Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

FASTER FLATTENING OF HIERARCHICAL DEVS MODEL FOR ACCELERATED SIMULATION

Jang Won Bae
Su-Jin Shin
Il-Chul Moon

Department of Industrial and Systems Engineering
KAIST, 291 Daehak-ro
Yuseong-gu, Daejeon 305-701, Republic of Korea

ABSTRACT

DEVS formalism is a widely utilized representation for a discrete event model, and this formalism is frequently adapted to specify the structure and the behavior of simulation models. One advantage of DEVS is its hierarchy structure of flexible composition of models, but this hierarchy often becomes a double-edge sword that might hinder a faster simulation because the simulation engine should navigate the hierarchy. Hence, in practice, simulations are performed by a flattened model instead of a hierarchical model. This article introduces a new matrix-based algorithm to create the flattened model rapidly to shorten the flattening time. This algorithm exploits properties of coupling structure to limit the matrix element changes to accelerate the process. This paper shows the competitiveness of this algorithm by comparing to the tree based flattening. We expect that this algorithm is particularly useful when a model dynamically changes its structure and the flattening should be redone repeatedly.

1 INTRODUCTION

Modeling and simulation (M&S) in discrete event systems have been adopted to understand, analyze and predict sophisticated systems. After an object-oriented paradigm was introduced, there seemed to be a trend about utilizing features of the object-oriented paradigm, such as encapsulation and inheritance to M&S. One of the reasons might be that the object-oriented paradigm could support hierarchical modeling. Hierarchical modeling describes a complex system as a simple model including multiple component models. With hierarchical modeling, we could make use of several advantages, such as developing models with ease, increasing reuse of models, and aiding in model validation (Sargent et al. 1993).

Discrete event system specification formalism enables hierarchical and modular descriptions of discrete event systems and furnishes abstract simulation algorithms to execute DEVS models (Zeigler, Praehofer, and Kim 2000). DEVS formalism have supported developing models in various domains, such as traffic control (Lee, Lee, and Chi 2003), military field(Kim, Moon, and Kim 2011), crisis management (Lee, Oh, and Moon 2012) and urban design (Palaniappan, Sawhney, and Sarjoughian 2006). However, DEVS formalism causes overheads in executing DEVS models. These overheads are caused from a hierarchical structure in DEVS. The hierarchical structure contains redundant models that describe hierarchical structures of systems, which requires extra costs for message passing and time scheduling during component models

There have been many studies to reduce the overheads in DEVS formalism. In Lee and Kim's result, the overheads could be categorized into three parts: state transitions in DEVS models, message passing and time scheduling among DEVS models (Lee and Kim 2003). They showed that message passing and time scheduling are more significant factors to simulation performances, which directly come from hierarchical structures. Most studies on improving the simulation performances branched out into

mitigating these three overheads. One approach to tackle these problems is revising both DEVS formalism and abstract simulation algorithms. However, these approaches do not essentially mitigate the overheads from hierarchical structures. The other trend is revising only the abstract simulation algorithms without modifications on DEVS formalism. Most studies of this trend focused on minimizing the overheads from hierarchical structures using the $flattening$ method. The $flattening$ method removes the hierarchy of the DEVS model so that the overheads from hierarchical structures would be minimized. The resultant model of $flattening$ procedure is called a flattened model. In constructing a flattened model, reconstructing the coupling structures in the flattened model could also be time consuming. In previous works, they used a simple searching algorithm to construct a flattened model. The algorithm, which is called a naïve method in this paper, finds coupling structures in a flattened model by checking all coupling relations in original DEVS models, which is similar to a depth-first search. Hence, the naïve method might be vulnerable to increasing hierarchy level and scalability of models. However, to our best knowledge, there are few studies about considering how to efficiently develop flattened models from DEVS models.

In this paper, we propose a matrix-based representation of coupling structures in a DEVS model and an algorithm for constructing a flattened model using matrix operations. In order to support the algorithm, we define coupling relations of DEVS models as a relation matrix form, such as coupling structure matrix (CSM) and flattened coupling structure matrix ($FCSM$). Constructing CSM could be performed using coupling specifications of DEVS model, but constructing $FCSM$ requires an additional method. That is why we proposed an algorithm describes how to develop $FCSM$ using multiple CSMs. The result of our experiments illustrates that our algorithm shows about 10% speedup in simulation performances than the naïve method. Furthermore, the result is sufficient to estimate that our algorithm would show better performance as increasing hierarchy level and number of component models.

2 BACKGROUND

In this section, we introduce several backgrounds of our work. Firstly, we briefly review DEVS formalism and abstract simulation algorithms. Secondly, we illustrate previous studies related to improving simulation performance.

2.1 DEVS Formalism

Discrete event system specification (DEVS) formalism is a general method of developing discrete event models (Zeigler, Praehofer, and Kim 2000). DEVS formalism provides a modular and hierarchical modeling framework and supports to realize models with abstract simulation algorithms. DEVS formalism is composed of two models: atomic model (AM) and coupled model (CM). AM describes behaviors of component models in a target system and CM describes coupling structures between CM itself and component models. The detailed specifications of AM and CM are in Figure 1:

Atomic Model, $AM = < X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta >$	Coupled Model, $CM = < X, Y, M, EIC, EOC, IC, Select >$
X : a set of input events	X : a set of input events
Y : a set of output events	Y : a set of output events
S : a set of states	M : a set of DEVS models of component models in CM
$\delta_{ext} : Q \times X \rightarrow S$, an external transition function,	$EIC \subseteq CM.X \times \cup M_i.X$: external input coupling relations
$(Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the total state set of $AM)$	$EOC \subseteq \cup M_j.Y \times CM.Y$: external output coupling relations
$\delta_{int} : S \rightarrow S$, an internal transition function	$IC \subseteq \cup M_i.Y \times \cup M_j.X$: internal coupling relations
$\lambda : S \rightarrow Y$, an output function	$Select$: tie-breaking function
$ta : S \rightarrow R^+$, a time advance function	

Figure 1: Atomic and Coupled Model.

