한국시뮬레이션학회 논문지
Vol. 21, No. 2, pp. 19-30 (2012. 6)

Structured DEVS Formalism: A Structural Modelling Method of Discrete Event Systems

Hae Sang Song[1†]

Structured DEVS Formalism: 이산사건 시스템의 구조적 모델링 기법

송해상

ABSTRACT

In recent decades, it has been known that the Discrete Event System Specification, or DEVS, formalism provides sound semantics to design a modular and hierarchical model of a discrete event system. In spite of this benefit, practitioners have difficulties in applying the semantics to real-world systems modeling because DEVS needs to specify a large size of sets of events and/or states in an unstructured form. To resolve the difficulties, this paper proposes an extension of the DEVS formalism, called the Structured DEVS formalism, with an associated graphical representation, called the DEVS diagram, by means of structural representation of such sets based on closure property of set theory. The proposed formalism is proved to be equivalent to the original DEVS formalism in their model specification, yet the new formalism specifies sets in a structured form with a concept of phases, variables and ports. A simplified example of the structured DEVS with the DEVS diagram shows the effectiveness of the proposed formalism which can be easily implemented in an objected-oriented simulation environment.

Key words : Structured DEVS formalism, DEVS Diagram, Phase, Port and Variable

요 약

최근 몇 십년간 이산사건시스템명세(DEVS) 형식론은 이산사건시스템을 모듈러하고 계층적으로 모델링할 수 있는 잘 정의된 의미론을 제공하여 왔다. 그럼에도 불구하고 실용 엔지니어들은 실세계의 시스템을 모델링에 적용하는데 어려움을 겪기도 하는데 이는 DEVS가 많은 상태와 사건들을 구조화되지 않은 형태로 명세해야 하는 것 때문이다. 본 논문은 집합 이론을 바탕으로 그러한 사건 및 상태집합들을 구조화된 형태로 표현하는 Structured DEVS 형식론과 이와 연관된 DEVS 다이어그램을 제안하고자 한다. 위상, 변수, 포트 등의 개념을 사용하여 집합들을 명세한 구조적 DEVS 형식론은 원래의 DEVS 형식론과 동등함을 증명하였다. DEVS 다이어그램을 이용하여 구조적 DEVS 형식론으로 표현된 예시 모델이 쉽게 객체지향 시뮬레이션 환경에서 구현될 수 있음을 보임으로써 제안된 형식론이 효과적임을 보였다.

주요어 : 구조적 DEVS 형식론, DEVS 다이어그램, 위상, 포트 및 변수

1. Introduction

Demands for Modeling and Simulation (M&S) have grown in recent decades, and Discrete Event Systems specification (DEVS)-based simulation applications have become more prolific especially in the domain of artificial systems such as industrial plant, war-game, communication network, and hybrid systems (Lee, 2010; Song, 2010; Lim, 2001; Kim, 2011). It is because DEVS has sound mathematical semantics for modular and hierarchical modeling and good simulation development environments (Kim 2010). Nevertheless, a practical M&S engineer still experiences difficulties in applying the

접수일(2012년 4월 14일), 심사일(1차 : 2012년 6월 12일), 게재 확정일(2012년 6월 12일)
[1] 서원대학교 컴퓨터공학과
주 저 자 : 송해상
교신저자 : 송해상
E-mail; hssong@seowon.ac.kr

classical DEVS formalism to a practical modeling of complex discrete event systems. This might be due to mainly two reasons: 1) the formalism itself is too mathematical for practical use and the behavior of a model is distributed into four separated functions; thus no integrated picture of a model is provided for the modeler to model and understand intuitively; and 2) in reality the number of sequential states or events is usually too large because of the complexity of systems under consideration.

For these reasons we need a more structured form of sequential states and sequential events to deal with these complex systems in M&S. Among DEVS-based simulation development environments, DEVSim++ (Kim 1992) introduces the notions of messages and phases rather than events and sequential states. Nevertheless, the environment does neither fully implement these concepts nor explicitly formalize this idea. Moreover, in the real-world fields, informal graphical modeling notations of their own have been utilized as an essential step before translating the graphical models into the mathematical DEVS model. Nevertheless, there has been no attempt to formally define a graphical language as well as to prove that it mathematically conforms to the semantics of the classical DEVS formalism.

There have been a few efforts on how to represent DEVS models in much easier ways rather than in mathematical one. Most of them are either graphical approaches or language-based approaches. The original version of DEVS diagram revised here is depicted as an example of RT-DEVS model (Hong and Song et al., 1997). The diagram notation has been used for a large number of commercial projects for large-sized defense modeling simulation fields (Kim 2010) with its capability to enrich the expressiveness. GGAD (Generic Graphical Advanced environment for DEVS Modeling and simulation) is a tool that adopts the diagram but with somewhat different appearance (Moallemi and Wainer 2010). An extension of UML for DEVS is proposed as a SysML/DEVS profile (Nikolaidou, 2008). However, we think that DEVS has to have its dedicated diagram rather than depending on existing ones to best express the mathematical model. It should also support necessary new notions for modeling large and complex systems, such as co-modeling (Kim, 2006) and structuring states and events.

Although the DEVS formalism and the DEVS graph form specify discrete systems in a modular and hierarchical manner in a system theoretical form, and they are adequate for logical analysis in our experience, it has been required to have other forms of graphical model representation for the development of large and complex systems' modeling simulations. Ad-hoc graphical notations have been used in war game modeling simulations (Kim, 2010; Lee, 2010). However, to authors' best knowledge, there has been no explicit literature dealing with such a graphical representation of the DEVS formalism in a structured form.

