cotonou, Bénin
E-mail: maurice.comlan@uac.bj

Narcisse Assogba

Erlab-Noroit,
Bouaye, France
E-mail: assogbanarcisse@yahoo.fr

Abstract: This paper presents an open source tool allowing to embed a grafcet onto a micro-controller board. This tool assists the user throughout the entire process, starting from the design of the grafcet, defining and assigning inputs and outputs, generating the code, and finally facilitating its download onto a microcontroller board. The graphical part of this software use a JAVA based package JGRAFCHART for the definition of the interface, and regarding the code generation, the semantics rules of evolution are precisely presented. An application case illustrates the use of the software from modeling to electronic assembly. Temporal data relevant to the application is provided, discussing the limitations stemming from input-output constraints. All of this information helps delineate the domain and scope of applications suitable for this approach.

Keywords: Grafcet ; ARDUINO ; Programmable Logic Controller ; Code Generation.

1 Introduction

This article is a revised and expanded version of a paper entitled Edit and Upload grafcet on an Arduino Boards presented at the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME) in 2021 [2]. Compared to the previous work, the major evolution, within the code, is the algorithm - which is presented in Section 3.2 - that computes the evolution rule. It has been improved and optimized (see the Algorithm 1). Additionally, instructions operating on outputs have been added to calculate the *Cycle Time*. As suggested by the Figure 1, this paper endeavors to present an advanced software approach for editing, compiling, and uploading grafcetdiagrams onto Arduino boards, catering particularly to small-scale or educational applications with cost-efficiency as a priority.

The specification language GRAFCET [3] intricately describes the control aspects of automated systems in an unambiguous and intelligible manner. It employs a graphic representation that readily conveys concepts like parallel and sequential evolution, as well as emergency stop functionalities, making it easily comprehensible and accessible. Comparable approaches and tools in the domain, such as Petri nets [6, 10] share similarities with GRAFCET but diverge in their formalisms and semantics. Petri nets offer a mathematical foundation for modeling control systems, including conflicts, synchronization, concurrency, and resource sharing, often incorporating rigorous semantics to define properties and formal reasoning.

However, despite GRAFCET's derivation from safe and non-autonomous Petri nets [5, 12], distinctions in the associated semantics of evolution rules exist, leading to potential instabilities in certain conflict situations. Practical design rules have been enacted to navigate these issues, emphasizing the importance of ensuring exclusivity in transition conditions. In industrial environments, Programmable Logic Computers (*PLCs*) [8, 15] serve as the physical apparatus for real-time control in automated systems where grafcet diagrams find deployment. While PLCs offer robustness and connectivity, their cost and complexity can be substantial. Alternatively, smaller-scale applications can benefit from the use of microcontrollers like Arduino boards, providing cost-efficiency and relative

