Published: 22 February 2016
Accepted: 01 November 2015
Revised: 01 November 2015
Received: 01 November 2014

RESEARCH-ARTICLE

Efficient Flattening Algorithm for Hierarchical and Dynamic Structure Discrete Event Models

JANG WON BAE, Korea Advanced Institute of Science and Technology, Daejeon, South Korea

SANG WON BAE, Kyonggi University, Suwon, Gyeonggi-do, South Korea

IL-CHUL MOON, Korea Advanced Institute of Science and Technology, Daejeon, South Korea

TAG GON KIM, Korea Advanced Institute of Science and Technology, Daejeon, South Korea

Efficient Flattening Algorithm for Hierarchical and Dynamic Structure Discrete Event Models

JANG WON BAE, Korea Advanced Institute of Science and Technology
SANG WON BAE, Kyonggi University
IL-CHUL MOON and TAG GON KIM, Korea Advanced Institute of Science and Technology

Discrete event models are widely used to replicate, analyze, and understand complex systems. DEVS (Discrete Event System Specification) formalism enables hierarchical modeling, so it provides an efficiency in the model development of complex models. However, the hierarchical modeling incurs prolonged simulation executions due to indirect event exchanges through the model hierarchy. Although direct event paths are applied to mitigate this overhead, the situation becomes even worse when a model changes its structures during simulation execution, called a dynamic structure model. This article suggests Coupling Relation Graph (CRG) and Strongly Coupled Component (SCC) concepts to improve hierarchical and dynamic structure DEVS simulation execution. CRG is a directed graph representing DEVS model structure, and SCC is a group of connected components in a CRG. Using CRG and SCC, this article presents (1) how to develop CRG from a DEVS model and (2) how to construct and update direct event paths with respect to dynamic structural changes. In particular, compared to the previous works, the proposed method focuses on the reduction of the updating costs for the direct event paths. Through theoretical and empirical analyses, this article shows that the proposed method significantly reduces the simulation execution time, especially when a simulation model contains lots of components and changes its model structures frequently. We expect that the proposed method would support the faster simulation executions of complex hierarchical and dynamic structure models.

Categories and Subject Descriptors: I.6.2 [**Modeling and Simulation**]: Simulation Types and Techniques—*Discrete-event simulation, simulation algorithm*

General Terms: Design, Algorithms, Performance, Experimentation

Additional Key Words and Phrases: Flattening algorithm, hierarchical model, dynamic structure model, DEVS, graph-based acceleration

ACM Reference Format:
Jang Won Bae, Sang Won Bae, Il-Chul Moon, and Tag Gon Kim. 2016. Efficient flattening algorithm for hierarchical and dynamic structure discrete event models. ACM Trans. Model. Comput. Simul. 26, 4, Article 25 (February 2016), 25 pages.
DOI: http://dx.doi.org/10.1145/2875356

1. INTRODUCTION

Complex systems generally consist of multiple components and intricate interactions among the components so that it is often difficult, or impossible, to describe their

Authors' addresses: J. W. Bae, Human Computing Research Section, Electronics and Telecommunications Research Institute, Daejeon, Korea; email: jwbae@etri.re.kr; S. W. Bae, Department of Computer Science, Kyonggi University, Suwon, Korea; email: swbae@kgu.ac.kr; I. C. Moon, Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea; email: icmoon@kaist.ac.kr; T. G. Kim, Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea; email: tkim@ee.kaist.ac.kr.

DOI: http://dx.doi.org/10.1145/2875356

behavior directly. In this sense, hierarchical modeling enables one to model complex systems in a bottom-up manner: (1) a complex system is decomposed into multiple components with relatively simple behaviors, (2) these components are developed as separate models, and (3) the developed components are hierarchically composed to describe the behavior of the complex system. For example, Bae et al. [2016] described courses of action in the military domain using a hierarchical model with mission (i.e., tactical level) and engagement (i.e., behavioral level) models.

On the other hand, to represent the collective behaviors in complex systems, such as self-organization and an emergence, structural dynamics among the components are also expressed in the models. Structural components in hierarchical models have been considered as passive components for the event router, but they should hold dynamics for complex systems. Hence, dynamic structure models, which change their model structures during simulation execution, have been considered in the model development of complex systems [Homer-Dixon et al. 2013; Helleboogh et al. 2007; Bar-Yam 1997].

From these perspectives, DEVS (Discrete Event System Specification) formalism is considered as a potential method to model complex systems [Quesnel et al. 2009; Wainer 2004; Vangheluwe 2000]: DEVS supports the structured modeling with its hierarchical and modular property [Zeigler et al. 2000], and several DEVS extensions, such as DSDEVS [Barros 1996], DynDEVS [Uhrmacher 2001], Mobile DEVS [Kim and Kim 2001], LDEF formalism [Bae et al. 2012], and DYS-DEVS [Muzy and Zeigler 2014], embedded dynamic structures to the classic DEVS.