Thus this paper somewhat extends and compensates our previous works (Song and Kim, 2010) as an effort to revise the original diagram (Hong and Song et al., 1997) to support the mathematical foundation of the diagram.

Fig. 1 shows the scope of this paper that provides the modeling practitioners with a new graphical modeling language called the DEVS diagram with its semantics. To narrow gaps of mathematical models in the classical DEVS formalism and implementation models, we first define the structured form of the classical DEVS formalism, called structured DEVS formalism. It exploits the notion of port and message as a structured form of related events, and of state variables to aggregate relevant sequential states. These two types of DEVS formalism are of mathematical form best fit for M&S experts.

Meanwhile, for modeling practitioners we provide a graphical modeling language, called DEVS diagram. It not only depicts models in structured DEVS formalism but also aggregates many state transitions into a smaller number of 'phase transitions' to express them graphically in a simple way. To give confidence to the practitioners we prove that a graphical model in DEVS diagrams can be transformed to a mathematical model in structured DEVS formalism, which will then turn to the model of classical DEVS formalism, if some conditions are met. By this mechanism modelers could use the DEVS diagram for discrete event systems modeling and implementation without any concerns to learn mathe-

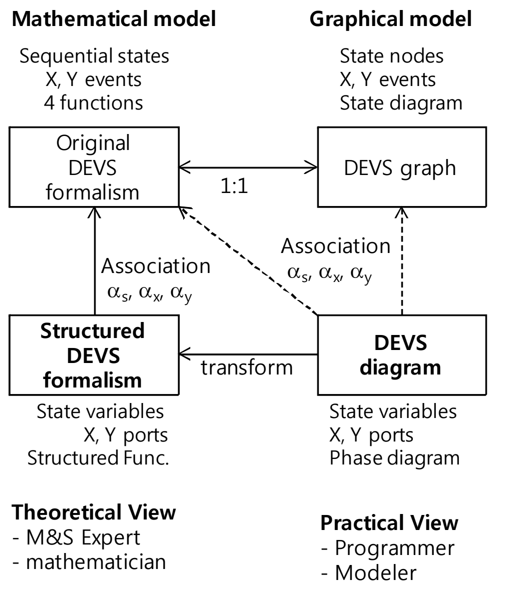

Fig. 1. Scope of the paper: structured DEVS and DEVS diagram with its relevance to the original DEVS formalism

matical notations of the DEVS formalism.

This paper is organized as follows. The next section introduces the background knowledge of the DEVS formalism, its basic graphical representation, and DEVS graph with their one-to-one relationship. Then we propose the structured DEVS formalism that is a structured form of the original one in Section 3. Then we define the DEVS diagram in Section 4. An illustrative example model in the DEVS diagram helps comprehend these concepts and shows how to implement the diagram models. Then we conclude the discussion.

2. Background

2.1 The classical DEVS formalism

Being well-known, the classical DEVS formalism can specify a system in two aspects: one for the behavior of a basic component, and the other for the overall structure of a system. An atomic DEVS formalism describes the behavior of a unit component as no further decomposable, which consists of three sets and four functions.

$$AM = < X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, ta >$$

where

X: input event set,

Y: output event set,

S: sequential state set, and total state set

$Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$,

$\delta_{ext} : Q \times X \rightarrow Q$: external transition function, for $\delta_{ext}(s,e,x) = (s',e')$, $e < ta(s), e' = 0$.

$\delta_{\text{int}} : Q \rightarrow Q$: internal transition function, for $\delta_{int}(s,e) = (s',e')$, $e = ta(s), e' = 0$.

$\lambda : Q \rightarrow Y$:output function, for $(s,e) \in Q$, $e = ta(s)$.

$ta : S \rightarrow R_0^+$: time advance function, where R_0^+ is the non negative real number set.

There are two types of transitions in an atomic model: 1) external state transitions caused by external events; and 2) internal state transitions in the case of no event occurrence until time on the current state has elapsed. In the latter case, just before the internal transition, an output event is generated at the state. In an analogy to the continuous systems, the external transitions correspond to the input-driven state transitions and the internal ones to the input-free state transitions.

The coupled DEVS formalism specifies the structure of discrete event systems composed of components communicating with each other through event couplings,

$$DN = < X, Y, M, EIC, EOC, IC, SELECT >$$

where

X: an input event set,

Y: an output event set,

M: a component model set, either atomic models or coupled models,

$EIC \subseteq DN.X \times \cup_i M_i.X_i$: external input coupling relation,

$EOC \subseteq \cup_i M_i.Y_i \times DN.Y$: external output coupling relation,

$IC \subseteq \cup_i M_i.Y_i \times \cup_i M_i.X_i$: internal coupling relation,

$SELECT$: $2^M - \varnothing \rightarrow M$: select function.

Notice that the coupled DEVS formalism above has the closure property, so that, for example, a coupled model may contain other coupled models as well as atomic models as its components. Thus it captures the structure of a system, the components hierarchy, and the interfaces between components. The SELECT function specifies the priorities of components when more than one component is to be scheduled at the same time in simulation.