Hierarchy structure of a DEVS model could be represented by a tree structure, which is called as a decomposition tree. Figure 2 shows a DEVS model, its flattened model, and their associated decomposition tree. For the result of the flattening procedure, intermediate coupled models, which are component models at the level of a coupled model, are removed from the original DEVS model. The removed models are easily discovered using decomposition trees in Figure 2.

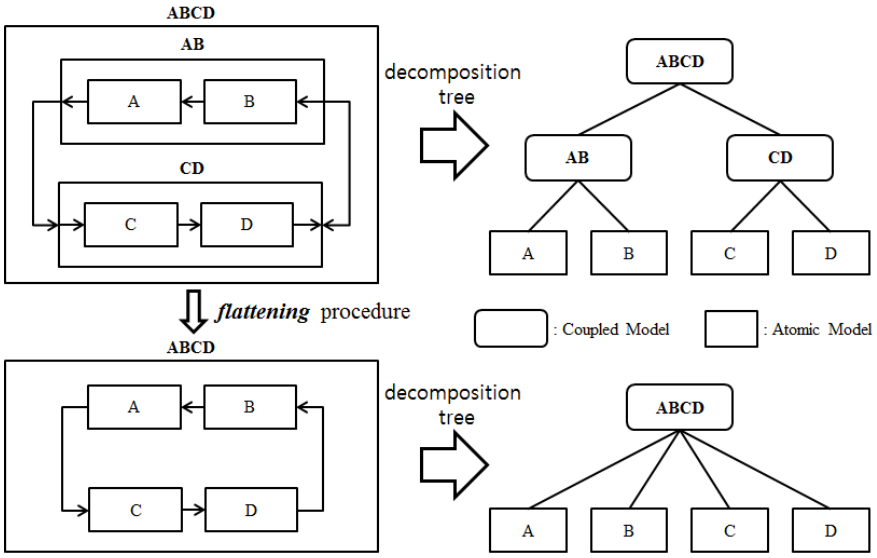

Figure 2: An example of a DEVS model, its flattened model and related decomposition trees.

In the abstract simulation algorithms, simulators and coordinators are assigned to each *AM* and *CM*. Simulators and coordinators execute assigned DEVS models by transmitting four messages, such as $(*, t)$, $(done, t_N)$, (x, t) and (y, t); Each message relates to computing output and next state, noticing next event time, receiving input "x" and sending output "y" at time t, respectively. Although only *AM*s generate and consume event messages, all of the messages are transmitted through coupling relations of *CM*s. Eventually, this simulation protocol of the abstract simulation algorithms causes the overheads in executing DEVS simulation.

2.2 Related Work

To improve simulation performance in DEVS models, many studies have been performed with two directions: extending DEVS formalisms and applying flattening procedure.

Extended DEVS formalisms were branched out into 1) parallel and distributed simulation and 2) hybrid simulation. Because DEVS formalism assumes sequential simulation execution, parallel and distributed simulation would reduce simulation costs. Chow firstly invented PDEVS and its abstract simulation algorithms (Chow 1996). PDEVS approach was applied to improving DEVS simulation environment (Zeigler et al. 1997), developing parallel Cell-DEVS (Troccoli and Wainer 1997), and realizing a DEVS environment on GPU (Liu and Wainer 2012). On the other hand, hybrid simulation is one of methods for reducing simulation costs by fast calculations from analytic models (Shanthikumar and Sargent 1983) (Goswami and Iyer 1993). Hybrid simulation environments are often developed in HLA/RTI (Venkateswaran, Son, and Jones 2004) and in plug-in manners (Bae and Kim 2010). Because these works are focused on improving computation power, they could not achieve a significant improvement of simulation performances.

In order to achieve a significant improvement in simulation performances, the overheads from message passing and time scheduling should be reduced. Kim and Kim developed a meta-model for optimizing hierarchy level of DEVS models to minimize the overheads of message passing and time scheduling (Kim

and Kim 1997). According to their result, lower-level hierarchy does not always imply better simulation performances. However, in general cases, simulation performances are getting worse as the hierarchy level of models are increasing (Kwon and Kim 2012). As reflecting this tendency, there have been many studies of removing the hierarchy of models by $flattening$ procedures for better simulation performances. Kim et al. suggested a DEVS cluster for distributed simulation and applied flattening method to the DEVS cluster (Kim et al. 2000). Lee and Kim applied composition-based compilation, the same as the $flattening$ method, using formal specifications (Lee and Kim 2003). Wainer and Giambiasi applied flattening methods to Cell-DEVS models and showed better simulation performances (Wainer and Giambiasi 2001). Additionally, there are several studies employing an event-oriented paradigm to DEVS models using the flattening method (Muzy and Nutaro 2005) (Kwon and Kim 2012). Although the flattening method was applied to many studies and showed improvements in simulation performances, there are few studies of how to efficiently get a flattened model from a DEVS model. Most previous studies developed a flattened model using a naïve method. However, as hierarchy level of model or the number of component models is increased, a performance of the naïve method is getting worse. Therefore, this paper proposes matrix-based representations and manipulations of the coupling structure in DEVS models to develop flattened models from DEVS models efficiently and robustly.

3 MATRIX REPRESENTATION OF COUPLING STRUCTURE

In this section, we illustrate matrix representations of coupling structures in DEVS models and in flattened models. Before introducing matrix representations of coupling structures, we summarize frequently used notations in Table 1.

Table 1: Summary of Notations.

