ECD++ A DEVS based Real-Time Simulator for Embedded Systems

Mohammad Moallemi, J. Marcelo Gutierrez-Alcaraz, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, Canada.
moallemi@sce.carleton.ca, marcelo@sce.carleton.ca, gwainer@sce.carleton.ca

Keywords: Embedded CD++ (ECD++), Real-Time, Mindstorms, Robocart

Abstract

In this paper we will present an M&S-driven framework to develop embedded systems based on the DEVS (Discrete Event systems Specification) formalism. DEVS provides a formal foundation to M&S that proved to be successful in different complex systems. This approach combines the advantages of a simulation-based approach with the rigor of a formal methodology. Another advantage of using DEVS is that different existing techniques (Bond Graphs, Cellular Automata, Partial Differential Equations, Queuing models, etc.) have been successfully transformed into DEVS models. CD++ is a software environment that implements DEVS theory with extensions to support real-time model execution in embedded systems. CD++ was used as the base for our development, building on previous research focused on real-time applications with hardware-in-the-loop. Embedded CD++ (ECD++) has been developed based on this tool to accomplish this aim. A small robocart has been built and tested with ECD++. The robocart uses sonar and touch sensor to detect obstacles on its way. At the end, ECD++ program has been compiled for the target and run using telnet connection on the board.

1. INTRODUCTION

One particular use of modeling and simulations tools is in the development of embedded systems, usually these systems have time constraints in which case they are also called Real-Time Systems. Real-Time Systems must provide reliable outputs to external inputs within a time limit. Depending on the strictness of the time limit, the systems are usually separated in soft or hard real time systems [1]. Another characteristic of embedded systems, is that most of them are application specific, although with the increase in computational power from microprocessors this trend is somewhat changing; many of these systems also have a low electrical power constraint because they are deployed in environments where grid-electricity is not commonly available or it is scarce, i.e. inside cars, space ships or remote sensors and actuators.

Many development methods and techniques exist for the creation of embedded systems, with the common denominator that most of them are based on hardware and software that exceeds the computational power of the intended system to be developed. The most common developing system is given by a general-purpose computer, a general-purpose operative system, the target software, which often includes a simulator, and required hardware to communicate with the embedded platform.

For engineering in particular, Modeling and Simulation (M&S) of embedded systems is of utmost importance. For example, engineers and scientists make heavy use of simulation tools when a process is difficult to replicate (because of the cost involved, or if the environmental conditions for the experiment are difficult to replicate or the danger is too high) or when the simulation of a natural process is many times faster than the real process. By using different techniques for modeling, we can predict the behavior of simple or complicated phenomena with, most of the time, a high degree of certainty. For systems that interact with real data, the preferred method for modeling is the use of continuous differential equations. However, one layer higher in the interaction between systems and the real world we deal with a different nature of modeling and control which is usually easy to model using discrete event modeling methods.

2. BACKGROUND

DEVS [2] is an increasingly accepted framework for understanding and supporting the activities of modeling and simulation. DEVS is a sound formal framework based on generic dynamic systems, including well-defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. DEVS theory provides a rigorous methodology for representing models, and it does present an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and middleware. A real system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled).

A DEVS atomic model is formally defined by:

M = <X, Y, S, dint, dext, ?, ta>,

Where:
$X = \{(p,v) \mid p \in IPorts, v \in Xp\}$ is the set of input ports and values;
$Y = \{(p,v) \mid p \in OPorts, v \in Yp\}$ is the set of output ports and values;
S is the set of sequential states;
dint: S ® S is the internal state transition function;
dext: Q × X ® S is the external state transition function, where:
$Q = \{(s,e) \mid s \in S, 0 < \in < ta(s)\}$ is the total state set, e is the time elapsed since the last state transition;
?: S ® Y is the output function;
ta: S ® $R^{+}_{0,¥}$ is the time advance function.