2.2 The DEVS graph

The DEVS graph has been used for a long time in DEVS communities to sketch an atomic DEVS model, so that they have one-to-one correspondence with each other as the following definition demonstrates:

Definition 1. (DEVS graph) *The DEVS graph of an atomic model* $M = < X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta >$ *is defined as a labeled graph:*

$$AG = < N, E, ta >$$

where

$N = S$*: the state node set.*

$E \subseteq N \times (X \cup Y \cup \{\epsilon\}) \times N$*: the two-color labeled edge set with constraints:* $E = E_{ext} \cup E_{int}$,

$E_{ext} = \{(s, ?x, s') | \delta_{ext}(s, e, x) = (s', 0)\}$

$E_{int} = \{(s, !y, s') | \delta_{int}(s, e) = (s', 0),\ y = \lambda(s)$ *or* $y = \epsilon$ *if* $\lambda(s)$ *is not defined.*

$ta : N \to R_0^+$*: time advance,* $ta(n) = ta(s)$*, for* $n = s$. ■

A node is depicted by a circle with a note name ‘s’ inside and its time advance @ta(s) located near the circle. Each element of external transitions set E_{ext} is depicted by a solid arc with label ?x; meanwhile, an element of internal transitions set E_{int} is denoted by a dotted arc with label !y.

By definition it can be easily noticed that there can be only one internal transition at a state. The non- event ϵ represents the absence of an event. Since the time advance only depends on its state, it is just subordinate information. If the time advance of a state is not specified, then it is assumed to be infinite. Fig. 2 shows how the notation of the DEVS graph is derived; at first, the semantics of the internal and the external state transitions are shown in Fig. 2(a); then, its abbreviated form in 2(b); finally, the notation of the DEVS graph is defined in Fig. 2(c).

Since both the atomic and coupled models have their

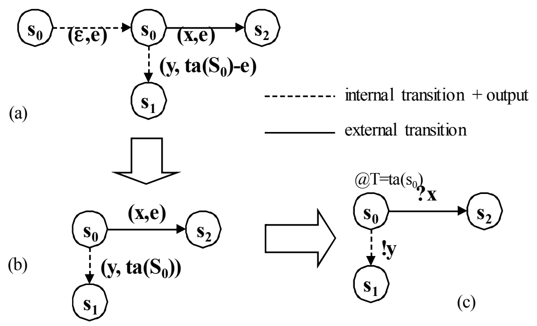

Fig. 2. The notation of DEVS graph (a) original idea, (b) more abbreviate form (c) the final form of DEVS graph

external interfaces, X and Y, we represent a model M as a box frame B with its name. Located along the border of the box are triangles inward for input events and outward for output events. Thus an atomic model corresponds to a box frame which owns a DEVS graph inside which represents the behavior of the atomic model drawn like above. The graphical model of a coupled DEVS model is also denoted by a box frame like the atomic model, in which, however, there exist many box frames of its child models connected to one another by links according to the coupling relations specified by EIC, EOC and IC. The name of a child model has the form of m:M, an instance or object ‘m’ of model ‘M’. The SELECT function should be described in a list of selections; for example, SEL (m1,m2,m4) = m1, SEL(m2,m3) = m3.

Consequently, the DEVS graph is a tool to specify in graphical form a discrete event system model which is one-to-one correspondence to the mathematical model in the classical DEVS formalism.

Definition 2. (Proper DEVS Graph) A DEVS graph is said to be proper if it satisfies the following constraints:

C1) (functionality) For outgoing edges from a node have unique events.

C2) (singleton output) For a node there is at most one internal transition.

Lemma 1. (Convertibility) A DEVS graph model always can be transformed to a (classical) DEVS atomic model in one to one manner if the graph is proper.

Proof) *It is obvious and trivial by the definition of DEVS graph in Def. 1.* ■

3. Structured DEVS formalism

According to Zeigler (1984, ch2), the classical DEVS atomic model specifies a system's behavior in I/O system level where states and events are sequential or flattened. This paper proposes a more structured form of the DEVS formalism called structured DEVS formalism, where the events sets and the state sets are transformed to structured sets. This section proposes a methodology of how to make structures over three essential modeling elements: events, states, and transitions. Then we define the structured DEVS formalism.

3.1 Structured set

Formally, a structured set A of sequential set A is defined as:

$$\mathrm{A} = < A, D, \{dom(a) | a \in D\}, \alpha >$$

where

A is a sequential set to be structured

D is an ordered set of coordinates, $a_1, a_2, \cdots$

$dom(a)$ is a range set of $a \in D$

α is the assignment function

subject to the constraint:

$$\alpha : A \rightarrow \times_{a \in D} dom(a)$$

is a one to one map. Since the assignment function i is a one to one map, it is assumed that the inverse function $\alpha^{-1} : \times_{a \in D} dom(a) \rightarrow A$ always exists.

Here we consider a coordinate as a variable with a name (e.g. a) and a range set (e.g. $dom(\mathrm{a})$). The concept of structured set is a system theoretic way to use variables in complicated systems modeling and simulation.

A structured function maps one structured set to the other. If A is structured by A and B by B, then function $f : A \rightarrow B$ is said to be structured. In this instance, f is a vector of coordinate functions, $f_{\mathrm{b}} : A \rightarrow B_{\mathrm{b}}$, one for each coordinate b of B. Note each f_{b} need not always depend on all of the coordinates of B so that a subset of coordinates of B, on which coordinate b depends, could be used to define the function.

3.2 Structuring event: port and message

First, event is a means of interaction between models. In real-systems modeling, however, the number of events is so huge and sometimes infinite that we cannot treat it easily by directly applying the DEVS formalism in the course of modeling. Thus, we need a mechanism of grouping these sequential events together into a related class to handle them in an easier way.

A collection of relevant events, in the sense that they are from the same source and headed to the same destination model with related information, can be classified into *equivalent events.* To treat them formally we introduce the notion of ports, message, and channel. A model has input and output ports instead of input and output events. A message is a datum that represents the information uniquely related to an event. A message variable is a container that can have the same type of messages. If a message m is arrived at an input port x, then it is called an input event denoted by $\mathrm{x}?m$; if a message m is leaving a port y, it is called an output event represented by $\mathrm{y}?m$. A port p, either an input port or an output port, is assumed to have its type or domain $\mathrm{dom}(\mathrm{p})$ and $m \in dom(\mathrm{p})$. A channel is a connection from one port y to another y such that $dom(\mathrm{y}) \subseteq dom(\mathrm{x})$. Therefore, it can be said the events of the same port are equivalent.

