Evaluation of the uModel Factory Software Used For the Modeling of Embedded Systems with Concurrent States

M. Prieto, L. Sugezky, N. González, M. Giura, Y. Kuo, M. Trujillo, J.M. Cruz

Electronics Engineering Department, Univ. Tecnológica Nacional, Fac. Regional Bs. As., City of Buenos Aires, Argentina

mprietocanalejo@frba.utn.edu.ar

***Abstract*—uModel Factory, a didactic-professional application modeling software for embedded systems, was developed as part of the Research and Development project of UTN.BA's Electronics Engineering Department (PID UTN1562, Resolution CD FRBA No. 2040/11). This software enables the creation of a state chart through its graphic interface and the automatic generation of portable C code and all its associated documentation while keeping synchronicity between the model, the generated code and the documentation at all times. It further enables simulation, validation and formal verification of the state charts designed. This work presents the design of modeling and implementation tools for systems with concurrent tasks with the aim to integrate them into the new version of the modeling software.**

***Keywords*—UML; embedded systems; model; concurrency**

I. Introduction

The implementation of embedded systems is posing an increasingly complex challenge given the evolution of today's electronic and information systems. Modeling permits to simulate the operation of a system from the first stages of design and to automatically generate the source code and the documentation associated to the project, keeping synchronicity between model, code and documentation at all times. This work focused on the development of a PC software application that may serve as a tool for modeling and simulating embedded systems as well as generating the C code of the model and its associated documentation as a result.

The modeling tool used was the FSM (Finite State Machine), a graphic tool that has been traditionally used for modeling the behavior of electronic and information systems. A state machine is a model that describes a system whose behavior depends on current and past events. At each time instant, the machine is in a single state and, depending on current or past inputs and their order of arrival, it will or will not change its state, performing actions that interact with the environment. These diagrams are usually described by means of: states (Figure 1.a), transitions (Figure 1.b), events (Figure 1.c), actions (Figure 1.d) and activities [1].

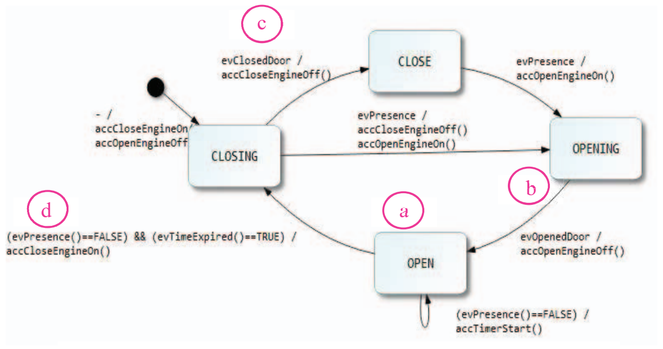


Figure 1. State diagram elements

FMSs constitute an extremely useful tool and they have been widely used in embedded system modeling. However, part of their efficacy is lost when the system to be modeled includes parallel activities which may or may not interact among themselves and which trigger a multiplicity of actions, that is to say, when the system becomes more complex. For this reason, David Harel [2,3] proposed an extension to the formal conventionalism of these machines and called them Statecharts. These charts extend FMSs mainly through hierarchy, orthogonality and communication elements [4]. Hierarchy makes it possible to differentiate between the various levels of dependency; orthogonality, also referred to as concurrency or parallelism, allows coexistence of several tasks which have little relation among themselves or which are independent; communication enables the performance of several actions given the same event and allows several events to trigger an action and to send messages to a task.

This work presents the implementation, development and evaluation of a new version of the uModel Factory software, which includes tools for the modeling of embedded systems with concurrent tasks. In order to evaluate the results, the microprocessor LPC1769 was programed with the C code generated by the software as well as with code for managing peripheral devices. After the microprocessor was programed, the system was tested on the Infotronik kit [5], thereby completing the embedded system implementation process.

978-1-5090-5859-4/17/$31.00 ©2017 IEEE

II. Objectives

The 2015 version of uModel Factory (v2.0) permits to generate the model of a simple system with the handling of multiple events and actions, allocation of variables to actions or events, and association of actions at reset. Once the model is designed, the software can be used to simulate the system to verify its correct operation and finally, to generate the C code and the documentation.

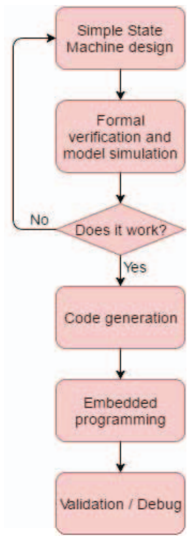


Figure 2. Block diagram

As it can be observed in Figure 2, with this v2.0 it was not possible to design a model for a system requiring implementation of parallel state machines. When this requirement existed, two separate projects were implemented for each of which a state machine was designed and then integrated without the tool. This implied extensive embedding work since both developments might have actions or events in common or require signals for communication between them. Figure 3 shows that the development of concurrent states may share elements, thus not enabling the development of separate state machines.