The semantics for this definition is given as follows. At any time, a DEVS coupled model is in a state $s \in S$. In the absence of external events, the model will stay in this state for the duration specified by ta(s). When the elapsed time e = ta(s), the state duration expires and the atomic model will send the output λ (s) and performs an internal transition to a new state specified by δint(s). Transitions that occur due to the expiration of ta(s) are called internal transitions. However, state transition can also happen due to arrival of an external event which will place the model into a new state specified by δext(s,e,x); where s is the current state, e is the elapsed time, and x is the input value. The time advance function ta(s) can take any real value from 0 to ∞. A state with ta(s) value of zero is called transient state, and on the other hand, if ta(s) is equal to ∞ the state is said to be passive, in which the system will remain in this state until receiving an external event.

A DEVS coupled model is composed of several atomic or coupled submodels, which is formally defined by:
CM = <X, Y, D, {Md | d∈D}, EIC, EOC, IC, Select>,
Where:
$X = \{(p,v) \mid p \in IPorts, v \in Xp\}$ is the set of input ports and values;
$Y = \{(p,v) \mid p \in OPorts, v \in Yp\}$ is the set of output ports and values;
D is the set of the component names, and the following requirements are imposed on the components, which must also be DEVS models:
For each $d \in D$, Md = (Xd, Yd, Sd, dint, dext, ?, ta) is a DEVS with
$Xd = \{(p,v) \mid p \in IPortsd, v \in Xp\}$,
and $Yd = \{(p,v) \mid p \in OPortsd, v \in Yp\}$.
The component couplings are subject to the following requirements:
External input coupling (EIC) connects external inputs to component inputs,
$EIC \subseteq \{((N, ipN), (d, ipd)) \mid ipN \in IPorts, d \in D, ipd \in IPortsd\}$;
External output coupling (EOC) connects component outputs to external outputs,
$EOC \subseteq \{((d, opd), (N, opN)) \mid opN \in OPorts, d \in D, opd \in OPortsd\}$;
Internal coupling (IC) connects component outputs to component inputs,
$IC \subseteq \{((a, opa), (b, ipb)) \mid a, b \in D, opa \in OPortsa, ipb \in IPortsb\}$;
Select: 2D - { } ? D is the tie-breaking function for imminent components.

Due to the closure property, a coupled model is regarded as a new DEVS model. This property clarifies that the overall behavior of a coupled model is equivalent to a basic atomic model, and therefore allows hierarchical model construction.

CD++ [3], [4] is a modeling tool that was defined using the DEVS and Cell-DEVS specifications. The toolkit includes facilities to build DEVS and Cell-DEVS models. DEVS Atomic models can be programmed and incorporated onto a class hierarchy programmed in C++. Coupled models can be defined using a built-in specification language. Cell-DEVS models are built following the formal specifications for DEVS models (informally presented in the previous section), and a built-in language is provided to describe them. CD++ makes use of the independence between modeling and simulation provided by DEVS, and different simulation engines have been defined for the platform.

CD++ is built as a class hierarchy of models related with simulation processing entities. DEVS Atomic models can be programmed and incorporated onto the Model basic class hierarchy using C++. Once an atomic model is defined, it can be combined with others into a multicomponent model using a specification language specially defined with this purpose.

3. EMBEDDED CD++

As mentioned before, CD++ is a software environment that implements DEVS theory. In this work we added an extension to support real-time model execution in embedded systems[5]. CD++ was used as the base for our development, building on previous research focused on real-time applications with hardware-in-the-loop. A robocart development technique, based on the creation of Experimental Frameworks showed success. We will discuss how to use this framework to incrementally develop embedded applications, and to seamlessly integrate simulation models with hardware components. Our approach does not impose any order in the deployment of the actual hardware components, providing flexibility to the overall process. The use of DEVS improves reliability (in terms of logical correctness and timing), enables model

reuse, and permits reducing development and testing times for the overall process. Consequently, the development cycle is shortened, its cost reduced, and quality and reliability of the final product is improved.