Contrary to the advantages in the model development, the hierarchical modeling from DEVS might lead to longer simulation execution time. One reason for the lengthy execution is *indirect event exchanges*: an event is exchanged through extra structural components in the hierarchical structure. In fact, such time overhead was represented in many works; for example, Glinsky and Wainer [2005] show 50% increased performance on simulation execution by flattening the hierarchical model structure. Such time delay incurs serious drawbacks because simulation models should be repeatedly simulated for significant insights.

To tackle this overhead, various studies have been performed in the DEVS community. Among them, Kim et al. [2000] utilize direct paths in the event exchanges for reducing the costs from the indirect event exchanges. More specifically, they constructed direct paths before a simulation execution and performed direct event exchanges utilizing the direct paths during the simulation execution, which is called *flattening algorithm*. To build the direct paths, they applied a tree-based search algorithm to the hierarchical structure of DEVS models. Also, many researchers applied the direct event exchanges to their applications and showed remarkable performance in their works [Glinsky and Wainer 2005; Jafer and Wainer 2009; Chen and Vangheluwe 2010; Zacharewicz et al. 2010].

Nonetheless, the situation is changed when the model structure is changed during simulation execution, that is, dynamic structure models. To update the direct paths responding to structural changes, the previous methods have no choice but to reconstruct all of them from scratch (see the top of Figure 1). Assuming extreme model conditions, for example, a model with a high-level hierarchy and frequent dynamic structural changes, the previous works can show a worse performance than the classic DEVS algorithm due to the heavy costs of reconstructing direct paths. Hence, to apply the flattening algorithm to hierarchical and dynamic structure models, another method that considers an efficient management on the direct paths is required.

To this end, this article proposes a flattening algorithm for the construction and the dynamic updates of the direct paths. Figure 1 represents how the proposed algorithm handles dynamic structural changes compared to the previous studies. When a structural change occurs, the proposed algorithm identifies a part of the direct paths to be

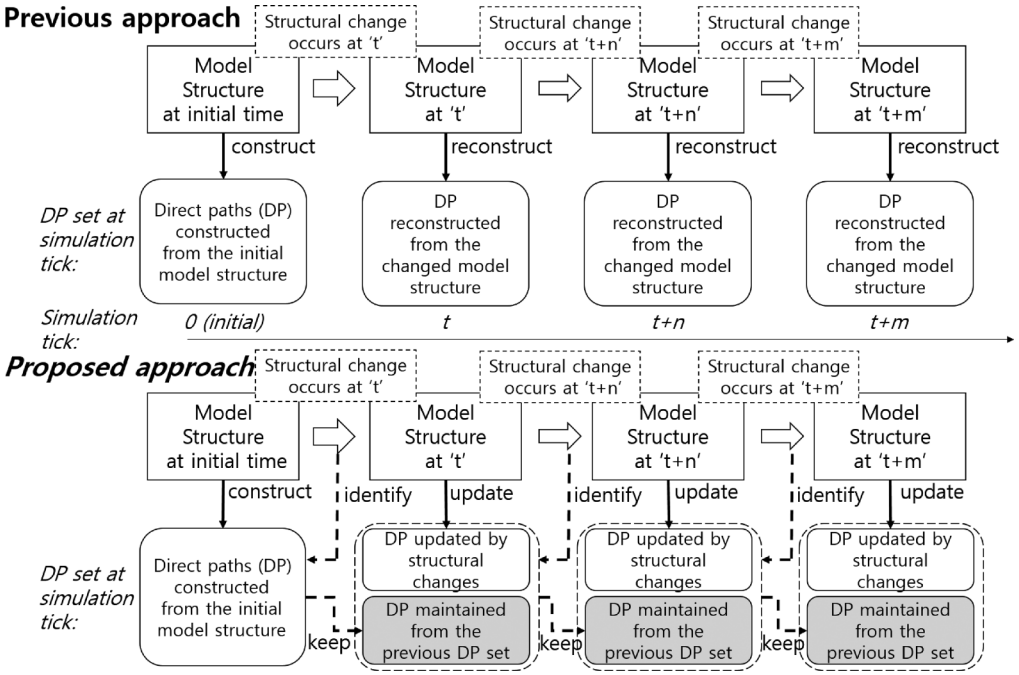

Fig. 1. Constructing and updating the direct paths for event exchanges in hierarchical and dynamic structure models: comparison of the previous and the proposed approach.

reused and updates other direct paths with respect to the structural changes. Hence, by combining reused and updated ones, the next direct paths are constructed. To allow this approach, we (1) propose a graph-based structure, called *Coupling Relation Graph* (CRG), for representing the current model structure and (2) identify grouped components in CRG, called *Strongly Coupled Components* (SCCs), for efficiently handling dynamic structural changes.

