A Simulation Platform for Automation Teaching

Nuno Canadas, Carlos Barros, José Machado, *IEEE Member*
R&D Center CT2M,
Mechanical Engineering Departament
Guimaraes, Portugal
jmachado@dem.uminho.pt

Filomena Soares, *IEEE Member*
R&D Center ALGORITMI
Industrial Electronics Department
Guimaraes, Portugal
filomena.soares@algoritmi.uminho.pt

***Abstract*—This paper presents a simulation platform that was developed on the context of Automation teaching to students of Mechanical and Electronics Engineering. All the steps, for the development of the platform, are presented and, also, used formalisms and tools are described. A case study is presented, in order to illustrate all steps performed in this context. The main advantage of the simulation platform is that the associated methodology of development can be reproduced to other illustrative examples and, this way, students have more possibilities for learning using new strategies and teaching methodologies related with laboratorial practice.**

Keywords—Automation; SFC; Ladder Diagrams; Finite Time Automata.

I. INTRODUCTION

The teaching/learning process has been under a detailed analysis and severe changes have been slowly implemented [1] through the Bologna Treaty. The reduction of contact hours between teachers and students centralizes now the main role of the teaching/learning process in the student [2]. The attitude towards education has now changed. This implies a precise definition of objectives and capacities that the student must acquire, but also strategies and teaching methodologies reformulations. The student now should have more initiative and work harder to his education besides going to classes and listening to the lecturer and taking notes.

It is well known that students retain better what is taught if they are able to practice [3]. The concept "Learning by doing" is particularly important for engineering students, once in the future they will be confronted with practical situations in their professional career. Therefore, laboratories and working places to do practical exercises and put on practice the theory given in classes are of the utmost importance. However, it is difficult to have and maintain these working places as the devices are expensive, there is not enough laboratorial place and qualified personnel to help and supervise students are not sufficient [1].

To make it possible for students to put their knowledge into practice without the need of laboratories and working places, we can turn our attention to simulation in virtual environment and animations. In this way, students, with the tools given by the teacher, have the possibility of performing simulations of practical exercises anytime, anywhere, with the only need of a Personal Computer (PC) and the right software. The student can also have an immediate feedback, allowing him/her to work at his/her own pace [1].

There are also several remote laboratories in the Internet [4], with applications in electronic problems and process control, but there is still a gap in what concerns automation subject. For example, there are not Programmable Logical Controllers (PLC) remotely available for students to test their real-world programs, as Ladder diagrams [1].

In order to develop such environment, it is necessary to model the controller behaviour, the plant behaviour and the interaction between the both models, following the schematic representation presented in figure 1.

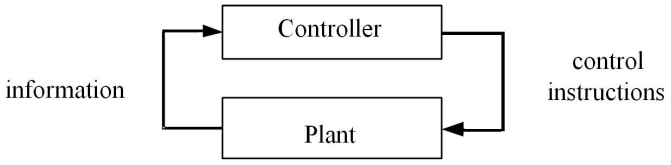

Fig. 1 - Schematic representation of an automation system parts interaction.

The interaction between both models (done, in real-world by electrical signals) must be modelled as Boolean and not-Boolean variables that will be the basis for the elaboration of the respective models.

With this work, we intended to develop a tool where students can test the behaviour of automation systems corresponding to the exercises given in the classes of automation, through the conversion of the specification of the system behaviour to the respective controller program and the visualisation of animations of these exercises working as commanded by the student.

For this, Sequential Function Charts (SFC) [5] (or Grafcet) formalism is used for the specification of the command part of the systems, because it is the one taught during the automation lectures, it is easy to use, easy to learn, by students, and well adapted to the specification behaviour of sequential systems. After the formalisation of the command

part of the system (by SFC) this specification can be, latter, translated in algebraic equations that will be the basis for elaboration of the Ladder [6] program, to be inserted into the PLC. [7].

For modelling the behaviour of the plant, it was used Finite Timed Automata. This part is inaccessible to students because they only need to design command programs to simulate the exercises. Also, if they accidentally change it, the simulation will go wrong and will not behave as it should be. This formalism is well adapted for modelling the plant behaviour because allows modular modelling approach [8], it is a non-deterministic formalism [9] and has timed aspects formally well-defined and formalised.