Notation	Description
M	DEVS model set
$M.X$	Input event set in M
$M.x_i$	i^{th} input event in $CM.X$, i is an index of $M.X(1 \leq i \leq \|M.X\|)$
$M.Y$	Output event set in M
$M.y_i$	i^{th} output event in $CM.Y$, i is an index of $M.Y(1 \leq i \leq \|M.Y\|)$
$CM.M$	Component model set in CM
$CM.m_i$	i^{th} component model in $CM.M$, i is an index of $CM.M(1 \leq i \leq \|CM.M\|)$
$CM.AM$	Component model set containing leaf node models (i.e. atomic model) of decomposition tree of model CM
$CM.am_i$	i^{th} component model in $CM.AM$, i is an index of $CM.AM(1 \leq i \leq \|CM.AM\|)$
(e_i, e_j)	Coupling relation from e_i to e_j, where e_i and e_j are events
R_M	Candidate set of source events in the relation matrix M
C_M	Candidate set of destination events in the relation matrix M
CSM_m	Coupling Structure Matrix of model named m
$FCSM_m$	Flattened Coupling Structure Matrix of model named m
H_m	Height of decomposition tree of model m

3.1 Coupling Structure Matrix

CSM of a DEVS coupled model cm (CSM_{cm}) is a relation matrix which represents coupling structures in the DEVS model cm. Coupling structures in a DEVS model mean how the model and its component models are interconnected. An (i, j) element of CSM_{cm}, which is $(CSM_{cm})_{ij}$, represents whether there is a coupling relation from an event r_i to another event c_j. When we construct the coupling structure matrix, we

should define source events (e.g. r_i) and destination events (e.g. c_j) of the DEVS model. The source events and destination events of the DEVS model could be found in the specifications of coupling relations in the DEVS model. Therefore, we can derive a set of candidates for source and destination events using coupling relations in *EIC*, *EOC*, and *IC* in the DEVS model. In detail, a set of source events in *CM* (R_{CM}) would be a union set of $CM.X$ and $\bigcup_{i=1}^{|CM.M|}(CM.m_i).Y$ and a set of destination events in *CM* (C_{CM}) would be a union set of $CM.Y$ and $\bigcup_{i=1}^{|CM.M|}(CM.m_i).X$. Based on this understanding, we could mathematically define CSM_{cm} in (1).

$$Let\ \ CSM_{CM}(\in \mathbb{M}(m,n))\ \ be\ \ an\ \ m\times n\ \ matrix: \tag{1}$$

$$R_{CM} = CM.X \bigcup \left(\bigcup_{i=1}^{|CM.M|} (CM.m_i).Y \right), \quad C_{CM} = CM.Y \bigcup \left(\bigcup_{i=1}^{|CM.M|} (CM.m_i).X \right)$$

$$CM.X = \{CM.X_{1:l}\},\ where\ \ l = |CM.X|, \quad CM.Y = \{CM.y_{1:p}\},\ where\ \ p = |CM.Y|$$

$$(CM.m_i).X = \left\{(CM.m_i).x_{1:q_i}\right\}, \quad where\ \ q_i = |(CM.m_i).X|\ \ for\ 1 \le i \le |CM.M|$$

$$(CM.m_i).Y = \{(CM.m_i).Y_{1:k_i}\}, \quad where\ \ k_i = |(CM.m_i).Y|\ \ for\ 1 \le i \le |CM.M|$$

$$\therefore R_{CM} = \{r_{1:m}\} = \left\{CM.x_{1:l}, (CM.m_1).y_{1:k_1}, \cdots, (CM.m_{|CM.M|}).y_{1:k_{|CM.M|}}\right\}, \quad where\ \ m = l + \sum_{i=1}^{|CM.M|} k_i$$

$$\therefore C_{CM} = \{c_{1:n}\} = \left\{CM.y_{1:p}, (CM.m_1).x_{1:q_1}, \cdots, (CM.m_{|CM.M|}).x_{1:q_{|CM.M|}}\right\}, \quad where\ \ n = p + \sum_{i=1}^{|CM.M|} q_i$$

For example, *CSM* of the model *cm* (CSM_{cm}) would be an "m by n" matrix. "m" is equal to the sum of the number of input events of the model *cm* ($|cm.X|$) and the number of output events of component models in the model *cm* ($\sum_{i=1}^{|cm.M|}|(cm.M_i).Y|$). "n" is equal to the sum of the number of output events of the model *cm* ($|cm.Y|$) and the number of input events of component models in the model *cm* ($\sum_{i=1}^{|cm.M|}|(cm.M_i).X|$).

$$(csm_{CM})_{ij} = \begin{cases} 1 & if (r_i, c_j) \in EIC\ \ in\ \ CM \\ & else\ if\ (r_i, c_j) \in EOC\ \ in\ \ CM \\ & else\ if\ (r_i, c_j) \in IC\ \ in\ \ CM \\ 0, & otherwise \end{cases} \tag{2}$$

An $(i,\ j)$ element of CSM_{cm}, $(csm_{cm})_{ij}$, is defined by *EIC*, *EOC* and *IC* in model *CM* (2). If a coupling relation from an r_i event to another c_j event is existing in the model *cm*, $(csm_{cm})_{ij}$ is set to one. Otherwise, $(csm_{cm})_{ij}$ is set to zero.

$$CSM_{CM} = \begin{pmatrix} (CSM_{CM})_{11} & (CSM_{CM})_{12} \\ (CSM_{CM})_{21} & (CSM_{CM})_{22} \end{pmatrix} = \begin{pmatrix} Self-loopM_{CM} & EICM_{CM} \\ EOCM_{CM} & ICM_{CM} \end{pmatrix} \tag{3}$$