Now using the notion of structured sets, we can formally define structured events sets stated above. An input events set X is structured to a structured input events set X as follows:

$$\mathrm{X} = < X, P_{\mathrm{in}}, \{dom(\mathrm{x}) | \mathrm{x} \in P_{\mathrm{in}}\}, \alpha_{\mathrm{in}} >$$

where

X is a sequential events set to be structured

P_{in} is a set of input ports, $\mathrm{x}_1, \mathrm{x}_2, \cdots$

$dom(\mathrm{x})$ is the range set of a port $\mathrm{x} \in P_{\mathrm{in}}$

α_{in} is the assignment function

subject to the constraint:

$$\alpha_{\mathrm{in}}^{-1} : \times_{\mathrm{x} \in P_{\mathrm{in}}} dom(\mathrm{x}) \rightarrow X$$

such that $X = \{\ \mathrm{x}?m | \mathrm{x} \in P_{\mathrm{in}}, m \in dom(\mathrm{x})\}$. Similarly the output events set Y can be represented by a structured output events set Y:

$$\mathrm{Y} = < Y, P_{\mathrm{out}}, \{dom(\mathrm{y}) | \mathrm{y} \in P_{\mathrm{out}}\}, \alpha_{\mathrm{out}} >$$

such that $Y = \{\ \mathrm{y}!m | \mathrm{y} \in P_{\mathrm{out}}\ , m \in dom(\mathrm{y})\}$ by the one-to-one assignment function $\alpha_{\mathrm{out}}^{-1}$.

Hereafter for convenience we will often use X instead of its input port set P_{in} and Y instead of P_{out} in case of no confusion; therefore, for instance, x∈X actually denotes x∈P_{in} . Also can a structured input set X often be called the input ports set without confusion. Usually the range of a port is called the port type.

A *channel* is a connection from a port of a structured event set to a port of another structured event set. For example, a coupling (y,x) of a source output port y∈Y and an destined input port x∈X such that $dom(\mathrm{y}) \subseteq dom(\mathrm{x})$ is called a channel. A message $m \in dom(\mathrm{y})$ placed at the output port is assumed to be transported to the input port *instantly* with no message buffer; that is, if the receiver cannot get it instantly, the message will be discarded.

3.3 Structuring state: state variable

Analogy to the continuous systems specification, a discrete-event system's status can be specified by a set of system variables or attributes variables. Each state variable represents an attribute characterizing the system. Thus a system state is a combination of values that the state variables have at a time. This is a useful means of forming the structure of the sequential states.

Formally, a sequential states set S in DEVS is structured to a structured set S,

$$\mathrm{S} = < S, V, \{dom(\mathrm{v}) | \mathrm{v} \in V\}, \alpha_s >$$

where

S is a sequential events set to be structured

V is a set of state variables, $\mathrm{v}_1, \mathrm{v}_2, \cdots$

$dom(\mathrm{x})$ is the range set of a state variable v∈V

α_{s} is the one-to-on assignment function

subject to the constraint:

$$\alpha_{\mathrm{s}}^{-1} : \times_{\mathrm{v} \in V} dom(\mathrm{v}) \to S$$

is a bijection such that $S = \{s_i = (v_{1i}, v_{2i}, \ldots) | v_{1i} \in dom(\mathrm{v}_1), v_{2i} \in dom(\mathrm{v}_2), \ldots, \forall\ \mathrm{v}_i \in V\}$. An element $v_i = (v_{1i}, v_{2i}, \ldots)$ mapped to a sequential state s_i is called a *composite* state. Since the assignment function is straightforward and the range set for each state variable is implicit, we often mixedly use the structured states set S and the variables set V; for instance, v∈S means v∈V.

3.4 Structured atomic DEVS formalism

Now that we have reviewed the notion of structured sets and structured functions, we apply them to the classical DEVS formalism by introducing state variables and ports. Formally the structured atomic DEVS formalism has a structure:

$$\mathrm{AM} = < \mathrm{X}, \mathrm{Y}, \mathrm{S}, \Delta_{\mathrm{ext}}, \Delta_{\mathrm{int}}\ , \Lambda, Ta >$$

where

X: a structured input event set,

Y: a structured output event set,

S: a structured state (or state variable) set, where

$$Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}\ ,$$

$\Delta_{ext} : Q \times X \to Q$:

external structured transition function, for

$\Delta_{ext}(s,e,x) = (s',e'),\ e < ta(s), e' = 0$.

$\Delta_{\mathrm{int}} : Q \to Q$:

internal structured transition function, for

$\Delta_{int}(s,e) = (s',e'),\ e = ta(s), e' = 0$.

$\Lambda : Q \to Y$: structured output function, for

$\Lambda(s,e) = y \in Y,\ (s,e) \in Q,\ e = Ta(s)$.

$Ta : S \to R_0^+$: structured time advance function, where

R_0^+ is the non negative real number set.

subject to the constraint:

- input port set X is a structured set as defined above
  $\mathrm{X} = < X, P_{\mathrm{in}}, \{dom(\mathrm{p}) | \mathrm{p} \in P_{\mathrm{in}}\}, \alpha_x >$.
- output port set Y is a structured set as defined above
  $\mathrm{Y} = < Y, P_{\mathrm{out}}, \{dom(\mathrm{y}) | \mathrm{y} \in P_{\mathrm{out}}\ \}, \alpha_{\mathrm{out}} >$.
- state variable set S is a structured state set as defined above,
  $\mathrm{S} = < S, V, \{dom(\mathrm{v}) | \mathrm{v} \in V\}, \alpha_s >$.