In order to achieve the proposed goals, the paper is divided in six chapters: Introduction, Fundamentals of SFC and Timed Finite Automata, Methodology of work, a simple Case-Study to show (and extrapolation) of practical results, Conclusions made along the paper and References consulted.

II. THEORETICAL FUNDAMENTALS

In this chapter there are explained some of the theoretical aspects of the developed work.

A. *SFC fundamentals*

SFC [5] is a specification language for the functional description used to describe the deterministic behavior of the sequential part of an automated system. The SFC fundamental structural elements are: step, transition and link as it is illustrated in figure 2.

The step defines the current state of a sequential control system. A dot may be used to mark which steps are active in a certain evolution instant. The number of active steps in a SFC defines the situation of the corresponding system. A step may have associated a particular action (for example, Turn on a motor).

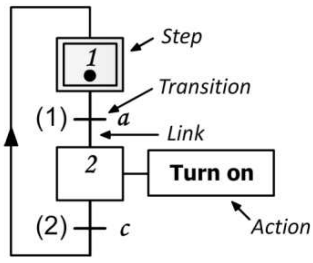

Fig. 2 - GRAFCET fundamental elements [2].

The SFC fundamental interpreted elements are transition-condition and actions, also illustrated in figure 2. Here the term interpretation refers to a connection between the sequential system input and output variables which is established through the transition-conditions and actions.

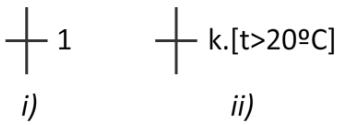

Fig. 3 - Examples of transition-conditions, accordingly to standard IEC 60848 [2].

The transition means the possibility of an activity evolution between two steps. Examples of transitions and their respective conditions are presented in figure 3, where: i) means an always true logical value, and ii) illustrates a logical condition involving a temperature set-point comparison.

In the design of a SFC, a syntax rule must be followed by always alternating steps and transitions. The SFC evolution is determined by five rules, which defines [10]:

- The initial situation;
- Clearing of a transition;
- Evolution of active steps;
- Simultaneous evolutions;
- Simultaneous activation and deactivation of a step [5].

B. *Finite Timed Automata Fundamentals*

Finite Timed Automata are finite state machines that can manipulate time, through the application of clocks. These can be used to model and analyze the timing behavior of computer systems, e.g., real-time systems or networks [11]. The Finite Timed Automata fundamental structure elements are: location, guard, invariant, transition, message and variable assignments as it is illustrated in Figure 4.

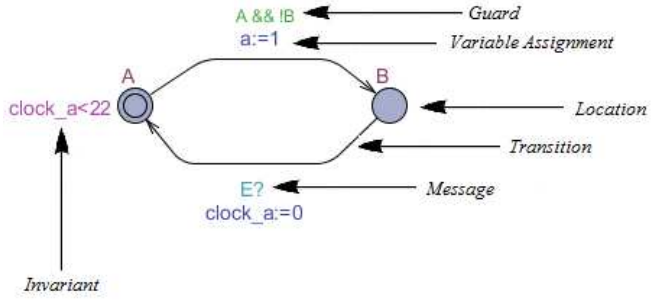

Fig. 4 – Finite Timed Automata fundamental elements.

The location defines the current mechanism where the system is in its evolution. Each transition is labeled by a logical equation called guard, which indicates that the transition can be fired. Variable values can be assigned when the transition is fired by variable assignment conditions. Invariants (conditions that must be always respected) can be used too. Messages are used to link and synchronize different automaton evolutions, when it is intended to have simultaneous evolutions of two or more modular automaton.

III. METHODOLOGY

As previously mentioned, the objective of this work is to develop a virtual environment for simulation and testing the automatic control of automation systems which configuration is illustrated in figure 1 [12].

The methodology developed divides the problem in two different parts: the modelling of the plant behaviour and the specification of the controller behaviour, considering that they exchange information between them. Each part was separately modelled and simulated following different approaches: the plant was modelled by timed automata and the controller behaviour was modelled by SFC [5].