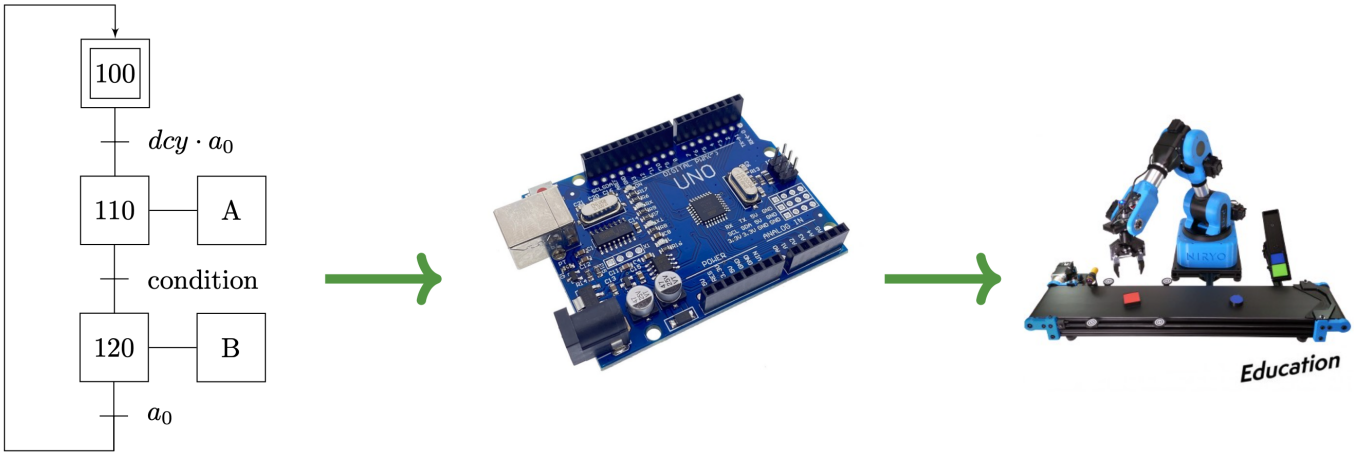

Figure 1: Embedding Grafcet on ARDUINO

reliability for specific contexts such as laboratory exercises or prototypes. The major software packages on the PLC market, such as *PL7Pro* or Unity pro [14], do not offer solutions for embedding on small 8-bit microcontrollers. This is to be expected, as they focus on solutions that ensure safety, security and maintenance. Only Codesys, offers solutions based on microcontrollers. However, these are solutions based on powerful 32-bit ARM-type cores or 64-bit processors which are not really cheap.

Our contribution focuses on presenting a software chain tailored for small, low-cost applications where high availability and security are not paramount. Specifically, we propose an approach for editing, compiling, and uploading GRAFCET diagrams onto Arduino boards, catering to scenarios requiring cost-effectiveness over extensive reliability and availability.

In Section 2 we present some basics concept (programmable logic controller, Sequential Function Chart) and give a definition of the semantic of the evolution in a GRAFCET. Then, in Section 3, we present the interface and the use of tool JGRAFCETTOARDUINO. Finally, in Section 4.2, a case study is proposed. This example implements a fire detection system based on brightness and temperature threshold detection.

2 Basics definition

2.1 Programmable Logic Controller

A *PLC* is a device for controlling an industrial process (manufacturing system, production line,...) consisting of a power supply, inputs/outputs and a link for interfacing with a PC. A programmable logic controller (PLC) controls the manufacturing processes for integrated production lines and equipment. PLCs were designed to replace the need for a large bank of relays or timers in facilities with numerous inputs and outputs. Due to their durability and ability to automate multiple processes, PLCs have become a staple in modern manufacturing. In this section, we will review the basic architecture of programmable logic controllers [1]. The main components of a *PLC* are: (i) Power Supply: Inputs and outputs are generally isolated (galvanic isolation). Inputs support a voltage of $24v$ while outputs offer different standardized voltages. (ii) Central Processing Unit: it is usually made up of a micro-controller and its role is to continuously perform cycles. Each cycle is composed of the reading of the inputs, the evolution of the system control and the assignment of the outputs. (iii) I/OModules: They allow to connect the micro-controller to the physical process under control. They prevent the use of relay providing several normalized voltage: $24v$, $48v$, $110v$ or $220v$. (iv) Programming Device: A *PLC* requires a programming interface and a software chain. The programming device can be a handheld device, a desktop console, or a computer. The software chain is used for the edition, the monitoring of the I/O, and the download or the upload of the program.

2.2 The GRAFCET *and Sequential Function Chart*

The "Association Française pour la Cybernétique Economique" (AFCET) proposes in 1977, a sequential function chart called GRAFCET and in 1987 an international norm, *IEC-60848* is defined. This norm defines the following components:

- Situation: Defines the actives steps of the grafcet.
- Steps: A step can be active or inactive. A step represent a state of the sequential part of the system.
- Transitions: A transition is associated to a transition condition and indicates a possible evolution between input steps and output steps.
- Evolution: An evolution is the clearing of one transition moving the system from one situation to another. A *transient transition* leads to the clearing of several successive transitions on the occurrence of a single input event.

- Directed link: Connect steps to transitions or transitions to steps.
- Transition condition (receptivity) : Evaluation of a grafcet boolean expression. A *grafcet boolean expression* is a boolean epxressions where terms can be boolean variables, input events or internal events.
- Input event: Rising edge or falling edge of events, boolean variables, internal events or *grafcet boolean expressions*.
- Action: Activity associated to a step or affectation of output variables.
- Internal event: Event associated to the current state of the grafcet.

In the Figure 2, illustrates links, transitions, and receptivities. Step 100 and 1 participates in the initial situation. The *or divergence*, is associated to two antagonist receptivities. If *code* is true the control passes 0 to state 11 else to state 21. Concerning the last receptivity, an *And convergence*, the control passes in state 1 as soon as the control is simultaneously in state 21 and 100.