We divided CSM_{cm} into four sub-matrices under the constraints with $|cm.X|$ as a row size and $|cm.Y|$ as a column size, four sub-matrices are generated. Figure 3 shows four divided sub matrices from CSM_{cm}. The top-left sub matrix represents coupling relations between input events and output events in the model *cm*, which is called as $self-loop$ coupling relations. In DEVS formalism, these self-loop relations are forbidden in *CM* (Zeigler, Praehofer, and Kim 2000). Hence, we suppose that self-loop matrix ($self-loopM_{cm}$) is a zero matrix. The top-right sub matrix represents coupling relations between input events in model *cm* and output events in component models of the model *cm*, which is referred as *EIC* in the model *cm*. We could understand that the bottom-right sub matrix and the bottom-left sub matrix represent *IC* and *EOC* in the model *cm* in the similar way. Therefore, the top-right, bottom-left, and bottom-right sub matrices are identical to *EIC* matrix ($EICM_{cm}$), *IC* matrix (ICM_{cm}) and, *EOC* matrix ($EOCM_{cm}$) in the model *cm*, respectively (3).

$$
\begin{array}{c}
CM.x_1 \\ \vdots \\ CM.x_l \\ (CM.m_1).y_1 \\ \vdots \\ (CM.m_1).y_{k_1} \\ (CM.m_2).y_1 \\ \vdots \\ (CM.m_{|CM.M|}).y_{k_{|CM.M|}}
\end{array}
\overset{\begin{array}{cc} CM.y_1 \cdots CM.y_p & (CM.m_1).x_1 \cdots (CM.m_1).x_{q_1} \; (CM.m_2).x_1 \cdots (CM.m_{|CM.M|}).x_{q_{|CM.M|}} \end{array}}{\left(\begin{array}{c|c}
Self-loopM_{CM} & EICM_{CM} \\ \hline
EOCM_{CM} & ICM_{CM}
\end{array}\right)}
$$

Figure 3: Coupling Structure Matrix (CSM) divided into four sub-matrices.

3.2 Flattened Coupling Structure Matrix

FCSM of a DEVS model cm ($FCSM_{cm}$) is also a relation matrix that represents coupling relations in the flattened model of the model cm. Because the flattened model should contain only AMs as component models, $FCSM_{cm}$ illustrates coupling relations between the model cm and AMs in recursive-component models in the model cm, which indicate all leaf nodes of the decomposition tree of the model cm ($CM.AM$). Similar to defining CSM_{CM}, R_{CM} of $FCSM_{CM}$ would be a union set of $CM.X$ and $\bigcup_{i=1}^{|CM.AM|}(CM.am_i).Y$ and C_{CM} of $FCSM_{CM}$ would be a union set of $CM.Y$ and $\bigcup_{i=1}^{|CM.AM|}(CM.am_i).X$. $FCSM_{CM}$ could be mathematically defined in (4).

$$
Let\ \ FCSM_{CM}(\in \mathbb{M}(m,n))\ \ be\ \ an\ \ m\times n\ \ matrix: \tag{4}
$$

$$
R_{CM} = CM.X \bigcup \left(\bigcup_{i=1}^{|CM.AM|} (CM.am_i).Y \right), \quad C_{CM} = CM.Y \bigcup \left(\bigcup_{i=1}^{|CM.AM|} (CM.am_i).X \right)
$$

$$
CM.X = \{CM.x_{1:l}\},\ where\ \ l = |CM.X|, \quad CM.Y = \{CM.y_{1:p}\},\ where\ \ p = |CM.Y|
$$

$$
(CM.am_i).X = \left\{(CM.am_i).x_{1:q_i}\right\}, \quad where\ \ q_i = |(CM.am_i).X|\ \ for\ 1 \le i \le |CM.AM|
$$

$$
(CM.am_i).Y = \{(CM.am_i).y_{1:k_i}\}, \quad where\ \ k_i = |(CM.am_i).Y|\ \ for\ 1 \le i \le |CM.AM|
$$

$$
\therefore\ R_{CM} = \{r_{1:m}\} = \left\{CM.x_{1:l}, (CM.am_1).y_{1:k_1}, \cdots, (CM.am_{|CM.AM|}).y_{1:k_{|CM.AM|}}\right\}, \quad where\ \ m = l + \sum_{i=1}^{|CM.AM|} k_i
$$

$$
\therefore\ C_{CM} = \{c_{1:n}\} = \left\{CM.y_{1:p}, (CM.am_1).x_{1:q_1}, \cdots, (CM.am_{|CM.AM|}).x_{1:q_{|CM.AM|}}\right\}, \quad where\ \ n = p + \sum_{i=1}^{|CM.AM|} q_i
$$

Similar to defining CSM_{cm}, $self-loopM_{cm}$, $FEICM_{cm}$, $FEOCM_{cm}$, and $FICM_{cm}$ in $FCSM_{cm}$ could be defined except elements of R_{CM} and C_{CM} (4). Although elements in CSM_{cm} can be extracted from the specifications (EIC, EOC ,and IC) of model cm, elements in $FCSM_{cm}$ could not be extracted directly. Accordingly, we propose an algorithm for constructing $FCSM_{cm}$ using CSM_{cm} in the next section.

4 CONSTRUCTION OF FLATTENED COUPLING STRUCTURE MATRIX

In this section, we introduce an algorithm for constructing *FCSM* of a DEVS model. In order to handle multi-level hierarchy models, the algorithm contains a sub algorithm handling the case that the hierarchy level of a model is equal to two. In that case, *FCSM* of the model could be constructed using multiple *CSM*s of the model and its component models. However, the algorithm deals with constructing multi-level hierarchy models using the sub algorithm. In other words, the algorithm generates *FCSM* of a multi-level hierarchy model using 1) *CSM* of the model and 2) multiple *FCSM*s and *CSM*s of its component models. Moreover, The *FCSM*s of the component models are resultant of the sub algorithm, which means that the algorithm works in a recursive manner. Following subsections would explain these algorithms in detail.