Note that the four characteristic functions are all structured functions; therefore, any one of the functions can be defined as a multiple of sub-functions for each state variable. For instance, a state $s \in S$ is equivalent to a structured state or a composite state $s = (v_1, v_2, \ldots) \in \times_{\mathrm{v} \in V} dom(\mathrm{v})$ with a map $\alpha_s(s) = (v_1, v_2, \ldots)$ and a structured internal transition function $\Delta_{int}((v_1,v_2,...),e) = ((v'_1,v'_2,...),e')$, so $e = Ta((v_1,v_2,...)), e' = 0$ could be defined as a multitude of individual transition functions on *k*th variable, $\Delta^k_{int}((v_1,v_2,...),e) = (v'_k,e')$.

As indicated in Zeigler (1984, p. 45), we know the coordinate assignment functions α_s, α_x, α_y can be trivially defined by assigning each structured element to a distinct sequential element name (i.e., unique integer).

Lemma 2. (Association) *A structured atomic DEVS model always can be transformed to a classical DEVS atomic model.*

Proof) It is trivial to transform from a structured DEVS model to the corresponding non-structured one by inversely applying the coordinate assignment functions, $\alpha_s, \alpha_x, \alpha_y$. ■

3.5 Structured coupled DEVS formalism

The structured version of coupled modeling formalism is the same as the original formalism except the fact that the events sets are all structured, and that the model set contains structured model instances. Formally a structured coupled model has a 7-tuple,

$$DN = <X, Y, M, EIC, EOC, IC, SELECT>$$

where

X: a structured input event set,

Y: a structured output event set,

M: a component instances set of either structured atomic model or structured coupled model, where an instance has its name and model type.

$EIC \subseteq DN.X \times \cup_i M_i.X_i$: external input port coupling relation,

$EOC \subseteq \cup_i M_i.Y_i \times DN.Y$: external output port coupling relation,

$IC \subseteq \cup_i M_i.Y_i \times \cup_i M_i.X_i$: internal port coupling relation,

$SELECT : 2^M - \varnothing \rightarrow M$: select function,

with constraints:

for any for $(p_i,p_j) \in EIC \cup EOC \cup IC$,

$$dom(p_i) \subseteq dom(p_j).$$

It is guaranteed that the above-mentioned constraints will work as a strong type of tool to check messages flowing from a source port to the destination port.

4. The DEVS Diagram

4.1 Abstracting states into a phase

Although adopting the notion of state variables seems to suffice for structuring sequential states, we often need more abstraction of the behavior of a system with respect to controlling a point of view. For this purpose, in general, the concept of modes or phases has been frequently used.

Formally, a *phase* is a representative value of a set of equivalent states which produce the same output event and/or have the same time advance at the states. Using the notion of phase, we can simplify the state transitions even more by fewer numbers of phase transitions. A logical expression that filters out a subset of composite states belonging to a phase is called the *phase guard.* We call such states of a phase selected by the phase guard the *phase elements*.

Now consider that we partition the composite state set of a structured set S into disjoint equivalent classes such that each class has a set of sequential states with the same state sojourn time and/or output event. Let each class have a single representative name, called a phase. We can add a phase variable having such phases as its values to the state variable set to get an augmented state variables set. We designate a phase variable as a high-level state variable, and the original ones as the low-level state variables or *system attribute variables*. Then, a state of the system becomes a combination of a phase value and a composite state of state attributes. Formally an *augmented* structured state set S including a phase variable ψ is defined as:

$$S = < S, V \cup \{\psi\}, \{dom(v) | v \in V \cup \psi\}, \alpha_s >$$

where, $dom(\psi) = \{\varphi_i\}$, φ_i is a phase of the phase variable ψ. The phase guard of a phase φ_i is a logical expression on system attributes set V that discriminates the phase elements of the phase among all states. It is denoted by $G_{\varphi_i} : \times_{v \in V} dom(v) \rightarrow Boolean$, where the phase elements set of phase φ_i selected by guard G_{φ_i} is denoted by $S_{\varphi_i} = \{(\varphi_i, v) \in S \mid G_{\varphi_i}(v) = true\}$ and the phase elements sets should be disjoint with each other.

A phase can be hierarchical; that is, the elements of a phase can be once again partitioned into sub-phases having disjoint composite state members, and so on. It is always true that a union of states sets of sub-phases within a phase gives the states set of the phase and the intersection of any sub-phases results in an empty set.

Finally we define a structured state change function, called an *action,* as a function from the composite state set to itself; $A : \times_{v \in V} \mathrm{dom}(v) \rightarrow \times_{v \in V} \mathrm{dom}(v)$. As an action is a structured function, it may construct a vector of individual actions for each variable: $A_v : \times_{v \in V} \mathrm{dom}(v) \rightarrow \mathrm{dom}(v)$, $v \in V$. Using the notion of phase, guard, and action, we can now define the state transitions in a more finite form.

