Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 5812–5817

Formal Framework for Discrete-Event Simulation

Vincent Albert, Clément Foucher

LAAS–CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France
(e-mail: {Vincent.Albert, Clement.Foucher}@laas.fr)

Abstract: A formal framework for modelling and simulation of parallel systems named ProjectDEVS is presented. The objective of this framework is to apply a Model-Based System Engineering approach to the development of simulation products for cyber-physical embedded systems. It is intended for the design and automated deployment of virtual prototypes. Models are constructed by coupling concurrent components exchanging data through ports and executed by various simulation schemes, namely simulators. This paper focuses on the integration of a Time Petri Net implementation of a parallel simulator into the framework. The semantics of the parallel simulator is formally described using timed transition system to verify the correctness of the implementation. Then, a model with its simulator can be model checked against formal specification and be rapidly deployed on FPGA or PC via code generators.

Keywords: Discrete-event simulation, Parallel simulator, Time Petri Net, Formal methods, MBSE

1. INTRODUCTION

A cyber-physical system is a system composed of computing processes (the controller) in interaction with physical processes (the plant) for control and command. During the development cycle of a controller/plant system, different simulation platforms are used for validation purposes. We believe that a Model-Based System Engineering (MBSE) approach allows, for the development of both embedded system and their simulation products, ensuring their reliability, promoting design and test of candidate architectures, mixing real or simulated components of the controller and/or plant, and of course reducing their development time. A MBSE approach typically relies on model transformations and code generators.

At a very early stage, virtual prototyping is used to study the performance of the system and design the control algorithms with a simulated plant. Virtual prototype does not require neither the controller nor the plant to operate in real time. However, it can be executed as a software, i.e. on a desktop simulator, or instantiated on a dedicated digital hardware processing unit or a mix of both. Dedicated digital hardware processing units can be used to accelerate various computations instead of using software. With the advent of Field-Programmable Gate Arrays (FPGAs), one can design a hardware circuit and instantiate it immediately instead of going through the long process of designing an Application-Specific Integrated Circuit (ASIC). Using FPGA devices, the creation of hardware accelerators dedicated to a specific or occasional purpose becomes possible.

Discrete event simulation is widely used for the validation of parallel and distributed systems. NS3 and Omnet++, which are reference tools in the field of computer networks and VHDL simulation in the field of digital hardware processing units, use this discrete event paradigm. In the field of continuous systems simulation, there is a family of asynchronous (event-driven) numerical integrators called QSS, see Cellier et al. (2006) for an overview of this method, that shows very good results. Those simulators all use the same scheme: it is the change of a variable (or signal) value which triggers what we call an event. An event can occur at any time. A variable is updated only when a specific event occurs at a discrete point of time. We call an event a time event when the considered variable is time, or a state event when it is a variable of the model. Whereas discrete-time simulators needs a special mechanism (zero-crossing functions) to handle state event for hybrid systems simulation, in discrete-event simulation, everything is time event because an event is scheduled and it is the time of the next event that makes simulation time advance.

In that context we develop a modelling and simulation tool of parallel systems named ProDEVS based on the Discrete Event System Specification (DEVS) formalism and its simulators. DEVS, formulated by Zeigler (1976), has an abstract syntax for atomic component which is nothing else than an interface (input/output) timed automata. It provides a modular and hierarchical construction of the model with the concept of coupled component that connects atomic component output ports to atomic component input ports. DEVS also defines a set of operational semantics (the way of executing the model), called the abstract simulator in the DEVS community, that can be seen as a Model of Computation (MoC), see Ptolemaeus (2014) for MoCs in Ptolemy. For instance we have the following Discrete-Event (DE) MoCs: Classic DEVS simulator (CDEVS) where components execution is sequential (only one component is executed at a time) and conservative Parallel DEVS simulator (PDEVS), formulated by Chow (1996), where several components can be executed at the same time but causality violations are strictly avoided. Various parallel and distributed simulation researchers have implemented parallel simulators for DEVS. A time warp optimistic DEVS simulator, see Jefferson et al. (1985), where causality might be violated, detected and remedied using roll-back has been implemented by Christensen (1990). A risk-free optimistic DEVS simulator, see Ferscha (1995), where events are assessed for risk before sending has been implemented by Reisinger et al. (1995). See Zeigler et al. (2000) for pseudo-code description of these abstract simulators.