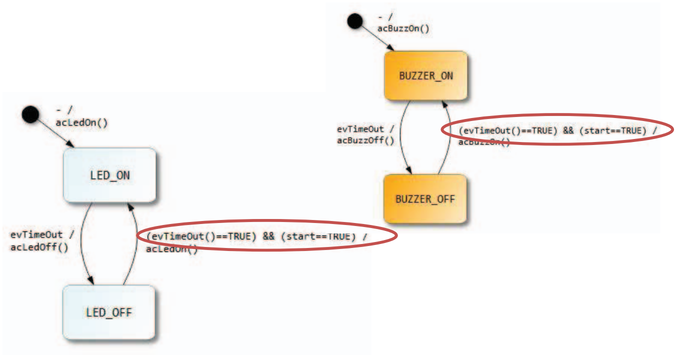


Figure 3. Concurrent state machines

To continue the work related to the development of this embedded system modeling software, the objective set for the new v3.0 version was the incorporation of another element of the statecharts, i.e. orthogonality, as well as other improvements resulting from the work done by the educational community that participated [6]. In order to achieve these goals, the new version includes:

- The possibility of generating more than one independent state machine in the same project.
- The simultaneous visualization of the machines, with the possibility to choose which one to visualize and in which position.
- The generation of code and documentation of the state machines running in parallel.

Although there are other tools that make possible code generation based on modeling in different languages [7, 8], most of them have a marked learning curve, paid license or they are not directed to embedded systems. This is the reason why this tool has a didactic approach since it generates a C language code in different implementations, which helps classroom analysis and discussion. This software has been used as a didactic tool in Informática II course throughout the years 2014, 2015 and 2016. [6]

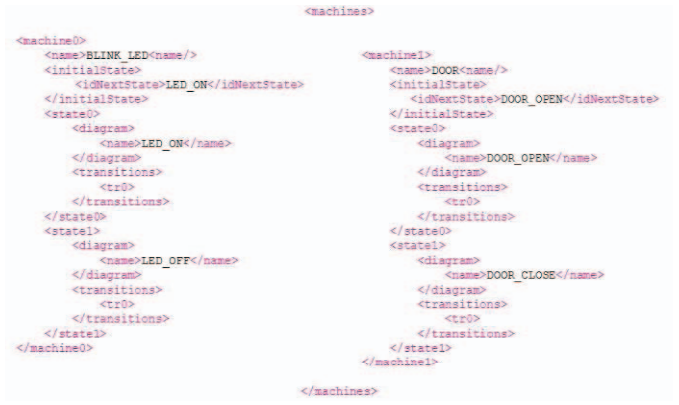
This new v3.0 version enhances the didactic aspects of this tool and includes more content related to the courses dealing with embedded systems.

III. Development

To achieve the proposed goal, the above-mentioned items were implemented, each of them substantially contributing to the new version (v3.0).

A. *Incorporation of multiple state machines*

With the aim of incorporating multiple state machines, the XML structure was studied in order to maintain the compatibility with v2.0 and to enable scalability of the design for future versions.



```
<machines>

<machine0>
    <name>BLINK_LED<name/>
    <initialState>
        <idNextState>LED_ON</idNextState>
    </initialState>
    <state0>
        <diagram>
            <name>LED_ON</name>
        </diagram>
        <transitions>
            <tr0>
        </transitions>
    </state0>
    <state1>
        <diagram>
            <name>LED_OFF</name>
        </diagram>
        <transitions>
            <tr0>
        </transitions>
    </state1>
</machine0>

<machine1>
    <name>DOOR<name/>
    <initialState>
        <idNextState>DOOR_OPEN</idNextState>
    </initialState>
    <state0>
        <diagram>
            <name>DOOR_OPEN</name>
        </diagram>
        <transitions>
            <tr0>
        </transitions>
    </state0>
    <state1>
        <diagram>
            <name>DOOR_CLOSE</name>
        </diagram>
        <transitions>
            <tr0>
        </transitions>
    </state1>
</machine1>

</machines>
```

Figure 4. XML v3.0 (this image shows the reduced XML)

Figure 4 shows the XML final design chosen for this version. From the labels corresponding to the state machines, it can be observed that a multiplicity of machines can be incorporated, and if only one is used, the compatibility with the previous version is maintained. The improvement of the

XML structure included other changes for the solution of problems found in v2.0.

A. *Workspace visualization*

The workplace permits to create a new state machine using the options menu (Figure 5). The state machines that are incorporated to the project as well as the events, actions and activities that are associated to them are visualized in priority on the screen.