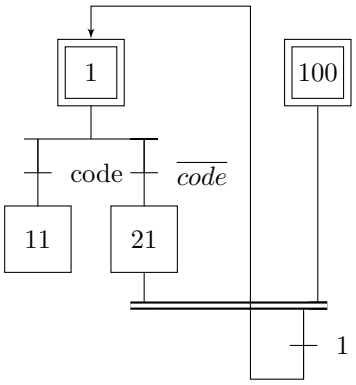

Figure 2: Cash Dispenser

Moreover abstract components such as macro-steps, encapsulating steps, parallelism and synchronization elements (divergence/convergence) improve expressive power. *IEC-60848* defines five evolution rules. These rules describe the principle of sequential evolution between situations. A situation (a marking in term of Petri nets) if formally defined as the set of active steps for a given time.

R_1 *Initial situation:* The initial situation defines the set of actives steps at the initial time.

R_2 *Clearing a transition:*

- Enabled transition: A transition is said *enabled* if every previous steps (linked to this transition) are active.
- Clearing a transition: The clearing of a transition occurs when the transition is *enabled* and *when* its receptivity is *true*;

A clearable transition is immediately cleared.

R_3 *Evolution of active steps:* The clearing of a transition provokes simultaneously the activation of all the immediate succeeding steps and the deactivation of all the immediate preceding steps;

R_4 *Transient evolutions:* If several transitions can be cleared simultaneously they are simultaneously cleared;

R_5 *Activation of a step:* If a step is included in the preceding situation and in the following one, it remains active. Moreover, if during an evolution, an active step is simultaneously activated and deactivated, it remains active.

An evolution from a situation may lead to a *transient* or *stable* situation. A situation is called *transient* if at least one transition may be cleared from it without any new change in transition-condition.

2.2.1 Semantics

IEC-61131-3 standard includes languages (Ladder, Instruction List,...) and graphical language (Sequential Function Charts) for programmable logic controllers. Among theses approaches two kinds of semantics coexists: The event-driven and a time-driven semantics. GRAFCET is defined with an event-driven approach. If we consider a simple sequence where several consecutive transition-condition are true, the evolution rule R_4, enforces that from a stable situation it evolves to transient situations until it reach a stable situation. On an other way, Ladder language or Instruction List, are based on the fact that the clearing time is short but null. In practice an input scanning cycle allows to define stable situation. The algorithm defined in Section 3.2, is based on the definition of an input scanning cycle allowing to only consider stable situations.

2.3 Arduino *boards*

Arduino is an open-source project which allows to implement code simply and quickly on micro-controller. Arduino refers as well to a hardware architecture, an *IDE* (Integrated Development Environment) and a programming language. Arduino boards are generally built around Microchip micro-controller ($ATMega_{328}$, $ATMega_{2560}$,...), and peripheral chips that ease programming and interfacing. Each board have digital input-output, pwm ouptuts, analogic inputs, ceramic resonator, USB connection, sram, flash and eeprom memories. The IDE allows to edit, to compile, to select the core, the board and the serial port, upload the byte code to the board. The programming language is based on C++ and can be compiled with avr-g++ .

3 JGrafcetToArduino

3.1 Editor

JGrafcetToArduino is a tool [4] freely available to *https://github.com/maurice-comlan/Grafcet-To-Arduino*. It requires fisrt the installation of NetBeans, JDK and Arduino. This tool is inspired of the JGrafchart , Automgen 8 and Unity Pro XL interfaces, in particular form for some elements of the graphical interface. The main window (see Figure 3) includes the following components:

- *Menu*: File, Edit, View, Execute, Automaton, Help.
- *Action*: Actions related to file (new, open, change directory), component edition (cut, paste, delete), actions related to software (compile, upload),. . . .
- *Side palette*: Step, transition, parallel split, parallel join, macro steps,. . . .
- *Editing box*: Panel where the grafcet is drawn. From side palette, components can be added by a double click. A right click in the panel brings up a contextual menu.
- *Error area*: At the bottom of the main window a message line display message issued from the execution of an action or errors.