4.2 Phase transition diagram

Let there be an input ports set, an output ports set and state variables set of a structured atomic model. Let there be a phase variable where each has a disjoint composite member states set with its unique phase guard. We then formally define the specification of the *phase transition diagram* as follows:

Definition 3. Phase (Transition) Diagram: *A phase diagram for a structured DEVS atomic model* $AM = < X, Y, S, -, -, -, - >$ *with a structured input event set* X*, a structured output event set* Y*, and an augmented structured state set* S *including a phase variable* ψ *is defined as a labeled graph,*

$PD = < N, E, T >$

where for actions set A *and guards set* G *on system attributes set* V *(excluding the phase variable),*

$N = dom(\psi)$*: a phase nodes set for phase variable* ψ,

$E \subseteq N \times L \times N$*: the two-color labeled edge set, where* $L = L_x \cup L_y$ *with a guarded input/action set* $L_x \subseteq X \times G \times A$, *a guarded output/action set* $L_y \subseteq G \times Y \cup \{\epsilon\} \times A$

$T : N \times G \rightarrow R_0^+$*: time advance of a phase.* ■

Fig. 3 shows the DEVS diagram notations for the behavior of a structured atomic DEVS model based on phase. Fig. 3(a) represents a list of structured variables with their domains (or types) and optionally their initial

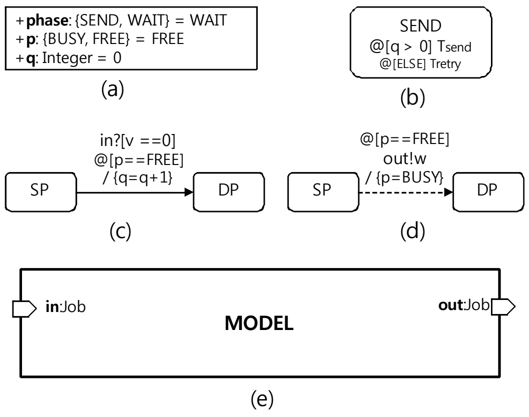

Fig. 3. Notations of atomic DEVS diagram (a) a variable box for a structured states set (b) a phase with time advances (c) an external phase transition with an input/guard/action label on it (d) an internal phase transition with a guard/output/action label (e) a model box for the model name at the center with a structured input port:type (left) and an output event port:type (right)

values. A variable is the form of ‘name:type=init_value’ to denote a variable name ‘name’ with its type ‘type’ and its initial value ‘init_value’ in accordance with UML notation. A model is represented by a box with port:type pairs. A structured input event ‘in:type’ and a structured output event ‘out’ of type ‘Job’ is denoted by (e). The rest of the sub-figures are elements of a phase transition diagram as defined above. Fig. 8(b) is a phase node with its name ‘SEND’ and two time advances at that phase: ‘@[q>0] Tsend’, ‘@[ELSE] Tretry’. This means, at phase ‘SEND’, if an attribute variable ‘q’ is greater than ‘0’, then the time advance is ‘Tsend’ or ‘Tretry’ elsewhere. An external transition is represented by (c); from a source phase ‘SP’ to a destination phase ‘DP’ with a guarded input action label, ‘in?[v==0] @[p==FREE] / {q = q+1}’, which means that if the value input to port ‘in’ is ‘0’ stored temporally in variable ‘v’, and a system attribute variable ‘p’ has value of ‘FREE’, then it changes the state to ‘phase = DP’ and ‘q = q+1’. Note that the action set is described in sub-function only for a variable ‘q’. The rest of attributes are not changed at all. Finally, Fig. 8(a) shows that if ‘p == FREE’ at phase ‘SP’, then an output to port 'out' with the value in variable 'v' is produced and it changes the state to 'phase=DP' and 'p=BUSY' leaving the rest attributes intact. Extended notations and applications of the diagram can be found in our previous work (Song and Kim 2010).

4.4 Atomic DEVS diagram

Now we define a class of phase transition diagrams that confirms to the four characteristic functions of the structured DEVS formalism.

Definition 4. (**Proper phase diagram**) A phase diagram $PD=<N,E,T>$ of an atomic structured DEVS model AM=< X,Y,S,–,–,–,–> is said to be *proper* if it satisfies all of the following constraints on each phase node:

C1) (functionality) for any outgoing edges from the node with the same (either input or output) event, the guards should be disjoint.

C2) (singleton output) for any outgoing edges with different output events, the guards of the edges should be disjoint. If no internal transition edge is defined, then the time advance is assumed to be infinite.

C3) (time integrity) if there are more than one time advances on the node, which are defined, then the guards of them should be disjoint.

Theorem 1. (Convertibility) A proper phase transition diagram $PD=<$N,E,T$>$ of a atomic DEVS model AM=< X,Y,S,–,–,–,–> can always be converted to a structured atomic DEVS model, AM=< X,Y,S,Δ_{ext}, Δ_{int} ,Λ, Ta >.

Proof) (*Rationale) It is straightforward to transform a proper phase transition diagram to a structured atomic DEVS* AM=< X,Y,S,Δ_{ext},Δ_{int} ,Λ, Ta > *by the constraints in Definition 3. Taking into account that a composite state consists of a combination of a phase and attribute values of system attribute variables, external/ internal transitions should form a function of composite states and input/output events respectively since the source states are all different by the guards, even though the event is the same. We see that this is guaranteed by constraint C1. Additionally, as an output event is defined on a state in DEVS, only single internal transition should be defined on a composite state. Constraint C2 makes sure of this. Just like the output function mentioned above, the time advance function is also a function of composite states which is assured by constraint C3.* ■

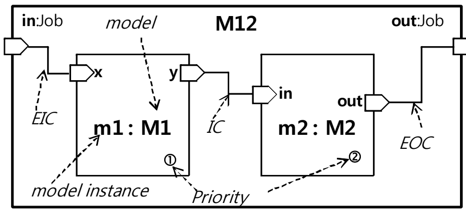

Fig. 4. DEVS diagram notations of structured coupled model. M12 - coupled DEVS model type, m1:M1, m2:M2 - component model instances, the priorities are specified for a select function, and three types of port couplings according to the context

If a model box has a variable box and a phase transition diagram, then the box is called *an atomic DEVS diagram*.

