Robot Control by Component-Oriented Interoperation of Real-Time DEVS Engines

Federico Bergero[2,4] bergero@cifasis-conicet.gov.ar

Mohammad Moallemi[1], Gabriel Wainer[1] {moallemi, gwainer}@sce.carleton.ca

Rodrigo Castro[3,4] rcastro@dc.uba.ar

1. Dept. of Systems and Computer Engineering, Carleton University, Canada.
2. CIFASIS-CONICET / Universidad Naciona de Rosario, Arngentina. 3. Computer Science Department, FCEN, UBA, Argentina.
4. Laboratorio de Sistemas Dinámicos FCEIA - UNR, Argentina.

Abstract

Model reuse and interoperability are cost and effort saving solutions for the simulation-driven development of embedded real-time systems. Different embedded systems share the same components (e.g. motors, sensors, actuators, controllers, etc), and remodeling them is costly in terms of time and effort. Instead, by combining different existing models, developers can improve productivity. We present a new generic lightweight interface for message transfers between DEVS models running on different DEVS-based tools. This allows the definition of component-based models to be deployed on different tools collaborating in real-time. The components work autonomously.

DEVS Modeling and Simulation

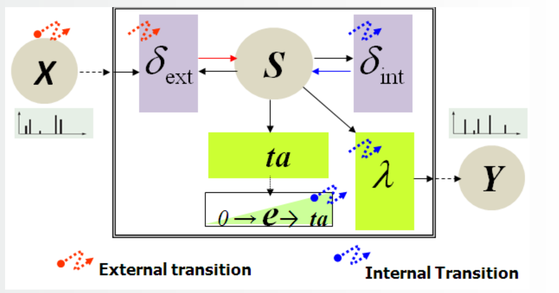
DEVS[6] is a general, system theoretic-based formal framework and mathematical language:

> Description of General Discrete Event Systems
> Well-defined coupling of components Hierarchical, modular construction
> Fosters models repository and models reuse
> Supports accurate approximation of Continuous Systems, Hybrid Systems and Generalized Stochastic Systems.

A **DEVS atomic component** is formally defined by:

AM = < X, S, Y, δext, δint, λ, ta >, where:

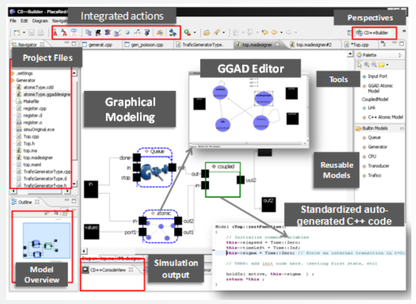
X: a set of external input event types
S: a sequential state set
Y: an output set
δext: Q × X → S, an external transition function
where Q is the total state set of M = {(s,e) | s in S and 0 ≤ e ≤ ta(s)}
δint: S → S, an internal transition function
λ: S → Y , an output functionta: S → R+0,∞, a time advance function



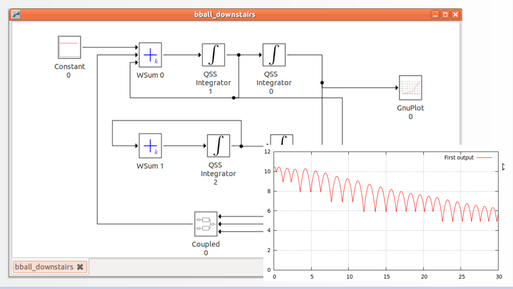
Real-Time DEVS Engines:

ECD++ and PowerDEVS

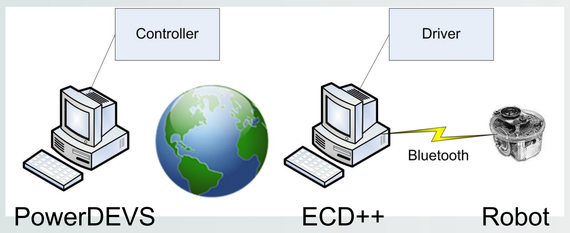
ECD++ [1,5,8] (Embedded CD++) is an engine that can execute DEVS models in embedded environments. It features a Flat Coordinator and a GGAD Graphical Modeling tool, supporting the Parallel DEVS formalism. Its real-time extensions allow users to develop Hardware-In-the-Loop applications with ease, being able to integrate them in DEVS-based environments.