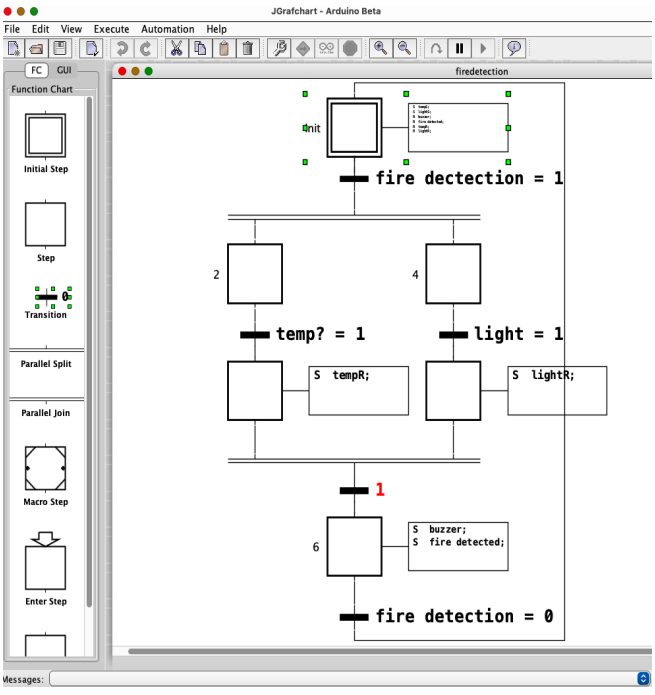

Figure 3: Editor main interface

The JGrafchart interface was used as a model and specifications for the implementation of our software. JGrafchart is a freeware developed in the Automation Department of Lund University in Sweden [16]. This software was developed in Java with the Swing library [7], which makes it compatible with every operating systems. Some unnecessary features for our application have been removed while others have been added:

- *Misc*: Has been disabled. This item controlled the application's interconnection to a computer network via a server on which files are shared using the Devices Profile for Web Services protocol.
- *Control module by PID*: Has been disabled. It was in used to command servomotors. [13].
- *Side palette*: Some components, related to disabled modules, have been removed.
- *Automaton*: Has been created. It manages the interactions with the Arduino board. Allows to:
  - Edit grafcet variables (see figure 8);
  - Define the type of card and the *COM* port to which it is connected;
  - Generate the source code;
  - Upload the code.
- *Variable management* has been redesigned to be similar to Unity pro or PL7. Three types of variables are now available: Standard variable, function block, and transition session.
- *Edit box* has been modified to include modification on step and transitions.
- *File* has been enriched with "Open recent files"
- An *Upload* icon has been added to the quick action bar.

The edition phase produces several XML files that encode the structure of the grafcet. In the code generation phase, theses files will be parsed to extract a set of essential objects structuring the grafcet.

3.2 Code generation

Concerning the code generation phase, a pseudo interpretation code (Algorithm 1) is presented below. This algorithm computes the evolution rule of a grafcet. The essential rule is $M_2 = M_1 \rightarrow^t$ which compute the marking M_2 after the firing of the transition t from the marking M_1.

The inputs of the algorithm are M_0 (the initial marking), *initial_steps* the set of initial step(s) of the grafcet and *receptivities*[] the set of boolean expressions associated to each transition. Theses sets have been parsed from the structural definition after the edition phase from XML intermediate files (see Section 3.1).

Moreover, $Enabled = enabled(M)$ is the the set of enabled transitions for a marking M. $receptivities[t]$ is the receptivity associated to the transition t.

The outer *while* implements the input scanning cycle evoked in section 2.2.1.

Algorithm 1: Evolution Step

Input: M_0, initial_steps, receptivities[]

Activate Initial_steps;
foreach *istep* $\in$ *initial_steps* **do**
  activation of actions associated to istep
$M_1 \Leftarrow M_0$;
$Enabled \Leftarrow enabled(M_1)$;
while *(true)* **do**
  $transient \Leftarrow true$;
  while *(transient == true)* **do**
    $nbfired \Leftarrow 0$;
    $read(receptivities)$;
    foreach $t \in Enabled$ **do**
      if *(receptivities[t] == true)* **then**
        $M_2 \Leftarrow M_1 \rightarrow^t$;
        $nbfired \Leftarrow nbfired + 1$;
        $transient \Leftarrow true$;
        foreach $istep_1 \in M_1$ *and* $istep_2 \in M_2$ **do**
          activation and deactivation of actions associated to $istep_1$ or $istep_2$;
        $M1 \Leftarrow M_2$;
        $Enabled \Leftarrow enabled(M_1)$;
    if *(nbfired ==0)* **then**
      $transient \Leftarrow false$;

4 Case Studies

4.1 Cash Dispenser

The first case study refers back to Figure 2. This grafcet refers to a simplified version of a cash dispenser. *Step* 1 models an initial state where the code has not yet been entered. If the code is entered successfully, the system transitions to *Step* 11; otherwise, it goes to *Step* 21, where *Step* 100 allows the user another code attempt. As there is no loop connecting *Step* 21 to *Step* 100, there is only one additional attempt.