In order to be sure about the correct modular modelling of the plant, the automata models was simulated and, this way, some mistakes and imprecisions were eliminated before connecting the plant model to the controller specification model. This simulation was performed using the simulation software UPPAAL [13], well adapted for modelling and simulation of this formalism.

The same software tool UPPAAL was used in order to simulate the interaction between the controller specification model and the plant model. For this, the controller specification model was translated to timed automata using the approach proposed by [14].

After constructing all the model (controller specification + plant + interaction between them) the global model was translated to Ladder language, following the approach proposed by [15] and the software of company OMRON [16].

Simulations were developed using OMRON's software CX-One (CX-Programmer, CX-Simulator and CX-Designer) [12]. The model of PLC used was CP1L and the memory areas of the PLC were defined as illustrated in Table 1. The information exchanged between the two parts is exchanged in common memory areas.

Table 1 - Words used in PLC CP1L

Function	Words and Bits
Inputs	0.00 to 0.07 & 1.00 to 1.07
Command Program	3.00 to 99.15
Outputs	100.00 to 100.07 & 101.00 to 101.07
Operative Program	103.00 to 399.15

Also, during the scripting of the transitions and the actions definition in Ladder language in CX-Programmer, capital and lowercase letters, were used to distinguish the command and the operative programs, respectively.

It is intended that the virtual simulations behave as the real automated systems do, thus the simulation has to be similar to the real automated system.

The plant models were divided in three different parts:

i) Actuator part such as motors, cylinders and valves;
ii) Model influenced by the actuators work such as barriers, treadmills, cranes;
iii) Product models that are, for instance, the objects moved/produced in the simulation as parts, trains, fluids, for example. Usually, these models don't react directly to orders coming of the controller model and, sometimes, they are created and developed under the concept of "observer automata". [17]

The models described in i) and ii) were designed as blocks that can be instantiated in different case-studies, being available in a database of function blocks. Models described in iii) are specific to the problem in analysis and they rarely can be reused in different exercises, due to their specificity.

IV. CASE-STUDY

A simple, real-world problem, which is an illustration of the use of automation in real life, was selected as the case-study scenario. Furthermore, this exercise shows how the three different parts of the program can be modelled, and how they interact with each other.

A. *Problem*

It is intended to develop a command program of a railroad gate with an automatic barrier, which moves downwards (MD) when a train comes to the gate and upwards (MU) when it goes away. The train can come from both directions.

There are three sensors (Table 2) in the railway that can detect the train: SEA, SEB and SEC as shown in the figure 5. There are also sensors which detect when the barrier is down (BD) and up (BU).

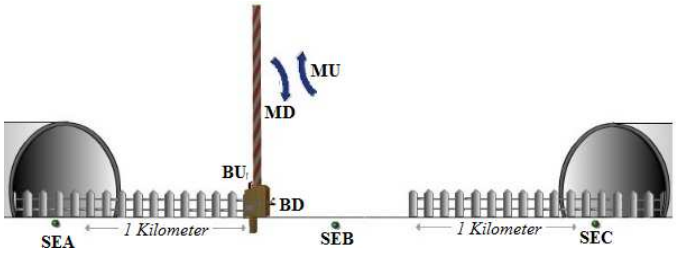

Fig. 5 – Railroad gate scheme [18].

B. *Input and output variables*

The first step to solve the problem is to define the receptivities (inputs) and actions (outputs) that will be used in the programs. In Tables 2 and 3 are represented the descriptions, words and bits given to the input and output variables, respectively.

Table 2 – Descriptions, words and bits of all receptivities.

Receptivity	Description	Word & bit
SEA	Train over sensor A	0.00
SEB	Train over sensor B	0.01
SEC	Train over sensor C	0.02
BD	Barrier down	0.03
BU	Barrier up	0.04

Table 3 – Descriptions, words and Bits of all actions.

Action	Description	Word & bit
MU	Barrier moves upwards	100.00
MD	Barrier moves downwards	100.01

C. *Controller specification*

As said before, for each plant model, there can be developed different, but all functional, solutions.