PowerDEVS [2] is a general purpose software tool for DEVS modeling and simulation oriented to the simulation of hybrid systems. It allows defining atomic DEVS models in C++ language that can be then graphically coupled in hierarchical block diagrams to create more complex systems.
The environment automatically translates the graphically coupled models into a C++ code which executes the real-time simulation.



Collaborative Real-Time Simulation



The movitvation of this work is to introduce a **collaboration technique between discrete and continuous M&S-based systems under DEVS specifications**, in our case, PowerDEVS and ECD++ [4,7]. Our goal is to benefit from the formal and hierarchical features of DEVS to integrate the discrete models in ECD++ with continuous and hybrid ones in PowerDEVS[3].

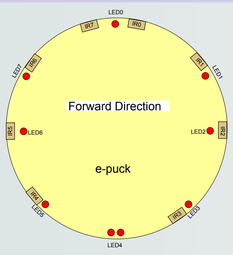
While the simulation engines are running in real-time, **different models can join this distributed network of running models** and feed from the outputs of other models while contributing their own outputs to the other models in the network.

The network interface for each DEVS port can be implemented in a different way (even using different network protocols), thanks to the abstract global message structure for transfer of the DEVS outputs.

This is a lightweight, decoupled, physically-based method to provide a unified notion of time advance across simulators. Thus, **the component-oriented aspect of DEVS allows different coupled components of a DEVS-based system to operate autonomously following a common physical notion of time advance.**

E-puck Robot:

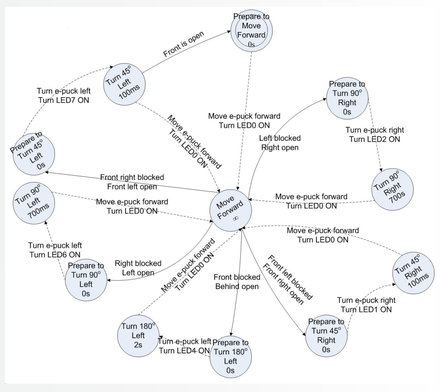
Obstacle Avoidance Model



E-puck [9] is a mobile robot equipped with sensors and motors. It is composed of eight infrared distance proximity sensors (IR), eight LEDs mounted on the top of the robot, and two motors.

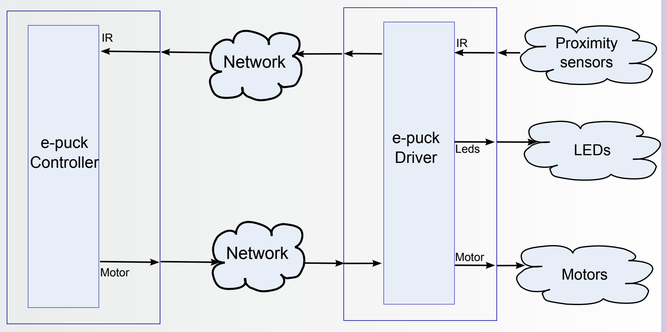
The controller model is designed with DEVS to steer the robot in a field while avoiding obstacles.

Based on the inputs received from the sensors, the controller takes the following different decisions: move forward, turn 45 degrees left, turn 45 degrees right, turn 90 degrees left, turn 90 degrees right, turn 180. The on the right shows a GGAD state machine model that implements the logical control of the e-puck taking decisions from what it receives from its IR sensors. This GGAD model can easily converted to a DEVS model and implemented in any DEVS simulator.