This paper focuses on the integration of a Time Petri Net (TPN) implementation of the PDEVS simulator into ProDEVS. This work results in a formal MBSE framework we called ProjectDEVS which takes a ProDEVS model and its simulator,

2405-8963 © 2017, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2017.08.535

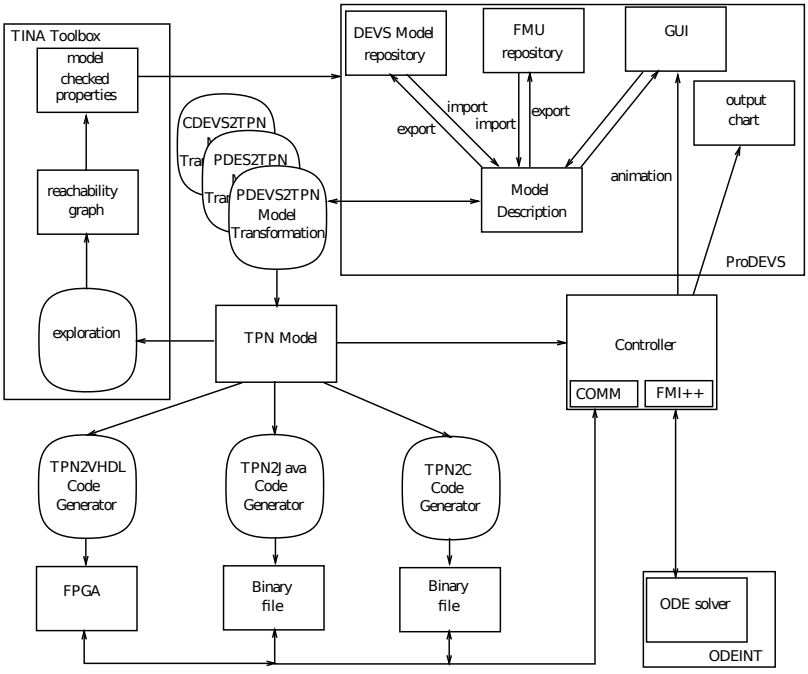

Fig. 1. ProjectDEVS Architecture

automatically transforms them into a TPN and deploy the latter as a program, as digital hardware or as a mix of both. Petri net is very efficient to describe parallelism and concurrency (resource sharing, synchronization) between tasks or processes. The advantages of using TPN as a backbone between a ProDEVS model and the platform dependant virtual prototype are : (1) the development of new simulators are not hand coded anymore, there are specified using temporal logic and designed using TPN, without any impact on the deployment phase, (2) a model can be checked against a formal specification to some extent (formal methods are subject to combinatory explosion) and we can ensure that the virtual prototype is correct, (3) formal verification can be coupled with simulation statically or dynamically (during run-time).

In the next section the architecture of the framework is detailed. Then, Section 3 defines the DEVS formalism, the principles of PDEVS simulator and the class of TPN we use. In Section 4, the rules for implementing a DEVS atomic component and a PDEVS simulator are given. Section 5 describes and illustrates the method we employ to verify the implementation and finally, perspectives and issues are given in Section 6.

2. ARCHITECTURE

The architecture of the ProjectDEVS framework is illustrated on figure 1. It includes ProDEVS, the model designer of ProjectDEVS, which includes a GUI offering a block-oriented view for model design. A model is constructed with concurrent components that can be imported from ProDEVS components repository or designed from scratch using input/output timed automata that we specially profiled for DEVS formalism, see Vu et al. (2015). User can create its own repository. Basically, the repository contains components for continuous systems, such as QSS integrators, or quantizers and switches for hybrid systems. We recently integrated FMI cosimulation and model exchange features, see MODELISAR (2014) for FMI specification, such that Fonctional Mockup Units (FMU) can also be imported in the model. We have developed a DEVS-FMI wrapper to synchronise discrete-time simulators with discrete-event simulators using FMI++. The FMI++ library is a utility package, implemented by Widl et al. (2013), that provides simulation functionalities for FMI model exchange and cosimulation specification. It includes a numerical integrator and a state record mechanism for roll-back. From a model description captured in the GUI and a given abstract simulator, a TPN model is generated. This TPN can be exported to the TINA toolbox, see Berthomieu et al. (2006), for model checking.

Then, a description of the structure of the Petri Net with dedicated components to implement places and transitions, and boolean or logical equations to represent enabling and firing conditions can be generated for simulation. This generated code is associated to a Time Manager which is in charge of simulation time events synchronisation and Action Managers to handle data computation in reaction of transition firing. A simulation clock provides events to make the internal TPN state evolve. Combined with a Run Manager aware of the given TPN structure, the prototype can be interfaced to a controller for simulation controls (run, step, break) and data visualisation. Data is recorded on a value change event, and stored along with the associated simulation time. A mapping between the ProDEVS model domain and the platform dependant variables is loaded into the controller for data charts. For software deployment, we have Java and C code generators that provide binary code which is interfaced with the controller. For hardware, a VHDL code is generated in a synthesizable form which can be used on a FPGA. The component representing the model is then wrapped in a register-based structure which can be adapted to most bus interfaces. Then, the simulator is interfaced with the controller through an Ethernet network using a small program running on a processor inside the FPGA. Interfaces for buses used by Xilinx and Altera, AXI and Avalon, are generated along with the model code.