Students can develop and test different controller specifications. The SFC presented in figure 6 is a possible controller specification applied to this exercise.

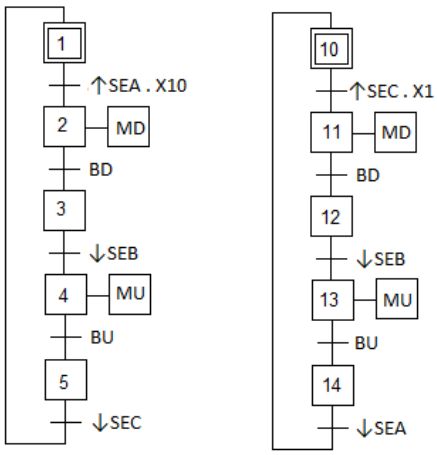

Fig. 6 - One possible controller specification to the Railroad gate exercise.

The methodology proposed in the last chapter is followed.

D. *Plant modules models*

The first modular model created was the barrier's motor model. The Motor can rotate in two different ways (one to move the barrier upwards and another one to move barrier downwards). Having this in mind, the model has three different possible locations, as shown in figure 7. The description of the variables is illustrated in tables 2 and 3.

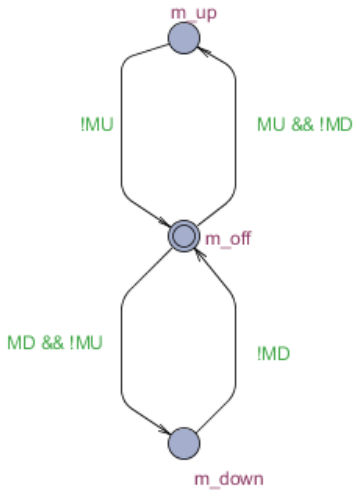

Fig. 7 – Barrier's motor model

The initial state of the Motor's model is *m_off*, which corresponds to the Motor being turned off. If MD or MU orders exist (or not), a guard will be beaten and the model will change its state to *m_down* or *m_up*, respectively. To get back to *m_off* again, it only needs to beat the guards for this purpose.

The second model that was designed was the Barrier's model. It was defined that each location would represent a barrier's different position, during the movement. The barrier is raised when it is in its initial position and it is dependent on the motor's model evolution. If motor's model is in *m*_down location, barrier will move downwards. On the other hand, if motor's model is in *m_up* location, barrier will move upwards, as shown in figure 8.

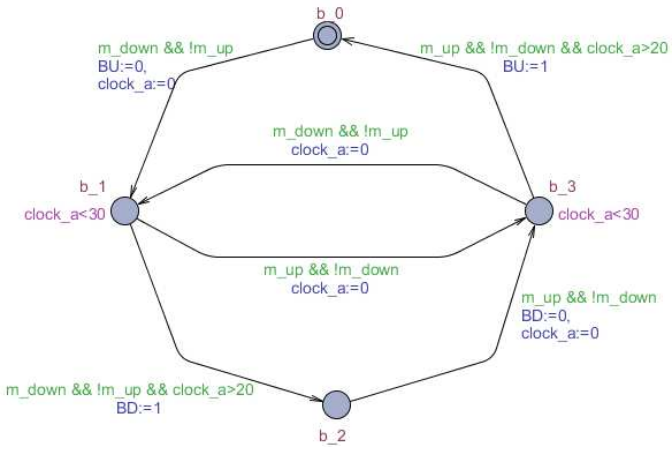

Fig. 8 – Barrier's model using timed finite automata.

The developed model for the train movement is presented in Figure 9. State *p0* represents that there is no train on the railway. All the other locations represent train's position along the railway. It is a very restricted model because train's movement cannot be controlled by the controller; it is modelled by an observer automata. As the train can "travel" in two different directions, the model's states are divided in

two groups: the left side of Figure 9 corresponds to the train coming from "left to right" (the states, represented with an "*e*" at the end of the name, will turn on); when the train comes from "right to left" (right side of Figure 9), the states with a "*d*" at the end will be active. It is an observer model with seven positions in each direction to represent the most important points in train's movement.

