DVML: DEVS-Based Visual Modeling Language for Hybrid Systems[*]

Hae Young Lee, Ingeol Chun, and Won-Tae Kim

CPS Research Team, ETRI
Daejeon 305-700 Republic of Korea
haelee@ieee.org

Abstract. This paper proposes a visual modeling language for a large-scale hybrid system based on discrete event system specification formalism. Within theproposed language, basic model diagrams define the hybrid behavior of systems, whilethe structure can be represented through coupled model diagrams. The language can visually define large-scale hybrid systems without a sacrifice of formal semantics, and does not require the users to have programming skills. A prototype of the modeling and simulation environment based on an extended version of the language has been implemented.

Keywords: Visual modeling language, discrete event system specification (DEVS), DEV&DESS,hybrid system modeling, cyber-physical systems.

1 Introduction

Discrete event system specification (DEVS) formalism [1], which is a theoretically well-grounded means of expressing modular discrete event simulation models, has gained increasing popularity in recent years [2]. Various DEVS-based modeling and simulation (M&S) environments [3,4] have been developed to tackle complex problems through a broad array of domains. However, in most of these environments, the users must write models in certain programming languages, such as Java and C++, with predefined libraries. This therefore results in their relatively limited adoption within the industry [5].

Recently, several DEVS-based graphical modeling approaches [5-7] have been proposed to make DEVS more accessible to a wider community. A DEVS graph [6], which permits the users to write DEVS models using state machines, was proposed by Christen et al., and was later adopted in graphical modeling editors of CD++Builder [5]. Traoré proposed a DEVS-driven modeling language [7] that considers functional, dynamical, and static aspects of systems in a graphical way. However, they have addressed only the graphical modeling of discrete systems, while most large-scale systems, such as cyber-physical systems (CPS) [8,9], involve hybrid processes combining both continuous and discrete phenomena.

[*] This work was supported by the IT R&D Program of MKE/KEIT [10035708, "The Development of CPS (Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"].

This paper proposes a visual modeling language (DVML) that enables graphical modeling of hybrid systems, based on discrete event and differential equation systems (DEV&DESS) formalism developed by Praehofer [10]. In DVML, a hybrid system can be defined using two different kinds of diagrams: a basic model diagram and coupled model diagram. The hybrid behavior of systems can be graphically defined using basic model diagrams, while coupled model diagrams can describe the structure of the systems. For hybrid systems modeling, by using DVML, the users are no longer required to have programming experience. A prototype of a DVML-based modeling and simulation environment is introduced at the end of this paper.

2 DEVS-Based Visual Modeling Language (DVML)

This section presents the proposed modeling language, DVML, in detail.

2.1 Basic Model Diagram (BMD)

DVML specifies a basic DEV&DESS model using a basic model diagram (BMD). A BMD is depicted as an empty rectangle with the name of the model at the top left. All elements that belong to a BMD are shown within the rectangle.

Each basic model interacts with its external environment by passing continuous values or discrete events through its I/O ports. Each port is represented by a small rectangle drawn on the border of the BMD, with the textual notation for the ports. The notation is defined by the following Backus-Naur Form (BNF) expression:

<*port*>::=‘[‘<*port-type*>’]’<*port-name*>‘:’<*data-type*>

<*port-type*>::=‘[C]’|‘[E]’

‘[C]’ is specified for continuous value ports. For a discrete event port, ‘[E]’ is specified and its value can be either a predefined event (an element of <*data-type*>) or∅ (nonevent). An input port and an output port are drawn as a small empty rectangle and a small filled rectangle, respectively.A discrete event port can be accessed only within transitions, whilecontinuous value ports can be accessed at any time.

In DVML, the specification of the behavior of a basic model is organized around the phase variable in an enumerated type of programming language. The phase variable denotes some type of representative state that the model remains in. In each phase, an enumerator in the variable is rendered as a rounded rectangle along with its name. The initial phase is depicted as a small filled circle.

Other state variables are denoted by text strings in the notation defined by the following BNF expression:

<*state-variable*>::=‘[‘<*variable-type*>’]’<*variable-name*>‘:’<*data-type*>

<*variable-type*>::=‘[C]’|‘[D]’

‘[C]’ is specified for continuous state variables, while‘[D]’ is specified for discrete state variables, which are updatedby discrete events.

A transition from a source phaseto a target phase is renderedas an arc that connects them, with a textual notation for transitions. The notation is defined by the following expression:

<transition>::=[*<transition-name>*][‘[‘*<transition-condition>*’]’][‘/’*<state-definition>*]]

<transition-condition> is a Boolean expression that specifies the firing condition of the transition. A transition is performed if and only if the model is in the source phase and if the firing condition is true. If the firing condition for a transition is not specified, the transition is immediately performed whenever the model is in the source phase. A <*state-definition*> is an expression that defines a new state and/or generates outputs. The expression is executed when the transition fires. The firing condition for the initial transition must not be specified so that the transition can be immediately fired. The initial values for state variables and continuous output ports must be specified within <*state-definition*> of the initial transition. Consequently, the model actually starts in the target phase of the transition with the initial values.