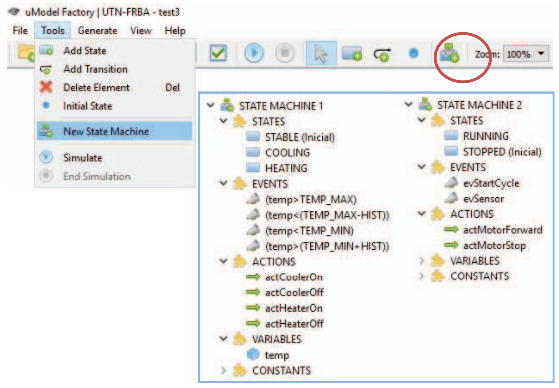


Figure 5. uModel Factory v3.0 workspace

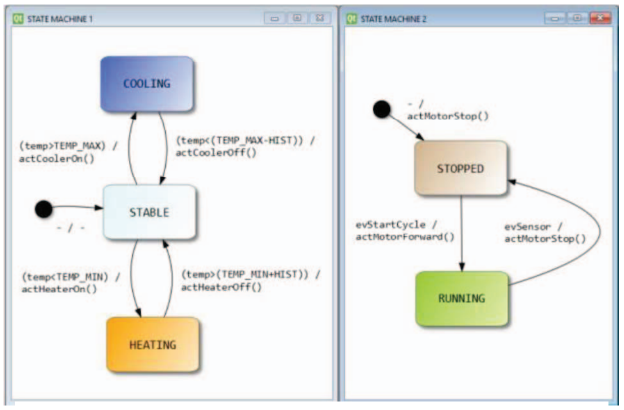

STATE MACHINE 1
COOLING
(temp>TEMP_MAX) / actCoolerOn()
(temp<(TEMP_MAX-HIST)) / actCoolerOff()
STABLE
- / -
(temp<TEMP_MIN) / actHeaterOn()
(temp>(TEMP_MIN+HIST)) / actHeaterOff()
HEATING
STATE MACHINE 2
- / actMotorStop()
STOPPED
evStartCycle / actMotorForward()
evSensor / actMotorStop()
RUNNING

Figure 6. Visualization of state machines in parallel

Figure 6 shows that it is possible to select which machines can be visible and to adjust the size in order to zoom in on the machine on which work is being done without losing sight of the other diagrams being used.

B. *Coding and Documentation*

Given the fact that the uModel Factory must automatically generate portable code C and all the associated documentation, keeping synchronicity between model, code and documentation at all times, the design with concurrent states was included. To that end, the implemented algorithm must analyze the redefinition of events and actions and generate the synchronization through signals which were implemented with global variables.

C. *Final Implementation of v3.0*

Finally, a new version installer was generated and it is now available for use in the subject Informática II of the UTN.BA Electronics Engineering degree program as well as by the industrial and education community.

The User's manual was also generated to facilitate the software utilization. The manual contains the following information:

- Instructions on how to create or open a project.
- Instructions on how to use the workspace and associated commands.
- Instructions on how to formally verify and simulate the model.

IV. RESULTS

For the uModel Factory v3.0 evaluation, a model with two state machines working in parallel was implemented in order to perform tasks simultaneously. The model responds to the characteristics described below:

A. *System used*

It consists of a system to control speed levels in public transport, which shares events and requires synchronization by means of a signal. That made it possible to test the new components of the software version under evaluation. Its features are:

- It measures speed indicating if it is: *LOW*, *MEDIUM*, *HIGH*.
- It allows to turn the system on or off.
- It keeps a led blinking at different rates depending on speed level.