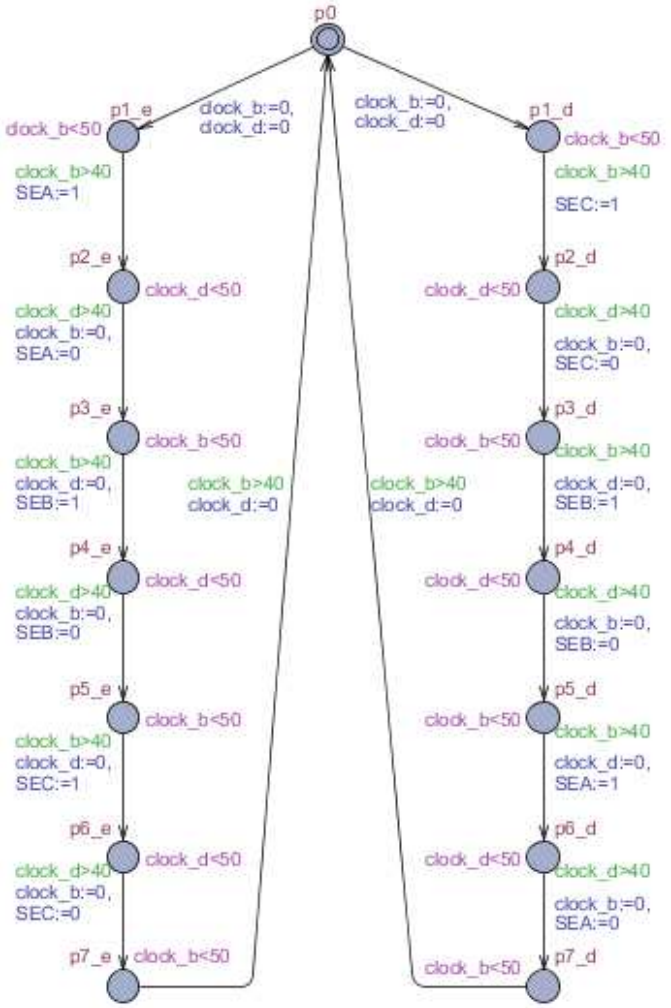

Fig. 9 – Train's model using timed finite automata.

When defining train's model it was assumed that the train moves in one direction at a time. This other restriction corresponds to a real world event as, if two trains move in opposite directions in the same line and at the same time, it would result in a catastrophic accident. Taking advantage of timed finite automata characteristics, the model was designed to choose the train's direction randomly. It could be also used a random event to the same function.

Random events are very useful in operative part's models. They can be used to simulate arbitrary processes close to reality. There are different ways of generating a random event. Figure 10 shows a random event representation using timed finite automata.

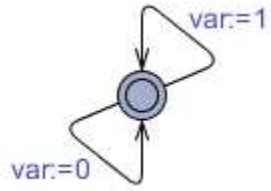

Fig. 10 – Example of a random event generator using timed finite automata.

E. Simulation

The SFC and the plant models were both converted into Ladder language to perform the simulation [15]. Both parts were written in the same file of CX-Programmer in different memory areas, allowing messages/information exchange. In the context of the translation of the model to Ladder, invariants do not exist and for each transition, the clocks defined in TFA must be converted into different timers.

Controller and plant programs were written in CX-Programmer (Figure 12a)) and a schematic representation of the problem was designed in CX-Designer, Figure 12b)). This functionality is considered relevant in pedagogical terms as it allows monitoring in real-time the evolution of the CX-Programmer program. It allows to easily detecting programming errors, or mistakes, and motivating students in the learning process.