ECD++ is derived from the stand alone CD++. The difference between them is in time advance function. In CD++ time advance is not counted from a real time clock. As the events happen time advances in the simulation, thus simulation ends sooner than what is stated in the program. however in embedded CD++ time advance function gets its time value from the dock of the CPU, hence events are generated exactly the time that are specified in the event file or the time they happen. The other difference between embedded and stand alone CD++ is real event handling of embedded CD++. Since embedded CD++ is designed for specific embedded use and works with external target or hardware, it must handle external events that come from the external target. Hence there is an option for the programmer to use event file or use real time events that are coming from the sensors on the target.

Besides, four new features have been added to CD++ to support embedded functions.

3.1. Compile2Target

It allows the compilation of the software with the cross-compiler, with a similar methodology as the one used for the Standalone version, with some modifications to adapt the automated process to the ECD++ tool.

3.2. Download2Target

A new feature inside the plug-in that allows the downloading of the binary file to the Target platform by establishing a Network File System (NFS) mounted between Host and Target. Whenever NFS 'mount' is set up the Host downloads up to three files: the ECD++ simulation binary, the model file and the external event file if any or both are selected, when the copying of the files is finalized the NFS folder is 'unmounted'.

3.3. Run Simulation on Target

It allows the execution of the simulation remotely from the Host machine, with user selectable parameters, redirecting the display output of the Target machine to a non-interactive Console window in Eclipse.

3.4. Telnet2 Target

The last feature offers a way to establish a remote connection with the Target, which can be used to execute the simulation, to debug such simulation remotely by using standard Linux remote debugging tools or for maintenance purposes, i.e. to configure network parameters on the Target.

4. ROBOCART TEST

Lego Mindstorms NXT[6][6] is a programmable robotics kit released by Lego[7] in late July 2006. The main component in the kit is a brick-shaped computer called the NXT brick. It can take input from up to four sensors and control up to three motors, via RJ12 cables, very similar to but incompatible with RJ11 phone cords. Power is supplied by 6 AA (1.5 V each) batteries in the consumer version of the kit and by a Li-Ion rechargeable battery and charger in the educational version.

The kit includes three identical servo motors that have built-in reduction gear assemblies and can sense their rotations within one degree of accuracy. The kit also includes four sensors, each with a different capability. The touch sensor, the light sensor, the sound sensor, the ultrasonic sensor can measure distances and detect movement.

NXT++[8] is a C++ open source library that provides functions to control the sensors and motor via a USB or Bluetooth connection from a computer. It is available for Linux and Windows environment.

In this project a DEVS model has been defined for a robotcart. ECD++ has been installed on a Linux system (Fedora Core 3). NXT++ has been downloaded and installed and necessary code for integration with ECD++ has been added. An external board has been provided and Linux (Redhat 9) been installed on it. A Telnet connection has been established between the board and the computer. ECD++ codes have been compiled on the computer and then the executable simulation file, the model file and the event files were copied to the Target board. The simulation executable file was then run remotely through Telnet connection. Connection between the robocart and the board was via USB port.

4.1. DEVS Model

Formal specifications of this model are as follows:

Spin Motor = $<X, Y, S, ta, \delta int, \delta ext, \lambda>$

X: USB port external events: these are the changes in the state of four inputs that are decided on the sensor values: Spin_Motor_Clockwise, Spin_Motor_CounterClockwise, Turn_Left, and Turn _Right

Y: output port: is the USB port.

S: system states: Spin_Motor_Clockwise, Spin_Motor_Counter_Clockwise, Stop_Motor, Turn_Left and Turn_Right.

ta: time advance function: handled externally from the simulator.

δint: internal transition function: resume the forward movement of the robocart.

δext: external transition function: checks for changes on either one of the control bits.

λ: output function: write the action to output console.