The main contribution of this article is that, to the best of our knowledge, it is the first research for the faster simulation execution of hierarchical and dynamic structure discrete event models applying graph theory. Moreover, graph theory has been widely adopted in DEVS-based modeling and simulation, such as model generation [Traoré 2009; Posse et al. 2003], model verification [Byun et al. 2009; Saadawi and Wainer 2009; Hwang 2005], and model allocations in a parallel and distributed simulation [Adegoke et al. 2013; Hu et al. 2005], but not for the faster simulation execution.

To show the efficiency of the proposed method, we carried out two kinds of performance analyses: time complexity analyses and simulation execution time analyses from two example models. These analyses represent that the proposed method efficiently reduces the simulation execution time compared to the traditional methods, particularly when a model contains lots of components and frequent structural changes. We expect that the proposed method would support the faster simulation executions of models describing complex systems.

2. BACKGROUNDS

As DEVS is widely applied to modeling various complex systems, many studies for the speedup of DEVS simulation have been carried out so far. In particular, this section presents the inefficiency of the previous works in simulating dynamic structure models.

Table I. Previous Studies for the Speedup of DEVS Simulation Execution with Four Categorizations: Enhancing Machine Environment, Applying Analytic Models, Reconstructing Coupling Relations, and Modifying Simulation Algorithms

Categories	Detailed Approach	Research Cases
Enhancing machine environment	Parallel and distributed system (PADS)	DEVS simulation environment on PADS [Himmelspach et al. 2007; Wainer 2000; Zeigler and Zhang 1990]
		DEVS simulation environment on GPU [Guo et al. 2012; Liu and Wainer 2010; Maggio et al. 2009]
		VECDEVS for parallel simulation [Bergero and Kofman 2014]
Applying analytic models	Hybrid simulation	HDEVS formalism [Pranevicius et al. 2011; Lim and Kim 2001]
		Hybrid simulation environment [Bergero et al. 2013; Alvanchi et al. 2011; Bae and Kim 2010]
Modifying simulation algorithms	Cellular DEVS model	Simulation engine for large-scale cellular DEVS models [Hu and Zeigler 2004]
	Event-oriented paradigm	A-DEVS [Muzy and Nutaro 2005]
		E-DEVSim++ [Kwon and Kim 2012]
	Distributed simulation	Distributed DEVS simulator [Syriani et al. 2011]
Reconstructing coupling relations	Model compilation	Composition-based compilation [Lee and Kim 2003]
	Direct paths for event exchanges	Transforming DEVS to nonmodular form [Shiginah and Zeigler 2011]
		DEVS cluster [Kim et al. 2000]
		Cell-DEVS [Wainer and Giambiasi 2001]
		Coupling Relation Graph (proposed method)

2.1. Previous Studies for Faster DEVS Simulation

To compensate for the prolonged simulation execution, there has been much research on the speedup of DEVS simulation. Table I summarizes such research categorized into four parts: enhancing the machine environment, applying analytic models, reconstructing coupling relations, and modifying simulation algorithms.

Enhancing the machine environment means executing DEVS simulation on a computer system containing multiple processors interconnected by a communication network. Since the early 1990s, parallel and distributed systems (PADS) have been recognized as a resolution to reduce simulation execution time [Fujimoto 1999]. A PADS subdivides a large simulation model into many subproblems and executes them concurrently. DEVS simulation environments applying PADS methods have been developed on a network system [Himmelspach et al. 2007; Wainer 2000; Zeigler and Zhang 1990] and GPU [Guo et al. 2012; Liu and Wainer 2010; Maggio et al. 2009]. Recently, vectorial DEVS (VECDEVS) for parallel simulation was proposed [Bergero and Kofman 2014].

Hybrid modeling and simulation, which develops a simulation model containing multiple analytic models and simulation models simultaneously, is considered as a method for the speedup of DEVS simulation executions. It originated from the fact that faster solutions of analytic models can accelerate the overall simulation execution [Goswami and Iyer 1993; Shanthikumar and Sargent 1983]. Lim and Kim [2001] invented a formal specification for hybrid modeling and simulation, called HDEVS formalism. Also, several hybrid simulation environments have been developed in single [Bae and Kim

2010] and multiple computing systems [Bergero et al. 2013; Alvanchi et al. 2011]. The previous works, however, were not used much because it is difficult to find their appropriate applications.