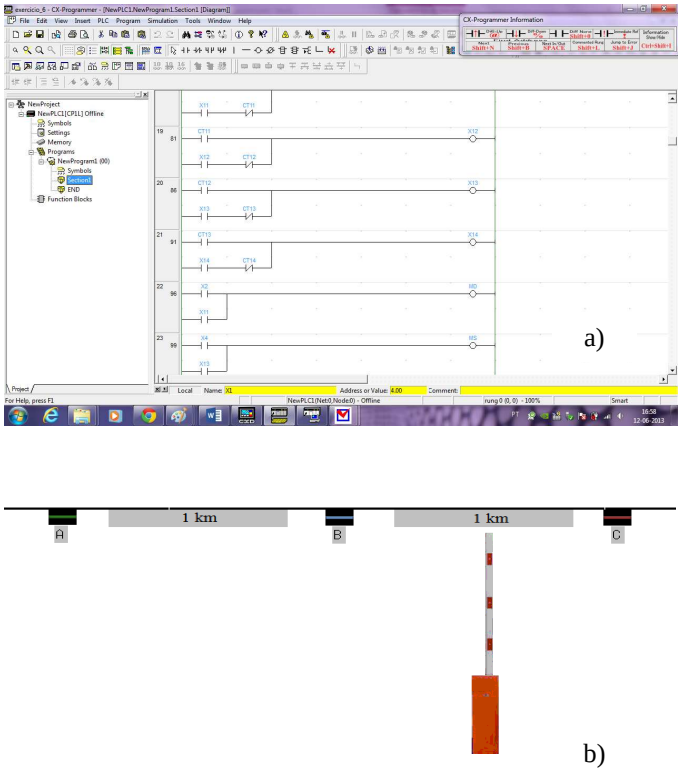

Fig. 12 – CX-Programmer (a) and CX-Designer (b) environments.

Running the simulation with CX-Simulator, trains move in CX-Designer environment. This movement is enabled by the controller and plant parts. It is important to notice that

command part sends orders to the operative part, while this sends back sensors signals. Figures 13 and 14 show two screenshots of the train's simulation.

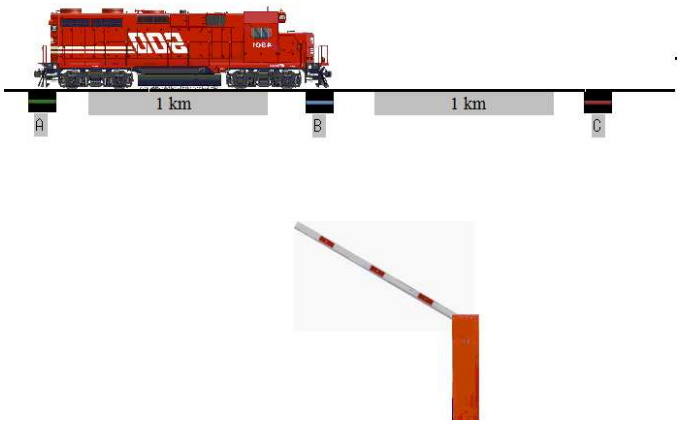

Fig. 13 – Simulation of train coming left to right.

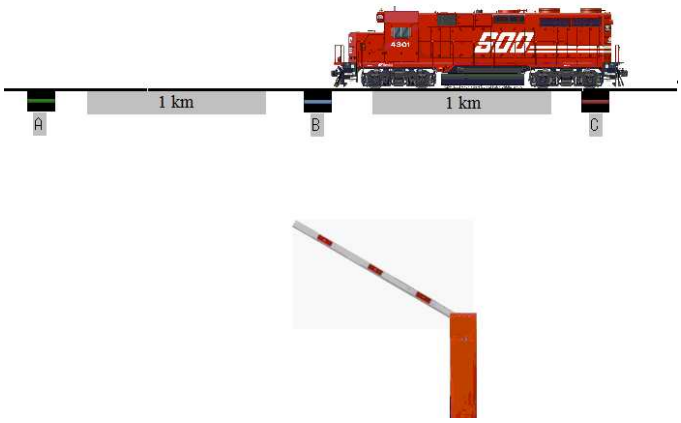

Fig. 14 – Simulation of train coming right to left.

V. CONCLUSION

The objective of this work was to develop a pedagogical tool where students can test the behaviour of automation systems corresponding to real-world case-studies.

The developed methodology included the design of plant models with timed finite Automata and several command programs using SFC formalism to test them. It was used OMROM's CX-One software to simulate the exercises.