A constraint is an algebraic or ordinary differential equation (ODE) that defines continuous behavior or computes continuous states. If a constraint is within a phase (i.e., a phase constraint), the constraint is enabled only when the model is in that phase.

2.2 BMD Example

Figure 1 illustrates a sample BMD for a barrel filler described in [1]. The sample model has two input ports: *Inflow*, a continuous value input port (‘[C]’) in *double* (real) data type, and *On*, a discrete event input port (‘[E]’) in *bool* (Boolean) data type. The value of the port *On* is read only within the conditions of two transitions, *StartFilling* and *StopFilling*. Three output ports, *Level*, *Room*, and *Barrel*, belong to the model. Port *Barrel* is written using *Barrel* = *BarrelID* in order to generate an output event when transition *GenerateBarrel* fires. The model has three phases: *Ready*, *Filling*, and the initial phase. A continuous state variable (‘[C]’), *Capacity*, and a discrete (‘[D]’) state variable, *BarrelID*belong to the model. The former and latter variables store the capacity of a barrel and the identifier of the current barrel, respectively.

Continuous value output port *Room* is updated using the following algebraic equation:

$$Room = Capacity - Level \tag{1}$$

While the model in *Ready* phase is updated by the following ODE:

$$\partial Level/\,\partial t = 0 \tag{2}$$

That is, the current barrel is not filled. If the model is in *Ready* phase and receives an event true through *On*, transition *StartFilling* is fired. While the model is in *Filling*, *Level* is updated as follows:

$$\partial Level/\,\partial t = Inflow \tag{3}$$

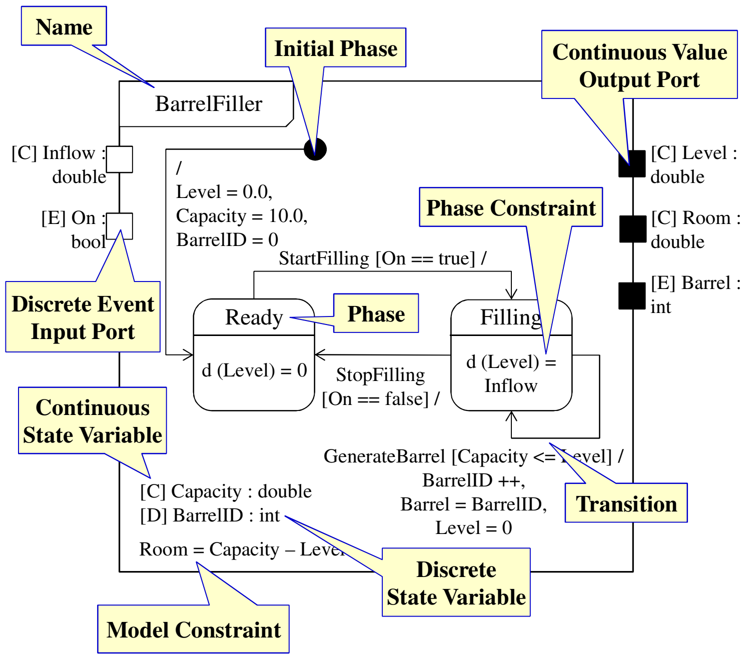

Fig. 1. Basic model diagram (BMD)

Thus, the current barrel is filled by *Inflow*. *GenerateBarrel* is fired if the model is in *Filling* phase and the value of *Level* is equal to or greater than the value of *Capacity* (i.e., if the current barrel is fully filled with content). Through the transition, the value of *BarrelID* increases in order to give the next barrel a new identifier, and the value of *Level* is set to 0 (an empty barrel). If the model is in *Filling* phase and receives a false through *On*, then *StopFilling* is fired.

2.3 Coupled Model Diagram (CMD)

A DEV&DESS coupled model is specified as a coupled model diagram (CMD) in DVML. Ports are drawn by small rectangles on the border with the textual notations. Each sub-model is depicted as a rectangle with its name. A coupling from a source port to a target port is rendered as an arc connecting them. When two ports are coupled, their types and data types must match.

2.4 CMD Example

Figure 2 shows a sample CMD for a barrel generator. The model consists of two sub-models, *BarrelFiller* (illustrated in Fig. 1) and Controller. A single external input coupling, three external output couplings, and two internal couplings are rendered as arcs.