The implementation with JGrafcetToArduino give the following screenshot (figure 4) :

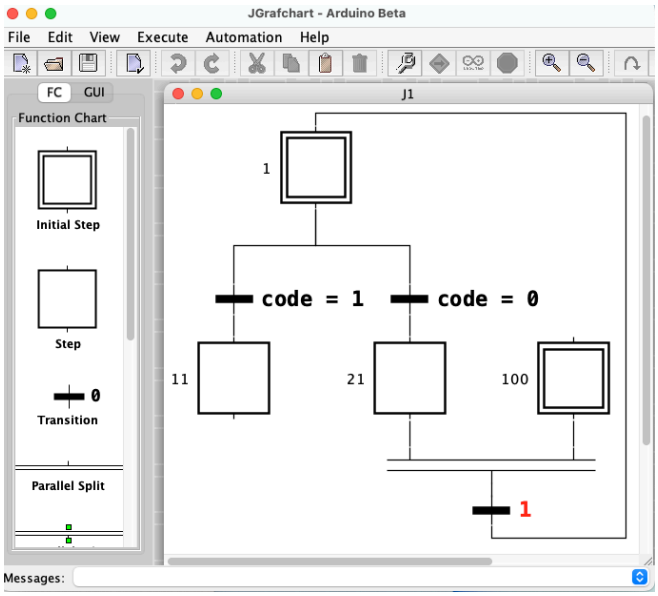

Figure 4: Cash dispenser to JGrafcetToArduino

After the edition phase, the code generation and the code uploading, the system can be tested with the following (figure **??**) montage diagram:

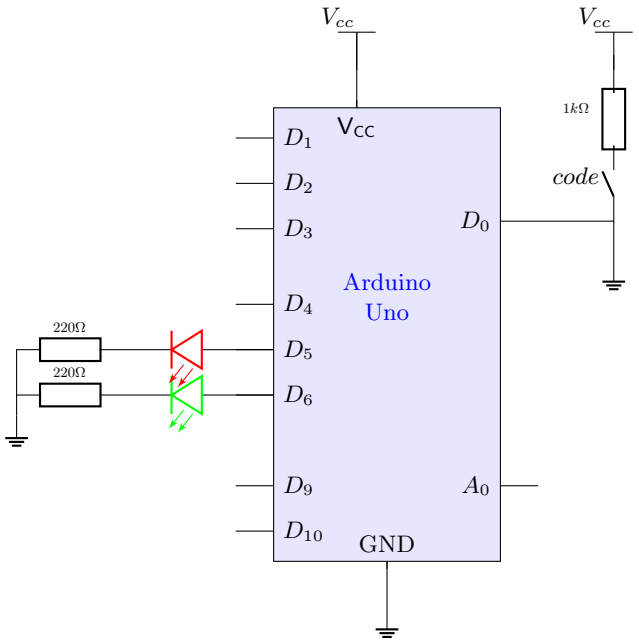

Figure 5: Electric Diagram

Montage diagram : . A green led is connected to D_6 to indicate that the code has been validated, either a red led indicates that the code has not been validated.