This tool allows virtual simulation of automation problems, considered an advantage to students to have a practical approach to what is taught in class.

The platform was tested in a closed environment and next step is to provide the platform to automation students to have some feedback how this experience is helpful to them in their teaching/learning method of automation subject.

VI. REFERENCES

[1] N. Carvalho, R. Silveira, C. Leão, J. Machado and F. Soares, "Platform WALC: design and development of a PLC network," December 10-11, 2009, Bratislava, Slovak Republic, E-Academia Slovaca, N.O. ISBN: 978-80-89316-11-3.

[2] E. Matias, P. Oliveira, J. Cunha, E. Pires and F. Soares, "E-GRAFCET: A MULTIMEDIA EDUCATIONAL TEACHING TOOL," Controlo 2010, 9th Portuguese Conference on Automatic Control, Coimbra, Portugal, 8-10 September, 2010.

[3] E. Hansen,"The role of interactive video technology in higher education: Case study and proposed framework", Education Technology, September 1990, 13-21.

[4] G. Carnevali and G. Buttazo, "A Virtual Laboratory Environment for Real-time Experiments", Proceedings of the 5th IFAC International, Symposium on Intelligent Components and Instruments for Control Applications (SICICA 2003), Aveiro, Portugal, July 9-11, pp. 39-44,

2003.

[5] European Standard EN 60848: GRAFCET specification language for sequential function charts, 2002.

[6] IEC Standard 61 131-3: Programmable Controllers - Part 3, 1993.

[7] Machado, José M; Denis, Bruno; Lesage, Jean-Jacques. 2006. "A generic approach to build plant models for DES verification purposes", In Proceedings of the Workshop on Discrete Event Systems (WODES'2006), Ann Arbor, USA, 2006.

[8] Kunz, Guilherme; Perondi, Eduardo; Machado, Jose. 2011. "Modeling and simulating the controller behavior of an Automated People Mover using IEC 61850 communication requirements", In 2011 9th IEEE International Conference on Industrial Informatics, Lisbon, Portugal.
doi: 10.1109/INDIN.2011.6034947

[9] J. Machado, E. Seabra, J.C. Campos, F. Soares and C. P. Leão, "Safe controllers design for industrial automation systems". Computers & Industrial Engineering (2011), doi:10.1016/j.cie.2010.12.020

[10] IEC - International Electrotechnical Commission (2000), "Langue de spécification GRAFCET pour diagrammes fonctionnels en sequence", 2 ed., CEI 60848.

[11] http://en.wikipedia.org/wiki/Timed_automaton (accessed on June 2013).

[12] P. Carneiro, "Desenvolvimento de protótipos virtuais para a utilização em simulação *Software-in-the-loop*," Master's Thesis in Mechanical Enginneering, Universidade do Minho, 2012.

[13] G. Behrmann, A. David and K.G. Larsen, "A tutorial on UPPAAL". In Proceedings of the 4th international school on formal methods for the design of computer, communication, and software systems (SFM-RT'04). LNCS 3185, 2004.

[14] Nanette Bauer, Sebastian Engell, Ralf Huuck, Sven Lohmann, Ben Lukoschus, Manuel Remelhe, Olaf Stursberg. "Verification of PLC Programs Given as Sequential Function Charts". Integration of Software Specification Techniques for Applications in Engineering Lecture Notes in Computer Science Volume 3147, pp 517-540, 2004.

[15] Murat Uzam. "A general technique for the PLC-Based implementation of RW supervisors with time delay functions". Int J Adv Manuf Technol (2012) 62:687-704. DOI 10.1007/s00170-011-3817-1

[16] http://www.omron.com/ (accessed on June 2013).

[17] Johan Blom , Anders Hessel , Bengt Jonsson , Paul Pettersson. "Specifying and Generating Test Cases Using Observer Automata". Proc. 4 th International Workshop on Formal Approaches to Testing of Software 2004 (FATES'04), volume 3395 of Lecture Notes in Computer Science, 2005.

[18] http://dei-s1.dei.uminho.pt/lic/AUT/animacoes_page.htm (accessed in June 2013).