4.1 Algorithm for constructing $FCSM$ with $H = 2$

In order to design an algorithm for constructing *FCSM*, it is practical to consider a simple model. Therefore, we confined our research question to generating *FCSM* of a model *cm* ($FCSM_{cm}$) with $H_{cm} = 2$. Because $FCSM_{cm}$ represents coupling structures in the flattened model of the model *cm*, we should illustrate how we construct the flattened model from the model *cm*. In order to generate the flattened model, the $flattening$ procedure, which is comprised of two processes, is required: 1) removing intermediate *CM*s in the model *cm*, and 2) redirecting coupling structures linked with the intermediate *CM*s. For removing intermediate *CM*s in the model *cm*, we could simply get rid of them from the set of component models in the *cm*. However, for redirecting coupling structures, we should consider coupling relations between the model *cm* and the intermediate *CM*s. In this paper, we introduce an algorithm that carries out the two processes and generates $FCSM_{cm}$ in Algorithm 1.

Algorithm 1 Construct$FCSM$_withH2, f_{G_FCSM}

Input: *CSMs* of *CM* and component models in *CM*

Output: $FCSM_{CM}$

Set $IntEOCM_{CM}$ to $EOCM_{CM.D_1} \oplus \cdots \oplus EOCM_{CM.D_{|CM.D|}}$; $IntEICM_{CM}$ to $EICM_{CM.D_1} \oplus \cdots \oplus EICM_{CM.D_{|CM.D|}}$; $IntICM_{CM}$ to $ICM_{CM.D_1} \oplus \cdots \oplus ICM_{CM.D_{|CM.D|}}$

Begin

procedure DEFINE_$FICM_{CM}(CSM_{CM}, CSM_{CM.D_1}, \cdots, CSM_{CM.D_{|CM.D|}})$

  $iFICM_{CM} \leftarrow IntEOCM_{CM} \cdot ICM_{CM} \cdot IntEICM_{CM}$

  $FICM_{CM} \leftarrow iFICM_{CM} + IntICM_{CM}$

end procedure

procedure DEFINE_$FEOCM_{CM}(CSM_{CM}, CSM_{CM.D_1}, \cdots, CSM_{CM.D_{|CM.D|}})$

  $FEOCM_{CM} \leftarrow IntEOCM_{CM} \cdot EOCM_{CM}$

end procedure

procedure DEFINE_$FEICM_{CM}(CSM_{CM}, CSM_{CM.D_1}, \cdots, CSM_{CM.D_{|CM.D|}})$

  $FEICM_{CM} \leftarrow EICM_CM \cdot IntEICM_{CM}$

end procedure

procedure GENERATE_$FCSM_{CM}(FEOCM_{CM}, FEICM_{CM}, FICM_{CM})$

  $(FCSM_{CM})_{11} = 0_{|CM.X|,|CM.Y|}$; $(FCSM_{CM})_{21} = FEOCM_{CM}$

  $(FCSM_{CM})_{12} = FEICM_{CM}$; $(FCSM_{CM})_{22} = FICM_{CM}$

end procedure

return $FCSM_{CM}$

End

Before detailed explanation of Algorithm 1, let us introduce a matrix operator called direct sum ($\oplus$). Let A be an "m by n" matrix and B be a "p by q" matrix. By the direct sum of A and B (A $\oplus$ B), we can get a "(m + p) by (n + q)" matrix formed as ((A 0), (0 B)). Algorithm 1 needs multiple *CSM*s of the model *cm*

and its component models in the model cm as inputs and generates $FCSM_{cm}$ as an output. Firstly, Algorithm 1 defined three matrices called as 1) Integrated EOC matrix of the model cm ($IntEOCM_{cm}$), 2) Integrated EIC matrix of the model cm ($IntEICM_{cm}$), and 3) Integrated IC matrix of the model cm ($IntICM_{cm}$). All three matrices can be defined by 1) direct sum of multiple ICMs, 2) direct sum of multiple $EOCM$s, and 3) direct sum of multiple $EICM$s (ICMs, $EOCM$s and $EICM$s come from component models in the model cm). Because $FCSM_{cm}$ consists of four sub matrices, $self-loopM_{cm}$, $FEOCM_{cm}$, $FEICM_{cm}$ and $FICMcm$, Algorithm 1 calculates each sub matrix independently, except $self-loopM_{cm}$ ($self-loopM$ is always zero matrix). Each sub matrices of $FCSM_{cm}$, except $self-loopM_{cm}$, is constructed by three sub functions in Algorithm 1. $FEOCM_{cm}$ is constructed by matrix multiplication of $IntEOCM_{cm}$ and $EICM_{cm}$ by a sub function named "construct_$FEOCM_{cm}$". $FEICM_{cm}$ is constructed by matrix multiplication of $EICM_{cm}$ and $IntEICM_{cm}$ by a sub function named "construct_$FEICM_{cm}$". $FICM_{cm}$ is also constructed by a sub function named "construct_$FICM_{cm}$". In this sub function, Intermediate $FICM_{cm}$ ($iFICM_{cm}$) matrix is a resultant matrix from matrix multiplication of $IntEOCM_{cm}$, $IntICM_{cm}$, and $IntEICM_{cm}$. Then, $FICM_{cm}$ is constructed by matrix addition of $iFICM_{cm}$ and $IntEICM_{cm}$.

4.2 Algorithm for constructing $FCSM_{cm}$ with multi-level hierarchy

In this section, we present an algorithm for constructing $FCSM$ for a multi-level hierarchy model. Particularly, the algorithm contains a sub algorithm, which is introduced in section 4.1 and works in a recursive manner.