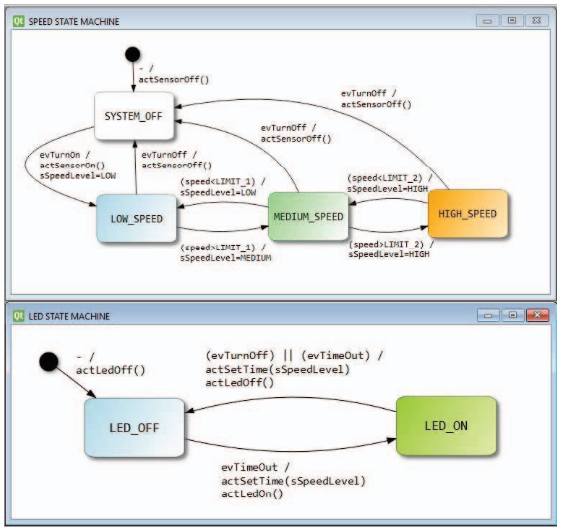

SPEED STATE MACHINE
- / actSensorOff()
SYSTEM_OFF
evTurnOff / actSensorOff()
evTurnOff / actSensorOff()
evTurnOn / actSensorOn() sSpeedLevel=LOW
evTurnOff / actSensorOff()
(speed<LIMIT_1) / sSpeedLevel=LOW
(speed<LIMIT_2) / sSpeedLevel=HIGH
LOW_SPEED
MEDIUM_SPEED
HIGH_SPEED
(speed>LIMIT_1) / sSpeedLevel=MEDIUM
(speed>LIMIT_2) / sSpeedLevel=HIGH
LED STATE MACHINE
- / actLedOff()
(evTurnOff) || (evTimeOut) / actSetTime(sSpeedLevel) actLedOff()
LED_OFF
LED_ON
evTimeOut / actSetTime(sSpeedLevel) actLedOn()

Figure 7. State machines of the evaluation system

As it is observed in Figure 7, the states of the system can be grouped together by concurrency in two state machines. Each of them has the events and transitions described in Table I and they are synchronized through a signal implemented by means of a global variable.

TABLE I. SYSTEMS FOR EVALUATION

State Machines	**Actions and Events**		**Synchronization variables**
	Events	***Actions***	
Led blinking	evTimeOut evTurnOff evTurnOn	acLedOn acLedOff	Led Blinking
Speed Sensing	evTurnOff evTurnOn speed > LIMIT1 speed > LIMIT2 speed < LIMIT1 speed < LIMIT2	acSetSSpeedLevel acSensorOn acSensorOff	Speed Sensing

Table I shows that these tasks that will work in parallel share the events **evTurnOff**, **evTurnOn** and to synchronize both tasks, signal **sSpeedLevel** is used.

B. *Results*

To analyze the results, the code was compiled and linked and the microprocessor LPC1769 was programmed. To that end, the C code generated by the software with the IDE LPCXpresso, which has a cross compilation for the microprocessor LPC1769, was used. In addition, the functions for the management of the specific peripherals to be used were added. The Infotronik kit was used to test the embedded system. It has peripherals that serve as input and output of the system to test the passage through all the states of both tasks that are working in parallel.

Finally, the following results were obtained:

- The compilation and linking through the gcc (version 5.2.1) for microprocessor lpc17xx without error or warnings.
- The correct state transition according to the different events of the system.
- The correct synchronization of both state machines through the generated signal.

V. FUTURE WORK

The next step of the project will consist of the incorporation of timed events to the design. To that end, all the necessary elements must be implemented in order to associate events and actions with a certain duration or time.

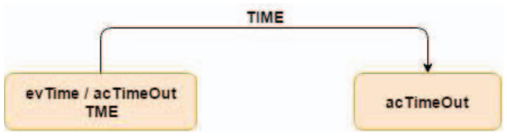


Figure 8. Timed events Diagram

When a timed event is launched, it will be associated with an action to be performed when timeout occurs and with the time that must elapse until timeout. In the software, a timer function must be implemented to measure the time and execute the associated action when timeout occurs (Figure 8).

This future implementation will permit to include the applications that require timing and to simulate them.

VI. CONCLUSIONS

In order to fulfill the objectives proposed in this work, software tools were designed and implemented for the modeling of embedded systems with concurrent tasks which were incorporated into a new version of the uModel Factory.

During the development, an extensive analysis of the XML structure and the coding to be implemented is conducted to consider the importance of compatibility with previous versions and the addition of new features in the future.

Results of the generated C code were evaluated through the programed microprocessor LPC1769, obtaining as a result the correct compilation, linking and execution in the implementation of an embedded system with concurrent tasks that had shared events and synchronization signals.

This work produced a new v3.0 version to which the user's manual was added. This version serves to strengthen the didactic aspects of the tool and will be available to the education community in order to widen its use.

REFERENCES

[1] C. Larman. "UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado". *Prentice-Hall*, 2003.

[2] David Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*. Vol 8, Nº 3, pp 231-274, 1987.

[3] David Harel (September 2016), Available at: http://www.wisdom.weizmann.ac.il/~dharel/

[4] Mariano Barrón Ruiz, "Desarrollo de Software Basado en Modelos para Sistemas Embebidos", *IEEE-RITA,* Vol. 4, N° 1, 2009

[5] Infotronik manuals (September 2016), Informatica II Web page, available at:

http://www.campusvirtual.frba.utn.edu.ar/especialidad/course/view.php?id=13

[6] N. González, L. Sugezky, "Evaluación del software uModel Factory como herramienta didáctica", *Congreso bienal de IEEE Argentina 3era edición,* 2016

[7] Visual Paradigm (December 2016). Available at: https://www.visualparadigm.com/features/code-engineering/

[8] Altova UModel (December 2016). Available at: http://www.altova.com/es/umodel/umlcode-generation.html