On the other hand, some researchers reduced time overheads from the indirect event exchanges by removing hierarchical and modular aspects in the DEVS abstract simulators. Hu and Zeigler [2004] developed a simulation engine for large-scale cellular DEVS models by observing the active cells in cellular DEVS models. Muzy and Nutaro [2005] modified the abstract simulators by a simple protocol for event exchanges and time synchronization in A-DEVS, Kwon and Kim [2012] adopted an event-oriented paradigm to the abstract simulators, and Syriani et al. [2011] optimized the abstract simulators for distributed simulations. These studies showed considerable improvements in their works, but the advantages from the hierarchical and modular form are hardly retained.

To accelerate simulation executions without modifying DEVS abstract simulators, several researchers have focused on reconstructing the coupling relations in DEVS models. Lee and Kim [2003] suggested a composition-based compilation method that transforms the outmost coupled model into a single AM by synthesizing model specifications of the component models. Also, many researchers [Kwon and Kim 2012; Shiginah and Zeigler 2011; Muzy and Nutaro 2005; Wainer and Giambiasi 2001; Kim et al. 2000] constructed direct paths of event exchanges in a DEVS model, which will be called *Flattened Relations* (FRs) in this article.

This article also belongs to the category of reconstructing coupling relations but considers efficient FR updates in dynamic structure models, which is in contrast with the previous research. Moreover, from now on, event exchange algorithms from Zeigler et al. [2000], Muzy and Nutaro [2005], and Kim et al. [2000] are called *ClassicDEVS*, *Tree*, and *TreeFR* so that they are compared with the proposed method in the experiments.

2.2. Flattened Relations in Dynamic Structure Models

Before introducing the proposed algorithm, we provide a detailed explanation about FR for understanding the motivation of this article. While FR enables faster DEVS simulation, it requires additional costs for the initial construction and update (especially in dynamic structure models). Thus, how to construct and update FR is crucial to simulation execution time. In this section, we exemplify one of the previous methods in detail and present its inefficiency in updating FR.

Many researchers have exploited FR and showed its efficiency [Kim et al. 2000; Glinsky and Wainer 2005; Jafer and Wainer 2009; Chen and Vangheluwe 2010; Zacharewicz et al. 2010], and their approaches are based on the work of Kim et al. [2000]. Even though the precise algorithm was not provided in their paper, it is possible to estimate how they construct FRs from DEVS specifications: FRs are established by tracking an event path from an output event of an atomic model to an input event of an input model through the model hierarchy.

Figure 2 shows an example of the TreeFR method. Before simulation execution (at simulation tick:0), the TreeFR method builds the FR of model *ABCD*. For example, from the model specifications, we see that an event *out* generated from model *A* (*A.out*) is delivered to an event *in* of model *C* (*C.in*) through the model hierarchy, that is, (*AB.out*1) and (*CD.in*1). Hence, one FR ((*A.out*), (*C.in*)) is established. To construct all FRs for model *ABCD*, this method should be applied to all output events of all component AMs (i.e., *A*, *B*, *C*, and *D*).

However, the TreeFR method reveals its limitation in dynamic structure models. Dynamic structure models can partially change their model structures during the simulation execution [Zeigler and Ören 1986; Zeigler and Sarjoughian 2013]. For example, let us assume that a part of the coupling relations in model *ABCD* is changed at

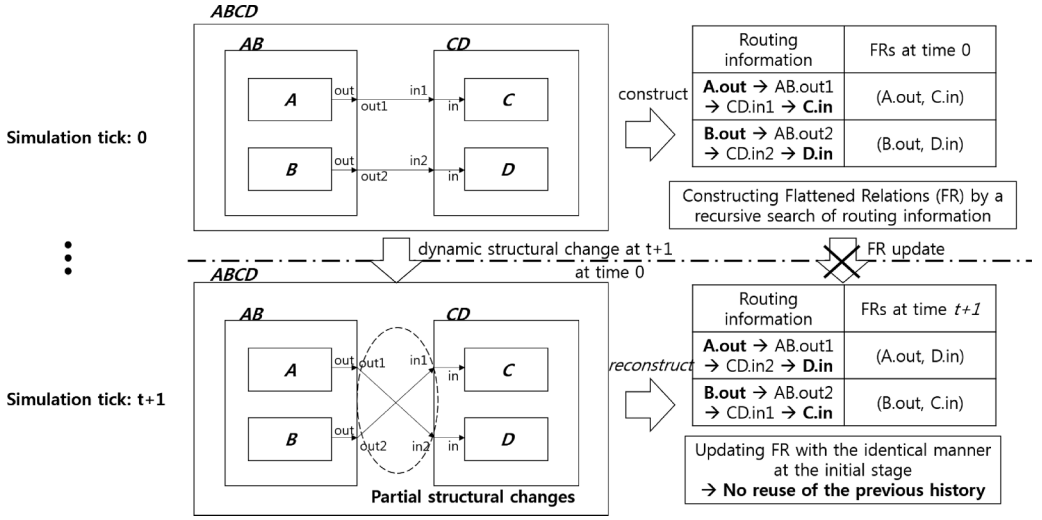

Fig. 2. Inefficiency of TreeFR method on updating flattened relations in a dynamic structure model: for any structural changes, an FR associated from an output event *out* of model *A* (i.e., *A.out*) is only updated by reconstructing the whole FR.