4.5 Coupled DEVS diagram

Fig. 4 illustrates the notations of a coupled DEVS diagram. As the diagram has a one-to-one relationship with the structured coupled DEVS formalism, extracting a mathematical model from the diagram is rather straightforward. For example, the mathematical model M12 of coupled DEVS diagram in Fig. 4 is obtained as follows:

M12 =<X,Y,M,EIC,EOC,IC,SELECT>
where
X = {in:Job}, Y = {out:Job}, M={m1:M1,m2:M2},
EIC = {(in, m1.x)}, IC={(m1.y, m2.in)},
EOC={(m2.out,out)},
SELECT({m1,m2}) = m1.

5. An Illustrative Example

Consider an example of proper atomic DEVS diagram in Fig. 5. The phase diagram is proper since it satisfies all the constraints of Def. 3: C1) functionality since the guards of edges with the same event name are disjoint; for example, for an external input event 'in' at

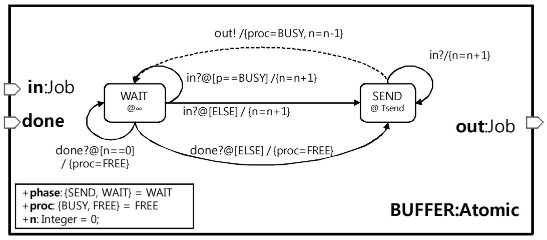

Fig. 5. An example of an atomic model: BUFFER

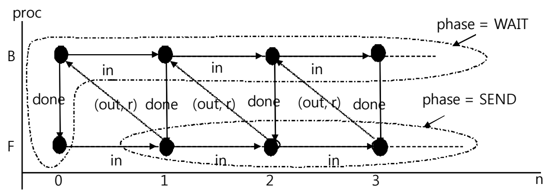

Fig. 6. The DEVS graph corresponding to the phase transition diagram of model BUFFER in Fig. 5

phase ‘WAIT’, the guards are [p==BUSY] and [ELSE] that are disjoint, C2) singleton output as there is only one internal edge with an output event ‘out’ at phase ‘SEND’ , C3) time-integrity because for each phase, one time advance is defined. Thus by corollary 1, we can obtain a mathematical model in the structured atomic DEVS formalism,

$$\text{BUFFER} = < X, Y, S, \Delta_{\text{ext}}, \Delta_{\text{int}}, \Lambda, Ta >,$$

whose sets are as follows:

X = {in:Job, done}, **Y** = {out:Job},

S = {phase:{WAIT,SEND}, proc:{FREE,BUSY}, n}

where the initial state is (WAIT, FREE, 0).

According to the phase diagram, the four functions are defined as follows:

$$\Delta_{ext}((WAIT,F,n),e,in) = ((SEND,-,n+1),0),$$
$$\Delta_{ext}((WAIT,B,n),e,in) = ((WAIT,-,n+1),e),$$
$$\Delta_{ext}((WAIT,-,0),e,done) = ((WAIT,F,-),0),$$
$$\Delta_{ext}((WAIT,-,n>0),e,done) = ((SEND,F,-),0),$$
$$\Delta_{ext}((SEND,-,n),e,in) = ((SEND,-,n+1),e)$$
$$\Delta_{\text{int}}((SEND,-,n),e) = ((WAIT,B,n-1),0),$$
$$\Lambda((SEND,-,-)) = out,$$
$$Ta((SEND,-,-)) = Tsend,$$
$$Ta((WAIT,-,-)) = \infty.$$

Note that letter ‘-’ in the lefthand side of the equation means that any value possible in the context and the one in the righthand side are not changed at all by the action. If there exist variables in an equation rather than values, the equation is a representative one for many possible equations with a combination of values of the variables. In addition, the elapsed time *e* appearing on the righthand side means the continuation of existing schedule.

Starting from the initial state, we can obtain a DEVS state transition graph as in Fig. 6 from the phase transition diagram in Fig. 5 substituting all possible combinations of values to the state variables.

Conversely, we see that the two phases ‘WAIT’ and ‘SEND’ are partitioning the total composite states set into two subsets with their phase guards:

o SEND := (n > 0) and proc == FREE

o WAIT := (n==0) or proc == BUSY

We can often use these phase guards to check the correctness of actions. Practically when we describe the behavior of an atomic model, usually we first define phases and their phase guards for consistency. Comparing the phase transition diagram in Fig. 5 with the corresponding state transition graph in Fig. 6, we can easily notice that the former is even a compact representation of the same behavior from a developers’ view.

Fig. 7 illustrates a pseudo-code to implement the phase diagram in Fig. 5. It also includes an optional assertion function to check the consistency of actions. As shown in the code, what developers have to do is to translate the diagram into code in a straightforward way.