Partitioned and Networked Model

The e-puck logical controller is divided into two parts: the **Controller** and the **Driver**.
> The Controller is the main decision making unit, where the commands to avoid obstacles are generated.
> The Driver model works as a client who forwards the inputs from robot to the Controller and the outputs from Controller to the robot. The interface to the robot is part of the Driver model.



The Controller model is implemented on PowerDEVS and the Driver on ECD++. We use UDP network protocol for message transfer over an Ethernet network. We have chosen UDP over TCP for its simplicity, and since the experiments were done on a local network, the chances of loosing a UDP datagram were negligible. The messages carry no time reference and clock syncronization is not implemented. Time synchronization is implemented with the underlting operating system primitives.

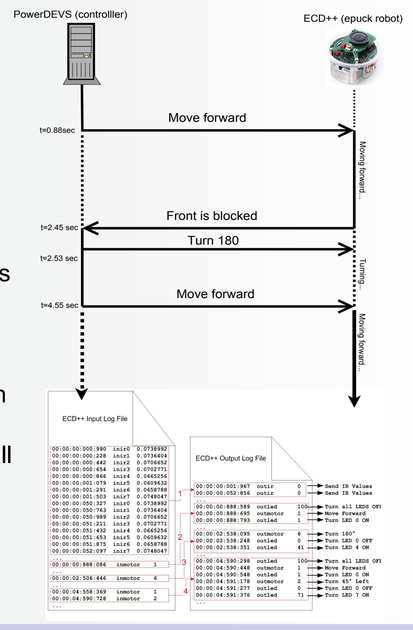
The payload of the UDP message contains an integer indicating to which port the message should go and a fixed-size buffer were sender and receiver have to agree on a format.

Experimental Results

We conducted various experiments implementing the example model presented.

The figure below shows the input and output log files of ECD++ simulator. The input log file records all the real-time incoming data (from the environment) to the model's input ports while the output file saves all the outputs of a DEVS model (with microseconds precision). The inputs and associated outputs are marked with red boxes in the figure. In the first box of the input file, two series of the IR sensor values inputted at time zero and after 50 milliseconds are shown (the IR sensor inputs are received every 50 milliseconds.)

The first box of the output file shows the output to the OutIR port, which triggers the output driver associated to this port to send the array of inputs containing the values of the eight IR sensors. Therefore, when all of the IR values are received, they are forwarded to the Controller.
Box 2 of the input file shows an input signal received from InMotor port containing value "1", which is interpreted in box 2 of the output file with the accompanying LED commands (added by the Driver).



Conclusions and References

> We introduced a generic lightweight interface for network I/O message transfers between DEVS models running on different DEVS-based tools.
> Splitting a model into components deployable to distributed real-time tools can be confined to an implementation layer, preserving all the original model specifications.
> The task of migrating subcomponents previously developed for ECD++ to PowerDEVS can be synthesized into a repeatable procedure.
> Messages across simulators do not bear time references into their semantics, leaving the synchronization responsibility to the modeling level.

[1] YU, J.; WAINER, G. "ECD++: a tool for modeling embedded applications".
[2] F. Bergero & E. Kofman. "PowerDEVS: A Tool for Hybrid System Modeling and Real-time simulation"
[3] Francois Cellier and Ernesto Kofman "Continuous System Simulation" Springer, New York, 2006.
[4] Moallemi, M..; Wainer, G. "Designing an Interface for Real-Time and Embedded DEVS"
[5] G. Wainer, et. al "A Model-Driven Technique for Development of Embedded Systems Based on the DEVS Formalism".
[6] B. Zeigler, T. Kim, H. Praehofer. "Theory of Modeling and Simulation". Academic Press 2000.
[7] Moallemi, M..; Wainer, G. "Designing an Interface for Real-Time and Embedded DEVS"
[8] Wainer, G. "CD++: a toolkit to define discrete-event models".
[9] E-puck robot website available at: http://www.e-puck.org/.