simulation time $t + 1$ (see Figure 2). To reflect partial structural changes to FRs, the TreeFR method should reconstruct all FRs with the same manner as the initial construction. Accordingly, when a model changes its structure frequently, the update cost would exceed time gains from the direct event exchanges so that the overall simulation execution takes a longer time than the classic DEVS algorithm.

In a dynamic structure model, the cost for FR updates is critical to the overall execution time. For efficient FR updates, this article suggests a directed graph structure for representing model structure, called CRG, and identifies grouped components in CRG, called SCCs. The proposed method identifies reusable FRs using CRG and its SCCs so that it encourages efficient updates of the next FRs by utilizing the reusable FRs.

Figure 3 illustrates event exchange algorithms from the traditional (e.g., classicDEVS, Tree, and TreeFR) and the proposed method. Also, their expected costs are departmentalized into (1) event exchange costs and (2) constructing and (3) updating FR costs. Note that although classicDEVS and Tree do not consider dynamic structures, we can estimate their expected costs about FRs with their event exchange mechanisms.

In the case of event exchanges, ClassicDEVS and the Tree would require much time because they utilize the indirect event exchanges. Instead, they did not need more costs for constructing and updating FRs. On the other hand, both TreeFR and the proposed method would show an efficiency in event exchanges due to utilizing FRs. However, in the case of constructing and updating FRs, the proposed method is expected to take less time than TreeFR by the partial FR updates. To sum up, the individual costs from the proposed method are not always the lowest, yet the sum of them would be lower than other algorithms, which will be shown in the experimental results.

3. CONSTRUCTING FLATTENED RELATIONS USING COUPLING RELATION GRAPH

Utilizing FRs would reduce the time overhead from the indirect event exchanges. However, the extra costs for constructing and updating FRs are required. This section introduces the concepts of *Coupling Relation Graph* and *Strongly Coupled Components* and presents how to efficiently develop and manage FRs using them.

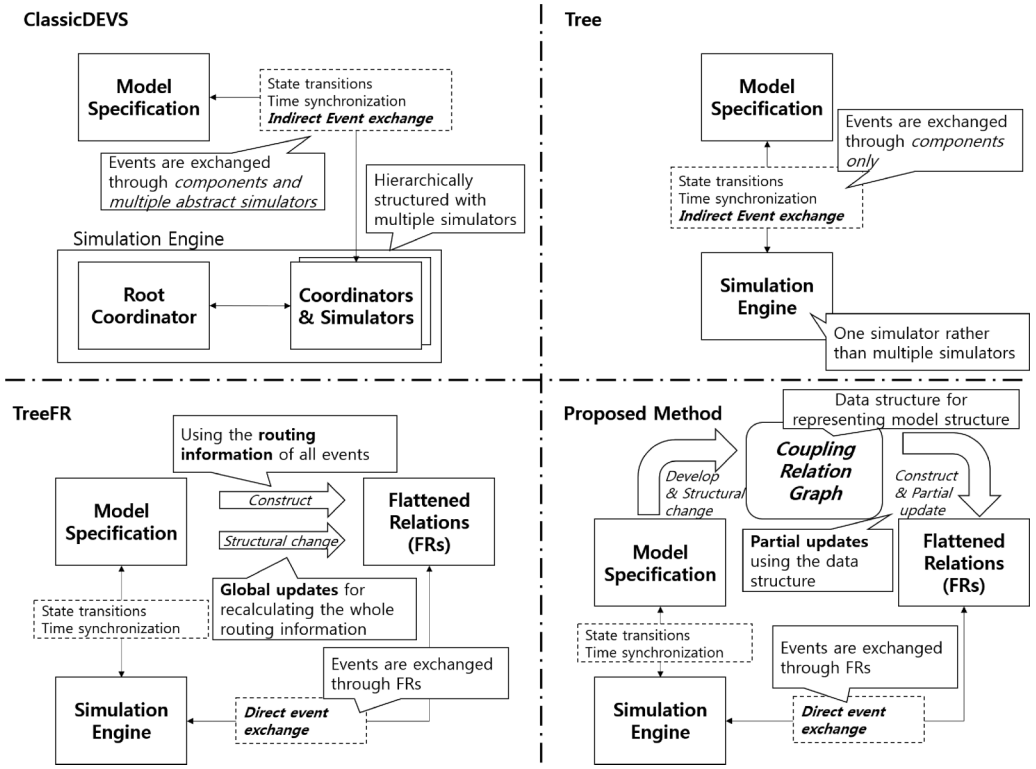

Fig. 3. Conceptual procedures on event exchanges from ClassicDEVS, Tree, TreeFR, and the proposed method.