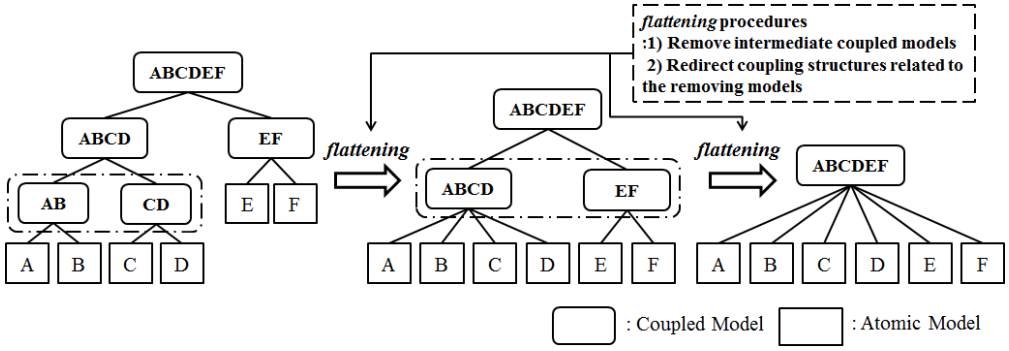

Figure 4: Constructing flattened model by a recursive manner.

Figure 4 illustrates the $flattening$ procedure for multi-level hierarchy model. In a model of multi-level hierarchy, the model needs multiple $FCSM$s (when a component model is at the coupled model level) or CSM (when a component model is at the atomic model level) of component models of the model. Therefore, the $flattening$ procedure starts from constructing multiple $FCSM$s of component models at the $(H = 2)$ using the sub algorithm. Then, $FCSM$s of component models at the level of higher hierarchy could be constructed using the $FCSM$s constructed at the previous step. This processes are repeated, until constructing $FCSM$ of the outmost coupled model. For example, Model $ABCDEF$ ($H_{ABCDEF} = 3$) has two component models, such as model $ABCD$ and EF. In order to construct $FCSM$ of $ABCDEF$, $FCSM$ of $ABCD$ should be constructed using the sub algorithm. Then, with the $FCSM$ of $ABCD$ and CSM of EF, we could construct $FCSM$ of $ABCDEF$.

Algorithm 2 describes a function named "Construct$FCSM$". In order to construct $FCSM$ of a model cm, Algorithm 2 utilizes multiple CSMs of the model cm and recursive-component models in the model cm as parameters and generate $FCSM_{cm}$ as an output. In Algorithm 2, there are three cases: 1) $H_{cm} = 1$ case, 2) $H_{cm} = 2$ case ,and 3) $H_{cm} > 2$ case (except $H_{cm} = 0$ case, because it means that the cm is an AM). If H_{cm} is equal to one, the cm has only AMs as component models. Hence, $FCSM_{cm}$ is identical to CSM_{cm}. If H_{cm} is equal to two, $FCSM_{cm}$ could be constructed by Algorithm 1. If H_{cm} is greater than two, $FCSM$

can be constructed using Algorithm 1 in a recursive manner. At first, we set $h = 2$ and define a set of models (M_h) whose height of decomposition tree in the *cm* is equal to h. Then, we can construct *FCSM* of elements in M_h (m_i) using Algorithm 1 with parameters as multiple *CSM*s of component models of m_i. After constructing all *FCSM* of all m_i, h is increased by 1 and M_h is redefined with respect to h. Similarly, we can construct *FCSM* of elements in M_h (m_i) using Algorithm 1. However, parameters, multiple *CSM*s, of this case could be changed to *FCSM*s constructed in the previous step. This recursive manner keeps until h is equal to H_{cm}. Eventually, we can obtain only one *FCSM* and the *FCSM* is identical to $FCSM_{cm}$.

Algorithm 2 Construct*FCSM* with multi-level hierarchy

Input: *CSMs* of *CM* and component models in (M) in *CM*, $M = \{m \mid H^m_{CM} > 0\}$
Output: $FCSM_{CM}$

Set h to a height of DT ; i to an index of model; M_h to a set of models with height of DT h; m_i to an i^{th} model in M_h; $m_i.D$ to a set of component models in m_i; $m_i.D_j$ to an j^{th} component model in $m_i.D$; *pFCSM* to $\{FCSM_{m_i}$ from previous stage$\mid i = 1, \cdots, |M_h|\}$; $pfcsm_m$ to an *FCSM* of model in *pFCSM*

Begin
if $H_{CM} = 1$ **then**
  $FCSM_{CM} \leftarrow CSM_{CM}$
else if $H_{CM} = 2$ **then**
  $h := 1$; Define M_h with respect to h
  $FCSM_{CM} \leftarrow f_{G\_FCSM}(CSM_{CM}, CSM_{m_1}, \cdots, CSM_{m_{|M_h|}})$
else
  $h := 2$; $pFCSM := \emptyset$
  while $h \leq H_{CM}$ **do**
    $i := 1$; Define M_h with respect to h
    while $i \leq |M_h|$ **do**
      if $H_{m_i} = 1$ **then**
        $FCSM_{m_i} \leftarrow CSM_{m_i}$
      else
        if $m* \in m_i.D$ and $FCSM_m \in pFCSM$ **then**
          $CSM_{m}* \leftarrow pfcsm_m$
        end if
        $FCSM_{m_i} \leftarrow f_{G\_FCSM}(CSM_{m_i}, CSM_{m_i.D_1}, \cdots, CSM_{m_i.D_{|m_i.D|}})$
      end if
      $pFCSM \leftarrow pFCSM \cup \{FCSM_{m_i}\}$; $i \leftarrow i + 1$
    end while
    $h \leftarrow h + 1$
  end while
end if
return $FCSM_{CM}$
End

5 CASE STUDY

In this section, we illustrate experiments to prove our algorithm is more efficient than a conventional method, i.e. naïve method. We have performed experiments for two objectives: 1) comparing simulation execution time in the non-flattened model and the flattened model and 2) comparing speed performances

of the naïve method and our algorithm. For these objectives, we developed a target model called Single Server Queuing Model (SSQ). Figure 5 shows DEVS diagrams and a decomposition tree of the SSQ model.