4.2. ECD++ Implementation

This model has been programmed by ECD++. The ultrasonic sensor takes continuous distance measurement every 500 milliseconds from the obstacle in the front. The touch sensor sends a true value whenever it is pressed. In these two cases an external event is generated with the value of the distance to the obstacle or the value 1000 for touch sensor, when it is pressed. Based on the desired distance appropriate action is decided in the external transition function. At first the state of the robotcart is set to "Spin_Motor_Clockwise" which moves the robocart forward. Every time an external input is generated the value of the input is tested. If the value (distance) is less than 18cm, the state of the robotcart is changed to either "Turn_Left" or "Turn_Right" (alternatively) for a specific time which is calculated with the speed of the turn to accomplish a 90 degrees turn. After this time robocart goes to passive state. While the turn time is finished internal transition function is called. In the internal transition function the forward movement is resumed by changing the state of the robocart to "Spin_Motor_Clockwise". The touch sensor is located under the ultrasonic sensor and close to the floor, thus detects low height obstacles that block the movement of the robocart. If the value of the external input is 1000, it shows that touch sensor is pressed. In this case the state of the robocart is changed to "Spin_Motor_Counter_Clockwise" for the preparation time. In this time the robocart moves backward to avoid the obstacle. At the end when simulation ends, the state of the robocart is changed to "Stop_Motor". Figure 1 shows a photo of the robocart.

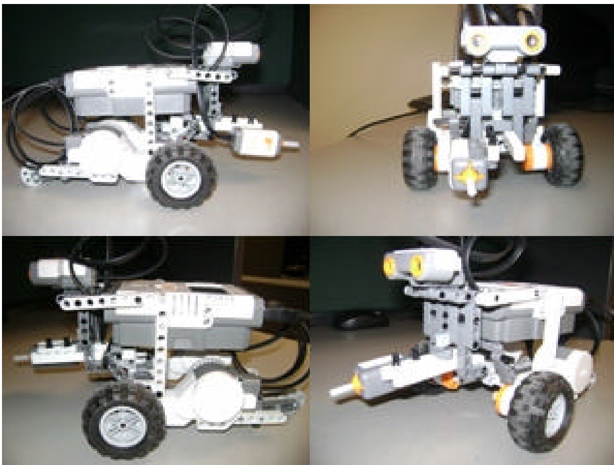

Figure 1- Robocart image

4.3. Test Results

After implementation, robocart has been tested with different obstacles with different height and degree of bias from the robocart. The test showed that obstacles that are not perpendicular to the ultrasonic sensor could not been detected properly. This problem can be overcome by using more than one ultrasonic sensor (e.g. three ultrasonic sensors). Short obstacles have been detected by touch sensor very well, but robocart got trapped in the corners which the walls are not perpendicular to the ultrasonic sensor and the touch sensor was not able to touch the corner. At the end a small maze has been provided and the robocart been tested in the maze, that it could find its way out of the maze.

5. CONCLUSIONS

In this paper, a DEVS based real-time simulator for embedded systems has been proposed. DEVS provides am M&S foundation that guarantees the validity of our simulation. CD++ simulator which is a stand alone simulator based on DEVS theory has been introduced and after that the modifications and new features that our work presented to CD++ have been proposed. At the end implementation of a robocart project using DEVS model and ECD++ toolkit has been proposed. DEVS model specifications of the robocart have been presented and the implementation of the model in ECD++ has been discussed. A small maze has been provided and the robocart being tested. The robocart was able to pass the maze without colliding to the walls. The result showed that ECD++ can be used in real time control systems. The model has been implemented in a way that can be extended to have more and different type of sensors or motors.

References

[1] Liu, Jane W.S. "*Real-time Systems*". Prentice Hall. Upper Saddle River, NJ. 2000.

[2] Zeigler, B.; Kim, T.; Praehofer, H. "*Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*". Academic Press. 2000.

[3] WAINER G., "CD++: a toolkit to define discrete-event models". Software, Practice and Experience. Vol. 32, No.3. pp. 1261-1306. November 2002.

[4] 56. Wainer, G. et al. "CD++ A tool for DEVS and Cell-DEVS Modeling and Simulation. User's Guide". Draft. August 2004.

[5] J. Marcelo Gutierrez-Alcaraz , " New Benchmarking and Embedded Extensions for CD++", M.A.Sc thesis submitted to Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada.

[6] Lego Mindstorms definition at wikipedia. http://en.wikipedia.org/wiki/Mindstorms_NXT.

[7] Lego Mindstorms NXT Kit. Website available at http://mindstorms.lego.com.

[8] NXT++ main page available at http://nxtpp.sourceforge.net/index.php.