6. CONCLUSION

This paper proposed structured DEVS formalism as well as the DEVS diagram for practical modeling of discrete event systems. The main idea is to introduce the notions of port and message, and state variable and phase, which group together sequential states and events into ports and variables. The diagram itself has been used for years but not defined formally, and it has lacked a mathematical foundation; thus, sometimes it

```
ExtTransFn(s, e, x)  //External Trans
function
      case x.port () of
          in:
             if phase == WAIT //current phase
                if proc == BUSY //guard
                    n := n+1,   // action
                    continue ;  // old
schedule
                else
                    phase := SEND,
                    n := n+1,
             if phase == SEND
                n := n+1,
                continue;
           done:
             if phase == WAIT && n == 0
                proc := FREE   //new schedule
             else
                phase := SEND, proc := FREE;
      AssertPhase(); // phase assertion

IntTransFn(s)   //  DEVS Internal Trans
function
      if phase == SEND
          phase := WAIT,
          proc := BUSY, n := n - 1;
      AssertPhase();

OutputFn(s) //   DEVS Output function
      if phase == SEND
          output to out; // output to port out

TimeAdvanceFn(s)  // DEVS Time advance
function
      if phase == SEND
          ta = Tsend;
      else
          ta = ∞;

AssertPhase() // optional phase assertion
      if phase == SEND
          if not (n > 0 and proc == FREE)error;
      else if phase == WAIT
          if not (n == 0 or proc == BUSY)
error;
```

Fig. 7. Pseudo-code of the characteristic functions of the expanded atomic model BUFFER

has lacked semantics, has been incomplete, and even violated the DEVS formalism. We did a comprehensive and thorough job for the definition of the DEVS diagram to provide engineers with an efficient tool for modeling and designing of a simulation software. Due to lack of space, however, this paper cannot provide the thorough mathematical foundation based on the structured DEVS formalism; thus we have tried to focus on the diagram itself. Our subsequent full paper to be submitted will prove that the DEVS diagram is based on the thorough mathematical foundation. Based on this work, we will update DEVS Specification Language in script form and plan to make a computer-aided modeling tool for the DEVS based simulation development. More work, however, is required to implement a DEVS simulation tool conforming to the proposed DEVS diagram. As we expect, the more prolific DEVS applications become as M&S demands grow, the more our work will be utilized in the domain of the complex M&S.

Acknowledgments

I would like to express special thanks to Prof. Tag Gon Kim who advised me on this work with a lot of discussions and comments.

References

1. Lee, S.Y, Jang, S.H., Lee, J.S. (2010) “워게임 시뮬레이션에서 전장상황을 고려한 최적경로 모델링 및 시뮬레이션”, 한국시뮬레이션학회논문지, Vol. 19, No. 3, pp. 27-35.
2. Lim, S.Y. and Kim, T.G. (2001) “DEVS형식론에 기반한 하이브리드 시스템 모델링 시뮬레이션 방법론 - 제1부:모델링 및 시뮬레이션 방법론”, 한국시뮬레이션학회논문지, Vol. 10, No. 3, pp.1-14.
3. Song, Hae S., Lee, J.Y., Kim, T.G. (2010) “DEVS-based Modeling Simulation for Semiconductor Manufacturing Using an Simulation-based Adaptive Real-time Job Control Framework”, 한국시뮬레이션학회논문지, Vol. 10, No. 3, pp. 45-54.
4. Hong, Jun S.; Hae S. Song; Tag G. Kim and K. H. Park. 1997. “A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development”, Discrete Event Dynamic Systems, Vol. 7, No. 4, pp. 355-375.
5. Hong, Ki J. and Tag G. Kim. (2006) “DEVSpecL-DEVS specification language for modeling, simulation and analysis of discrete event systems”, Information and Software Technology, Vol. 48, No. 4, pp. 221-234.
6. Kim, Jae H. and Tag G. Kim. (2006) “Parametric Behavior Modeling Framework for War Game Models Development Using OO Co-Modeling Methodology”, 2006 Spring Simulation MultiConf., Huntsville, USA, pp. 69-75.
7. Kim, Tag G.; C. H. Sung; S.Y. Hong; J.H. Hong; C.B.

Choi, J.H. Kim; K.M. Seo; and J.W. Bae. (2011) “DEVSim++ Tools Set for Defense M&S and Interoperation”, The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 8, No. 3, pp. 129-142.

8. Kim, Tag G. and S. B. Park. (1992) “The DEVS formalism: hierarchical modular systems specification in C++”, 1992 European Simulation Multi-conference, York, United Kingdom. pp. 152-156.
9. Moallemi, M. and Gabriel A. Wainer. (2010) “Designing and Interface for Real-Time and Embedded DEVS”, Proceedings of 2010 Spring Simulation Conference, pp. 154-161.
10. Nikolaidou, M.; V. Dalakas; L. Mitsi; G.D. Kapos; and D. Anagnostopoulos. (2008) “A SysML Profile for Classical DEVS Simulators”, Proceedings of 3rd Internal Conference on Software Engineering Advances, pp. 445-450.
11. Song, Hae S. and Tag G. Kim. (2005) “Application of Real-time DEVS to Analysis of Safety-critical Embedded Control Systems: Railroad-crossing Control Example”, Simulation, Vol. 81, No. 2, pp. 119-136.
12. Song, Hae S. and Tag G. Kim. (2010) “DEVS Diagram Revisited: A Structured Approach for DEVS Modeling”, Proc. of 2010 European Simulation and Modelling Conference, oct. 25-27 Hasselt, Belgium, pp. 94-101.
13. Zeigler, B.P. (1984), Multifacetted Modeling and Discrete Event Simulation, Academic Press.
14. Zeigler, B. P. and Tag G. Kim. (2000) Theory of Modelling and Simulation (2nd Ed.), Academic Press.

송 해 상 (hssong@seowon.ac.kr)

1991 한국과학기술원 전기및전자공학과 공학석사
2000 한국과학기술원 전기및전자공학과 공학박사
1999∼2000 고등기술연구원 연구원
2002∼현재 서원대학교 컴퓨터공학과 교수

학술활동 : 한국시뮬레이션 학회 종신회원, 한국시스템공학회 이사
관심분야 : 이산사건시스템 M&S 이론 및 응용, 국방시스템공학.