4.2 Fire Detection System

Our case study focuses on a fire detection system based on a dual detection system (see the Figure 6). Three bi-colour leds define a situation. A temperature sensor and a photo-resistor monitor the system. If the temperature is normal, the first led is green. It becomes red if the sensor detects that the temperature exceeds a certain limit. The second led is green or red upon the value return by the photo-resistor. The last led assets that temperature and brightness limits have been exceeded. A switch button allows to start or to stop the detection. The figure 7 presents the grafcet modelling the application case:

After the edition, variables are edited in order to connect the grafcet to its environment. The figure 8 illustrates that the outputs `buzzer`, `tempR`, `lightR`, `firedetected`,`lightR` have been affected to Q_5,Q_6, Q_7, Q_8, Q_9 and Q_{10} while inputs `light`? and `fire detection` have been affected to I_2 and I_3. `temp` is an analogic input

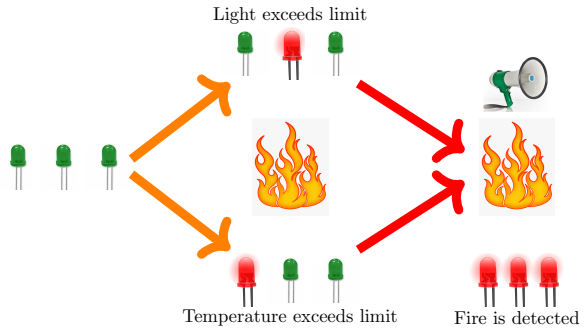

Figure 6: Application case

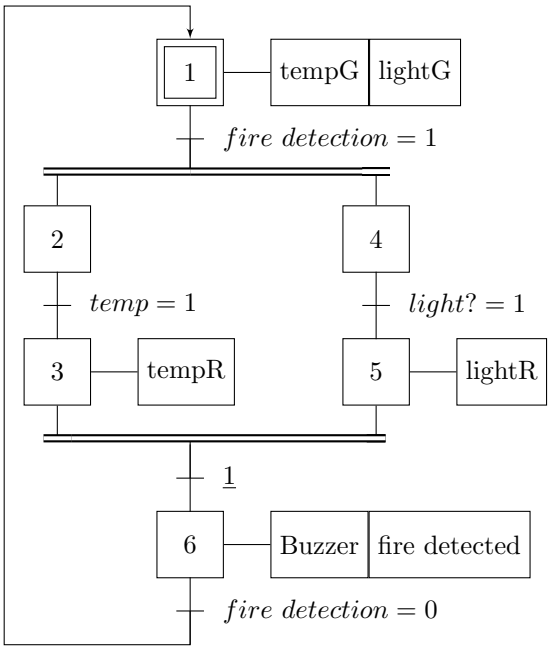

Figure 7: Grafcet of fire detection system

connected to the temperature sensor (LM35) associated to the adress A_0

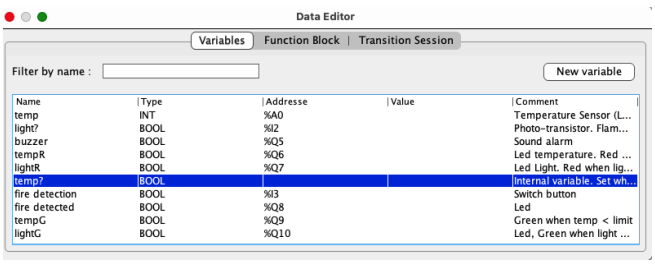

Figure 8: Variables of the grafcet

The design of the grafcet and the use of several functionalities of the tool has allowed to check the following aspects:

- Analogic and digital outputs;
- Steps without action or with multiple actions;
- Simple and complex transition-conditions;
- Forced transition;
- Parallel evolution and synchronization;
- Serial port selection, code generation, compilation and uploading into an ARDUINO board.

Montage diagram : . Figure 9 shows the component layout that was used for the validation of the example. The temperature sensor (LM35) is wired on the analogic input A_0, while the buzzer is wired on a PWM output. An anti-bounce filter is bounded to the switch S_1. Finally, the test was carried out with a source of temperature. It has allowed to validate the expected evolution of the grafcet.