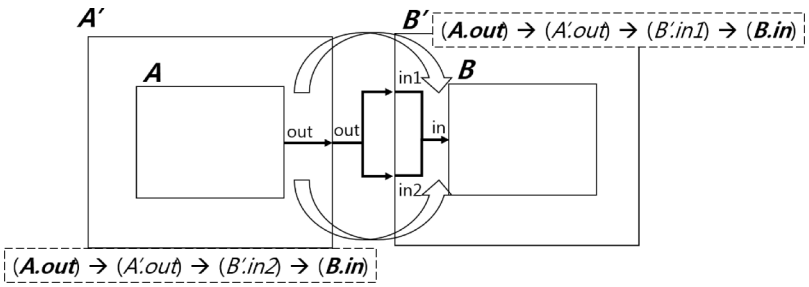

Fig. 4. Exception of the proposed method: there are two event paths between an output event *out* of model *A* (i.e., (*A.out*)) and an input event *in* of model *B* (i.e., (*B.in*)).

Before introducing the details, let us explain an assumption of the proposed method. The proposed method does not consider a model that holds more than two event paths between an output event of an atomic model and an input event of another atomic model (see the example in Figure 4).

The assumption is established for the following reasons: (1) In DEVS, structural and behavioral parts are separated and developed as independent models, that is, the coupled model and atomic model. The two paths between an output and an input event indicate that the output event is forwarded to the input event two times, but we consider that "sending an event two times" is a part of the model behavior, not the model structure. Hence, "sending an event two times" should be specified at the atomic

model level. (2) If the two paths are available, it would give rise to confusion in model reuses because the model reuses encouraged by DEVS are based on the complete model separations. (3) Lastly, we note that those two paths have never been considered in the previous flattening algorithms.

3.1. Coupling Relation Graph

The *Coupling Relation Graph of a model M* (CRG_M) is a directed graph that represents the overall coupling relations in M and its descendant components at the event level. The concept of *descendant components of model M* indicates either components in M or a component of M's descendant component. Using the DEVS specifications, the descendant component is more precisely specified: given a coupled model ($CM = <X, Y, D, \{M_i\}, EIC, EOC, IC, SELECT>$) [Zeigler et al. 2000], the descendant component set of model CM (dM_{CM}) is formally defined as follows:

$$dM_{CM} = \begin{cases} \{M_i\} \cup \bigcup_{i \in D} dM_{M_d}, & \text{if } \{M_i\} \neq \emptyset \\ \emptyset & \text{otherwise.} \end{cases}$$

A vertex of CRG is defined as a pair of a descendant component model and its input or output event. For instance, if a model A holds an input event in, the corresponding vertex is represented as (A, in) or simply $A.in$. An edge of CRG represents a relation between vertices in CRG.

Considering the DEVS specifications, the vertices of CRG_M ($V(CRG_M)$) are separated into in/out events of descendant component AMs (V_{ai} and V_{ao}) and CMs (V_{ci} and V_{co}) of M. The edges of CRG_M ($E(CRG_M)$) signify the coupling relations from M and its coupled descendant components (see Definition 3.1).

Definition 3.1 (*Coupling Relation Graph of DEVS model M* (CRG_M)). CRG_M is a directed graph:

$$CRG_M =< V, E >,$$

where $V = \{(m, e) \mid m \in dM_M$ and $e \in X$ or Y in $m\}$, and $E = \{(u, v) \mid u$ and $v \in V\}$.
In DEVS terms, $V = \{V_{ai} \cup V_{ao} \cup V_{ci} \cup V_{co}\}$ and $E = \{E_{eic} \cup E_{eoc} \cup E_{ic}\}$.
$V_{ai} = \{(am, x) \mid am$ is an atomic descendant component of M, and x is its input event.
$V_{ao} = \{(am, y) \mid am$ is an atomic descendant component of M, and y is its output event$\}$.
$V_{ci} = \{(cm, x) \mid cm$ is a coupled descendant component of M, and x is its input event$\}$.
$V_{co} = \{(cm, y) \mid cm$ is a coupled descendant component of M, and y is its output event$\}$.
$E_{eic} = \{((m, x), (n, y)) \mid (m, x) \in V_{ci}, (n, y) \in V_{ci} \cup V_{ai}$, and n is a component of $m\}$. $E_{eoc} = \{((m, x), (n, y)) \mid (m, x) \in V_{co} \cup V_{ao}, (n, y) \in V_{co}$, and m is a component of $n\}$.
$E_{ic} = \{((m, x), (n, y)) \mid (m, x) \in V_{co} \cup V_{ao}, (n, y) \in V_{ci} \cup V_{ai}$, and m, n are components of a common model$\}$.