To address the efficiency of the flattened model and our algorithms, we have designed two experiment cases using the SSQ model. Table 3 illustrates experimental design specifications of the two experiments. In the first experiment, we have focused on comparing simulation execution time in two model types: a flattened model and a non-flattened (original) model of SSQ model by varying the hierarchy level of SSQ model. In the second experiment, we have compared efficiencies of our algorithm and a naïve method with varying hierarchy level and degree of internal nodes of the SSQ model (e.g. degree of internal nodes in Figure 5 is equal to one. When the degree is changed to two, the number of generator and transducer in the EF and buffer and processor in the BP would be twice). For the second experiment, we have defined a measurement called Flattening Speed Ratio (FSR), which represents a ratio of execution time of constructing the flattened model between using matrix and naïve methods. With the definition of FSR, FSR greater than one means the matrix method is more efficient than the naïve method.

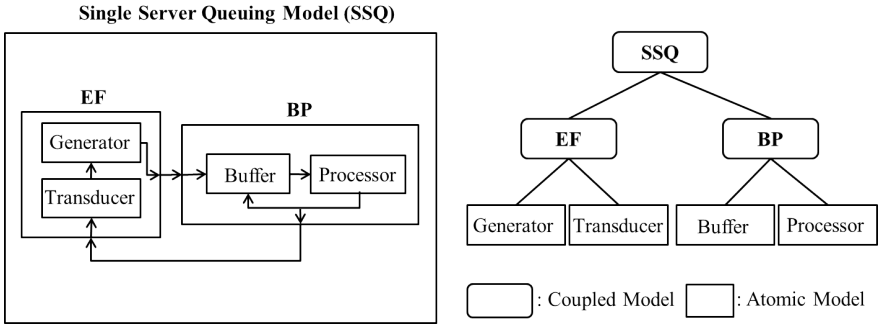

Figure 5: DEVS diagrams and a decomposition tree of SSQ.

Table 2: Specifications of two experimental designs.

Objectives	Variable	Variation Cases	Implications
Comparing simulation execution time	Height	Original, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 (11 cases)	Hierarchy level of SSQ model
	Model	Flattened or Non-Flattened (2 cases)	Simulated model types
	Total	$11 \times 2 = 22$ cases	300 replications in each case
Comparing efficiencies of algorithms	Height	4, 5, 6 (3 cases)	Hierarchy level of SSQ model
	Degree	1, 2, 3, 4 (4 cases)	Degree of internal nodes in SSQ model
	Methods	Naïve or Matrix (2 cases)	Flattening procedures
	Total	$3 \times 4 \times 2 = 24$ cases	300 replications in each case

Figure 6 shows result graphs of the two experiments. The top graph in Figure 6 represents that the flattened model shows, at most, 500 times better performance than the non-flattened model. In the top graph, simulation execution time of the non-flattened model is increasing, as the hierarchy level of the SSQ model is increasing. However, simulation execution time of the flattened model is invariable according to the hierarchy level of the SSQ model. This result means flattened models could guarantee robustness in simulation effectiveness with respect to varying the hierarchy level of models. This is because flattened models are not changed as model hierarchy of original models is varied. The left and right graph in Figure

6 represent result graphs of the second experiment. The left graph illustrates changes of FSR with respect to varying hierarchy level and degree of internal nodes in the SSQ model. The left graph represents that as the hierarchy level and the degree increases, FSR also increases. To change a perspective of the experiment, we converted the hierarchy level and the degree in the SSQ model to the number of nodes in the SSQ model. The right graph represents that the performance of our algorithm is at most 10% better than the naïve method. According to these results, we could estimate that the performance of our algorithm would increase as the number of nodes in the SSQ model increases.