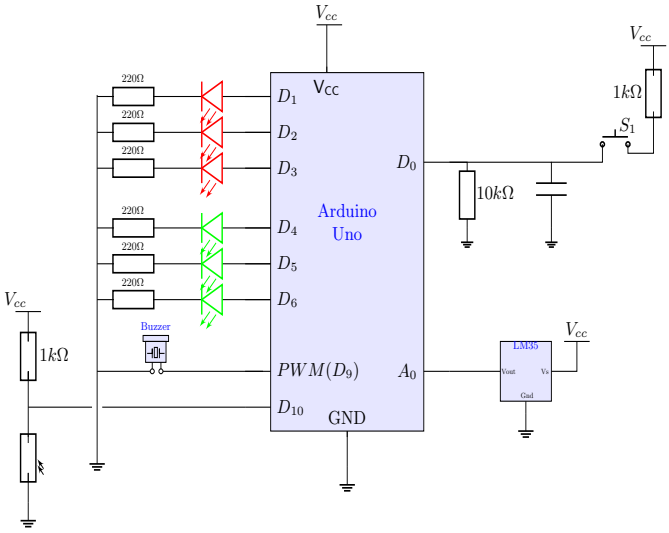

Figure 9: Electric Diagram

5 Analyses and discussions

For the grafcet edition, we heavily relied on the performance. JGRAFCHART is an open implementation of Grafchart written in JAVA. It is a language allowing to describe sequences constituted of steps, transitions with parallel or alternative paths. This language made it easier the implementation of this software. We have restrained our work to a subset of classes, and consequently, we have lost the checking of the consistency of actions associated with steps and conditions associated with transitions. Moreover, we migrated from the JGRAFCHART to C++ especially for compilation and downloading to ARDUINO boards. The grafcet panel is interpreted as an *xml* file, we have therefore integrated an XML parser, and a code generator into our editor to obtain this functionality. This change had also an impact on the simulation mode, which strongly depends on the syntactic and semantic analyzer to calculate the value of the variables and to interpret the actions.

For this first version, some GRAFCET functionalities have not been taken into account: Structuring by forcing the situation of a partial grafcet and structuring by encapsulation [3].

5.1 Testing

This application case has allowed the testing of all the interface sub-menus and item, with the different kinds of receptivities, action triggers, divergences, convergences, and sequential or parallel executions as well as the downloading of code to the ARDUINO board.

Moreover, the tests carried out on the ARDUINO Uno boards were conclusive. In the application case, actions associated with steps were performed in the expected order. Steps have been indeed activated and deactivated while transitions have been sensitized. Further implementation (in progress) will allow the use of new types of ARDUINO boards (Nano , $ATMega$, ...).

The installation of this software in a WINDOWS environment requires the installation of the program *nmake*, in addition to the C++ compiler, to take into account the *makefile* which is not supported by default on WINDOWS. This software has been tested successfully in a MACOS environment [1].

5.2 Metrics

The complexity of the algorithm (see Section 3.2) can be estimated to a polynomial order because of the imbricated loops. So the *Cycle Time* of input scanning cycle is proportional to the product of the number of places ($|P|$) and the number of receptivities ($|R|$).

With an $ATMega_{2560}$ clocked at 16 MHz, the Table 1 gives the *Cycle Time* (CT) and the memory amount of the embedded grafcet corresponding to the example for a grafcet containing $|P|$ places and $|P|$ transitions. . The first line of the table corresponds to the initial grafcet presented in section 4.2 while the following lines correspond to examples of growing complexity. Third column give execution times of the *Cycle Time*. Memory amounts are given by the compilation phase. Memory amount increase linearly, without surprise, with the size of the matrices which contains the structure of the grafcet (proportionally to the size of $|P| + |R|$). Whereas, the execution time grows more significantly: As expected, the execution time increases in a polynomial order of $|P| * |R|$. For the last test, 24 places and 20 receptivities, the *Cycle Time* rises to 2.5 ms, it corresponds near to the maximum number of input/output of the $ATMega_{2560}$.

Table 1 *Cycle Time*, memory ammount for various-sized grafcet

Num. of Pl. and Recepts.	*CT*	*Mem. Amount*
$\|P\| = 6$ $\|R\| = 5$	150 μs	592 bytes
$\|P\| = 12$ $\|R\| = 10$	180 μs	1.2 kbytes
$\|P\| = 18$ $\|R\| = 15$	840 μs	2.1 kbytes
$\|P\| = 24$ $\|R\| = 20$	2.5 ms	3.3 kbytes

6 Conclusion

The objective of this paper was to present a tool which allows the design of a grafcet editor, the simulation, compilation and code generation, a serial port selection and the downloading to an ARDUINO board. The main idea was to make available an open source, multi-platform automation workshop which provides practical industrial automation designed around ARDUINO micro-controller boards.

In view of the objective set, the following modules have been developed:

- grafcet editor: This software integrates the basic concepts defined of *IEC-60848* and *IEC-61131-3* standards;
- Code Generator: It includes a module for interpreting grafcet and translating it into source code for an ARDUINO board;
- Compilation, Serial configuration, and Uploading module: It includes the possibility to compile, to select an ARDUINO board, and to upload it.