Vertices of CRG_M ($V(CRG_M)$) are distinguished into three types from the perspective of graph theory: *source* (with no incoming edge), *sink* (with no outgoing edge), and *internal vertex* (with incoming and outgoing edges). From this separation, the properties of CRG are also induced (see Lemma 3.2).

LEMMA 3.2. *For any DEVS model M, the following are always true:*

(1) CRG_M *is a directed and acyclic graph (DAG).*
(2) $v \in V(CRG_M)$ *is a source vertex if and only if* $v \in V_{ao}$.
(3) $v \in V(CRG_M)$ *is a sink vertex if and only if* $v \in V_{ai}$.

PROOF.

(1) Because a coupling relation in DEVS indicates a direction between two events, all edges in CRG_M hold directions. Thus, CRG_M is a directed graph. If an edge (u, v)

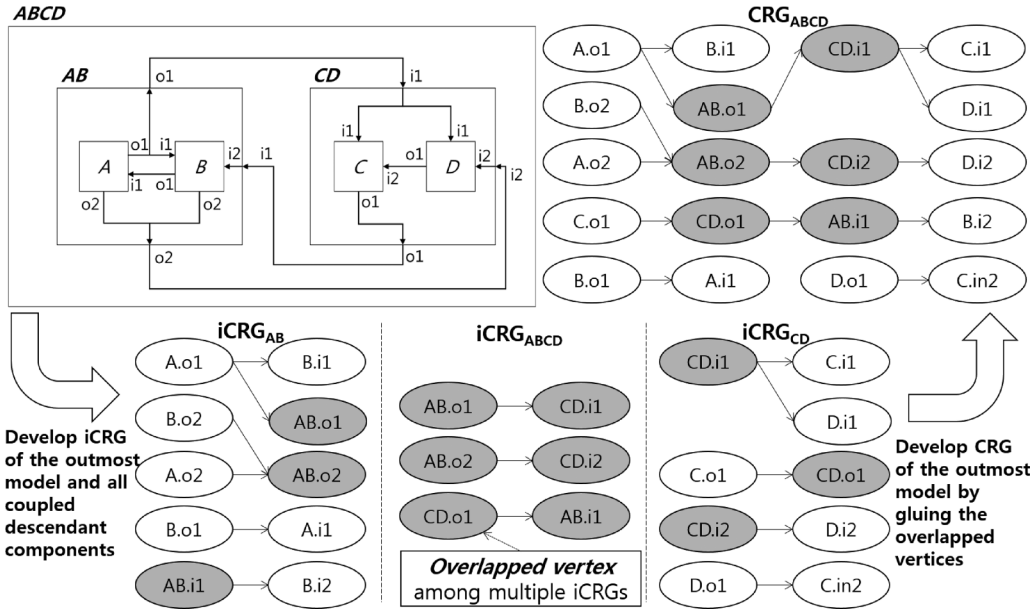

Fig. 5. Development of Coupling Relation Graph of model *ABCD* (CRG_{ABCD}): (1) developing *iCRG*s of the outmost model and all coupled descendent components with their coupling relations, (2) identifying the overlapped vertices among the multiple *iCRG*s, and (3) constructing CRG_{ABCD} by gluing the overlapped vertices in *iCRG*s.

is in E_{eic}, an edge (v, u) cannot be in E_{eoc} by the definition of E_{eic} and E_{eoc}, and vice versa. Also, if an edge (u, v) is in E_{ic}, an edge (v, u) cannot be in E_{ic} by the definition of E_{ic}. Thus, there is no cycle in CRG_M.

(2) Vertices in V_{ao} have outgoing edges and no incoming edge, because output events of atomic models are the starting points for any coupling relations. Also, a vertex of CRG_M with no incoming edge must belong to V_{ao} in CRG_M. Therefore, by the definition of a source vertex, any source vertex in CRG_M should be vertices in V_{ao}.