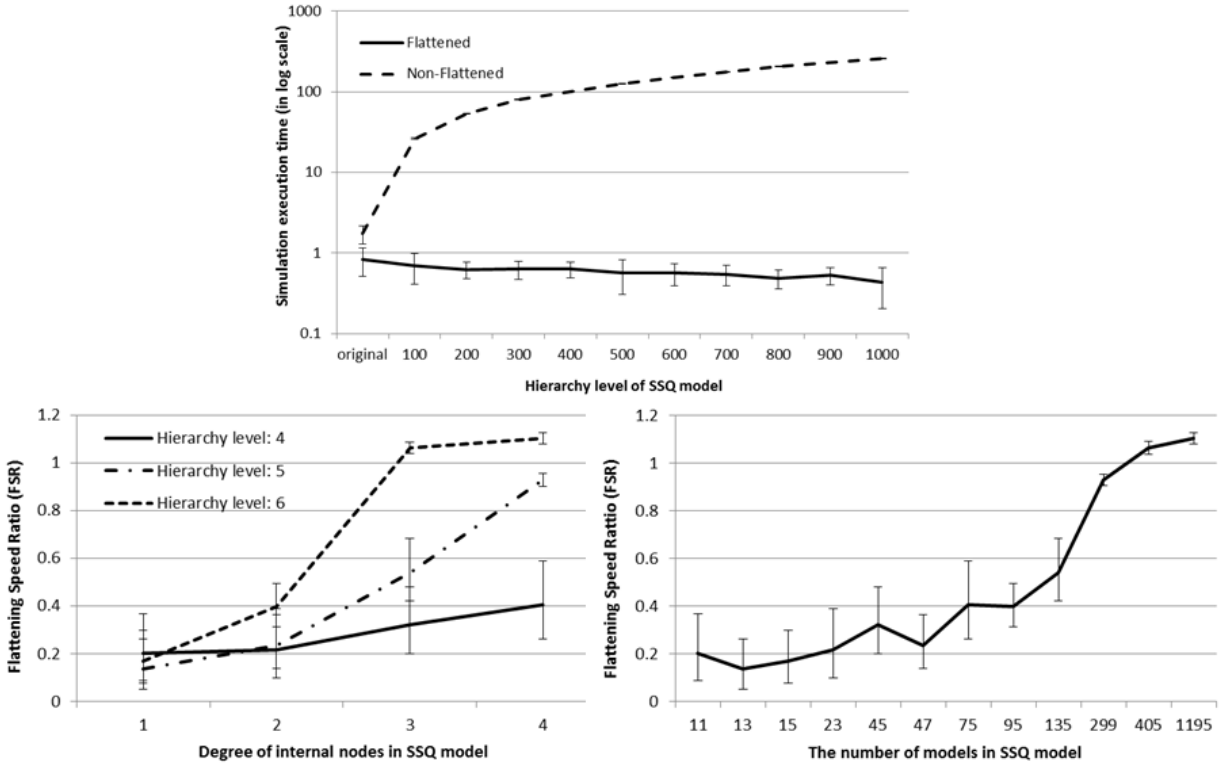

Figure 6: Result of experiments: Simulation execution time according to the height (Top), FSR according to the degree and the height (Left) and number of models (Right).

6 CONCLUSION

To improve simulation performance, most studies have utilized a flattened model. However, There are few studies about how to efficiently construct the flattened model. In this paper, we proposed a matrix-based algorithm to efficiently construct a flattened model. With the result of experiments, our algorithm shows 10% better performance than the naïve method and robustness to large-scaled models.

ACKNOWLEDGMENTS

This research was supported by the NRF of Korea funded by the MEST(20120006571).

REFERENCES

Bae, J. W., and T. G. Kim. 2010. “DEVS based plug-in framework for interoperability of simulators”. In *Proceedings of the 2010 Spring Simulation Multiconference*, 127:1–127:7. San Diego, CA, USA.

Chow, A. C.-H. 1996. "Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator". *SCS Transactions in Simulation* 13 (2): 55–67.

Goswami, K. K., and R. K. Iyer. 1993. "Use of Hybrid and Hierarchical Simulation to Reduce Computation Costs". In *Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, MASCOTS '93, 197–202. San Diego, CA, USA.

Kim, J., I.-C. Moon, and T. G. Kim. 2011, August. "New insight into doctrine via simulation interoperation of heterogeneous levels of models in battle experimentation". *SIMULATION* 88 (6): 649–667.

Kim, K., W. Kang, B. Sagong, and H. Seo. 2000. "Efficient distributed simulation of hierarchical DEVS models: transforming model structure into a non-hierarchical one". In *Simulation Symposium 2000(SS 2000) Proceedings. 33rd Annual*, 227–233. Washington, DC, USA: IEEE Computer Society 2000.

Kim, Y., and T. Kim. 1997. "Optimization of model execution time in the DEVSim++ environment". In *proc. of 1997 Eurpean Simulation Symposium, Serial. 9*, 215–219. Passau, Germany.

Kwon, S. J., and T. G. Kim. 2012. "Design and implementation of event-based DEVS execution environment for faster execution of iterative simulation". In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation*, TMS/DEVS '12, 14:1–14:8. San Diego, CA, USA.

Lee, G., N. Oh, and I.-C. Moon. 2012. "Modeling and simulating network-centric operations of organizations for crisis management". In *Proceedings of the 2012 Symposium on Emerging Applications of M&S in Industry and Academia Symposium*, EAIA '12, 13:1–13:8. San Diego, CA, USA.

Lee, J.-K., M.-W. Lee, and S.-D. Chi. 2003, August. "DEVS/HLA-Based Modeling and Simulation for Intelligent Transportation Systems". *SIMULATION* 79 (8): 423–439.

Lee, W. B., and T. G. Kim. 2003. "Simulation speedup for DEVS models by composition-based compilation". In *Summer Computer Simulation 2003*, 395–400. Montral, Canada.

Liu, Q., and G. Wainer. 2012. "Multicore acceleration of Discrete Event System Specification systems". *SIMULATION* 88 (7): 801–831.

Muzy, A., and J. Nutaro. 2005. "Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators". In *1st Open International Conference on Modeling & Simulation (OICMS)*, 401–407. ISIMA/Blaise Pascal University, France.

Palaniappan, S., A. Sawhney, and H. Sarjoughian. 2006, December. "Application of the DEVS Framework in Construction Simulation". In *Proceedings of the 2006 WinterSim*, 2077–2086. Monterey, CA.

Sargent, R. G., J. H. Mize, D. H. Withers, and B. P. Zeigler. 1993. "Hierarchical modeling for discrete event simulation". In *Proceedings of 1993 Winter simulation conference*, 569–572. New York, NY, USA.

Shanthikumar, J. G., and R. G. Sargent. 1983. "A Unifying View of Hybrid Simulation/Analytic Models and Modeling". *Operations Research* 31 (6): 1030–1052.

Troccoli, A., and G. Wainer. 1997. "Implementing Parallel Cell-DEVS". In *Proceedings of the 36th annual symposium on Simulation*, ANSS '03, 273–280. Washington, DC, USA: IEEE Computer Society.

Venkateswaran, J., Y.-J. Son, and A. Jones. 2004. "Hierarchical production planning using a hybrid system dynamic-discrete event simulation architecture". In *Proceedings of the 2004 Winter Simulation Conference*, Volume 2, 1094–1102. Washington, DC, USA: IEEE Computer Society.

Wainer, G. a., and N. Giambiasi. 2001, January. "Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation". *Simulation* 76 (1): 22–39.

Zeigler, B. P., Y. Moon, D. Kim, and G. Ball. 1997. "The DEVS environment for high-performance modeling and simulation". *Computational Science Engineering, IEEE* 4 (3): 61–71.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation*. 2 ed. Academic Press.

AUTHOR BIOGRAPHIES

JANG WON BAE is a Ph.D candidate student at KAIST. repute82@kaist.ac.kr

SU-JIN SHIN is a master candidate student at KAIST. sj-shin@kaist.ac.kr

IL-CHUL MOON is an assistant professor at KAIST. icmoon@kaist.ac.kr.