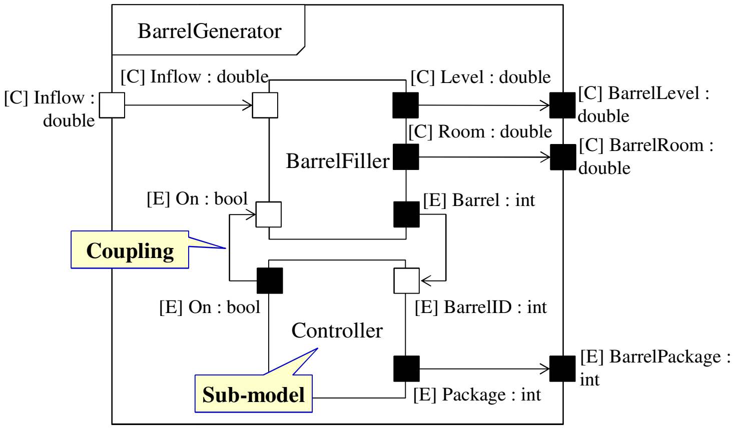

Fig. 2. Coupled model diagram (CMD)

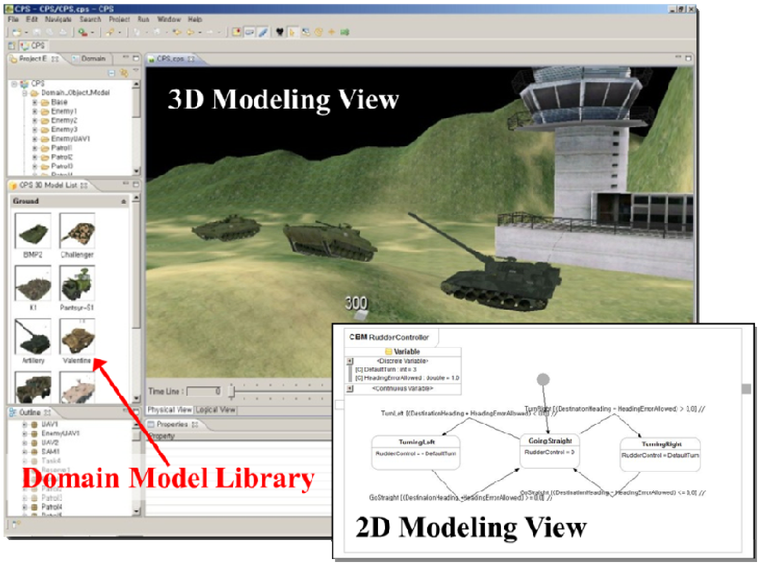

Fig. 3. A screenshot of a CPS domain modeler

3 DVML-Based Modeling Environment

We have implemented a prototype of a DVML-based CPS M&S environment. Figure 3 shows a screenshot of the modeling environment prototype, called *CPS domain modeler*, which is an Eclipse plug-in. Hybrid systems can be modeled with ease, by drag-and-dropping DVML elements (e.g., transitions, phases, and so on). That is, by using a CPS domain modeler, the users are not required to have programming experience in the modeling of hybrid systems. Models described in DVML are stored in a model base, called a *CPS domain model library*, together with 3-D objects. The modeler also provides a 3-D modeling environment. Within the modeler, a large-scale CPS can be easily composed by drag-and-dropping of 3-D objects embedding a hybrid behavior. CPS models can be executed on our DEV&DESS-based hybrid simulation environment, and their execution results can be reviewed by using a 3-D visualizer.

4 Conclusions and Future Work

In this paper, we proposed the formalism-based language for large-scale hybrid systems that can provide full graphical modeling capabilities without a sacrifice in formal semantics. A prototype based on DVML has been implemented for CPS. DVML is still being developed for a complex CPS. In the next version, some extensions for autonomic computing support will be included. We will also study the formal verification of DVML models.

References

1. Zeigler, B.P., Paraehofer, H., Kim, T.G.: Theory of Modeling and Simulation, 2nd edn. Academic Press (2000)
2. Feng, B., Wainer, G.: A NET Remoting-Based Distributed Simulation Approach for DEVS and Cell-DEVS Models. In: Proc. DS-RT 2008, pp. 292–299 (2008)
3. Adevs, `http://www.ornl.gov/~1qn/adevs/`
4. DEVSim++, `http://smslab.kaist.ac.kr/`
5. Bonaventura, M., Wainer, G., Castro, R.: Advanced IDE for Modeling and Simulation of Discrete Event Systems. In: Proc. SpringSim 2010 (2010)
6. Christen, G., Dobniewski, A., Wainer, G.: Modeling State-Based DEVS Models in CD++. In: Proc. ASTC 2004 (2004)
7. Traoré, M.K.: A Graphical Notation for DEVS. In: Proc. SpringSim 2009 (2009)
8. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: Proc. ISORC 2008, pp. 363–369 (2008)
9. Park, J., Yoo, J.: Hardware-Aware Rate Monotonic Scheduling Algorithm for Embedded Multimedia Systems. ETRI Journal 32(5), 657–664 (2010)
10. Praehofer, H.: System Theoretic Formalism for Combined Discrete-Continuous System Simulation. International Journal of General System 19(3), 226–240 (1991)