Moreover an application test case was developed to test the most important features and validate the work that was done. It is a *fire detection system* based on a dual captures system. A flame sensor that triggers a first light signal initially captures excessive light at the fire scene. The abnormal temperature increase is then captured by a temperature sensor, which triggers a second light signal. These two conditions trigger a third light signal followed by an audible signal. The action of an operator on a button stops the different light and sound signals.

The scalability of this type of application offered by the JGRAFCHART package is a major asset, as it will allow the editor's functionality to be improved by adding, for example, the languages defined in the *IEC-61131-3* standard to the list of languages used to edit actions related to a step or define instructions for block functions. In the perspective of this work, it would be interesting to add to the editor the other formalisms such as Ladder and also to extend the possibilities of automata to other micro-controller boards.

The limitations that have been discussed in the Section 5.2 on this approach are related to the number of physical inputs and outputs on the chip. However, one could imagine an approach that would multiplex the inputs and outputs using an external component.

The edition rules edited in *IEC-61131-3* can bring to problematic situations. Conflict transition combined to rule R_4 (Section 2.2) and loop can lead to unstable situation, moreover some structure can lead to non safe graph [11] and this implementation could allow to test alternative semantic rules.

References

[1] Ephrem Ryan Alphonsus and Mohammad Omar Abdullah. A review on the applications of programmable logic controllers (plcs). *Renewable and Sustainable Energy Reviews*, 60:1185–1205, 2016.

[2] Maurice Comlan, David Delfieu, and Narcisse Assogba. Grafcet to Arduino: Edit and Upload Grafcets on an Arduino Boards. In *The International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Mauritius, Mauritius, October 2021. IEEE.

[3] International Electrotechnical Commission. Iec 60848 ed. 2: Specification language grafcet for sequential function charts, 1999.

[4] M. Comlan D. Delfieu. Download grafcettoarduino. https://github.com/maurice-comlan/Grafcet-To-Arduino. Updated on : 2023-11-24.

[5] R. David and H. Alla. *Du Grafcet aux réseaux de Petri*. Série automatique. Hermès, 2 edition, 1992.

[6] Michel Diaz. *Les Réseaux De Petri - Modèles Fondamentaux*. Ic2 Traité Informatique Et Systèmes D'information. Hermès, 2001.

[7] Robert Eckstein, Marc Loy, Dave Wood, and Mike Loukides. *Java swing*. O'reilly Cambridge, MA, 1998.

[8] Felipe Fronchetti, Nico Ritschel, Reid Holmes, Linxi Li, Mauricio Soto, Raoul Jetley, Igor Wiese, and David Shepherd. Language impact on productivity for industrial end users: A case study from programmable logic controllers. *Journal of Computer Languages*, 69:101087, 2022.

[9] Microsoft. Makefiles (Windows), 2017.

[10] Carl Adam Petri. *Communication with automata*. PhD thesis, PhD thesis, Institut fuer Instrumentelle Mathematik, 1962.

[11] Valette R. Grafcet une introduction. http://valetterobert.free.fr/enseignement.d/grafcet.pdf. Updated on 2022-02-15.

[12] Valette R. Etude comparative de deux outils de representation: Grafcet et reseau de petri. *NOUV. AUTOMATISME; FRA; DA. 1978; VOL. 23; NO 12; PP. 377-382; BIBL. 7 REF.*, 1978.

[13] Daniel Ross, Etienne Deguine, and Mickaël Camus. Asservissement par pid. *rose. eu. org*, 3, 2010.

[14] Schneider Electric. Logiciel Unity Pro - EcoStruxure™ Control Expert | Schneider Electric, 2017.

[15] Martin A. Sehr, Marten Lohstroh, Matthew Weber, Ines Ugalde, Martin Witte, Joerg Neidig, Stephan Hoeme, Mehrdad Niknami, and Edward A. Lee. Programmable logic controllers in the context of industry 4.0. *IEEE Transactions on Industrial Informatics*, 17(5):3523–3533, 2021.

[16] Årzén, Karl-Erik. JGrafchart: Sequence Control and Procedure Handling in Java. In *Proceedings of Reglermötet 2002*. Reglerteknik och kommunikationssystem, Linköpings Universitet, 2002.