(3) Vertices in V_{ai} have incoming edges and no outgoing edge, because input events of atomic models are the endpoints for any coupling relations. Also, a vertex of CRG_M with no outgoing edge must belong to V_{ai} in CRG_M. Therefore, by the definition of a sink vertex, any sink vertex in CRG_M should be vertices in V_{ao}. □

Let us illustrate how to construct a CRG of a DEVS model with an explicit example (see Figure 5). Model *ABCD* in Figure 5 holds *AB*, *CD*, *A*, *B*, *C*, and *D* as its descendant components (i.e., $dM_{ABCD} = \{AB, CD, A, B, C, D\}$). To construct CRG_{ABCD}, the coupled descendant components need to first be discovered. In the example, model *AB* and *CD* are the coupled descendant components. Then, we develop *intermediate CRGs* (iCRG) of the coupled descendant components and the outmost coupled model.

While CRG reflects the global coupling relations of both the associated model and its descendant components, iCRG represents coupling relations of the associated model only. Therefore, the structure of iCRG is similar to that of CRG, but the path lengths in iCRG should be one. In the example figure, $iCRG_{AB}$, $iCRG_{CD}$ (as the descendant components), and $iCRG_{ABCD}$ (as the outmost coupled model) are presented.

The reason we separate iCRG from CRG is that iCRG enables us to construct CRG in an incremental manner; more specifically, when we have multiple iCRGs, we are enabled to construct a CRG with identifying overlapped vertices among the iCRGs. In the example figure, a vertex $AB.o1$ is included in both $iCRG_{AB}$ and $iCRG_{ABCD}$, and

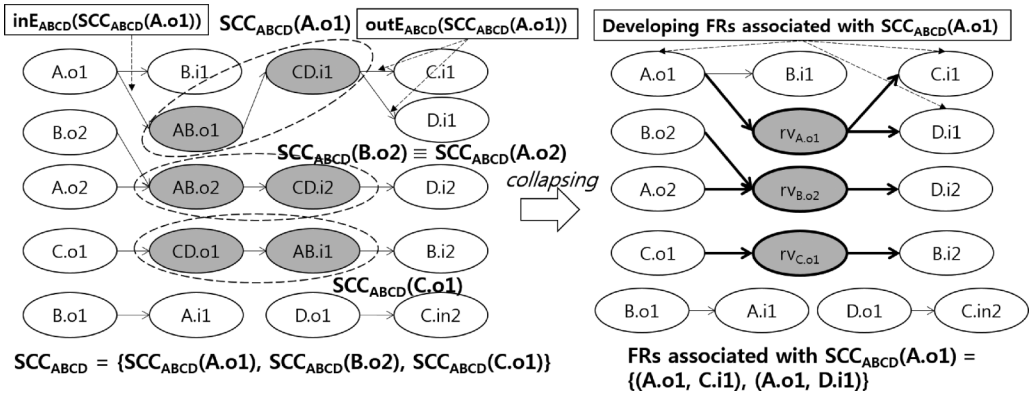

Fig. 6. Development of Strongly Coupled Components (SCC_{ABCD}) and their bridges in CRG_{ABCD} (left); identification of Flattened Relations (FRs) using the developed SCCs and bridges.

such a vertex is called an overlapped vertex (see grayed vertices in Figure 5). By gluing the overlapped vertices, multiple iCRGs (e.g., $iCRG_{AB}$, $iCRG_{CD}$, and $iCRG_{ABCD}$) are connected into one larger graph, and the resultant graph is exact to CRG_{ABCD}.

Before introducing how to construct FRs from CRG, we explicitly define the FR of model M using CRG terms (based on Lemma 3.2 (2) and (3)):

Definition 3.3 (*Flattened Relations in model M* (FR_M)).

$$FR_M = \{(u, v) \mid u \in V_{ao} \text{ and } v \in V_{ai}, \text{ and there is a path from } u \text{ and } v \text{ in } CRG_M\}.$$

From the definition of FR, it is clear that path-finding algorithms can be applied to develop FRs because there is no cycle in CRG (Lemma 3.2 (1)). Path-finding algorithms are exemplified by breadth-first search and depth-first search [Gersting 2007] and the Floyd-Warshall algorithm [Cormen et al. 2001]. However, these algorithms could be infeasible, because their costs exponentially increase by the number of vertices in a graph, and, moreover, the CRG of complex systems would have a large number of vertices and edges. Hence, to apply these algorithms to CRG, one must reduce the number of vertices.

3.2. Strongly Coupled Components

One method to reduce the number of vertices in a graph is collapsing multiple vertices into a single vertex. Moreover, coupling relations at the high-level hierarchy is potentially involved in multiple event paths so that collapsing them would lead to an efficiency in the path finding. To this end, we suggest the concept of *Strongly Coupled Components* defined as follows:

Definition 3.4 (*Strongly Coupled Components in* CRG_M (SCC_M)). $SCC_M(u) = \{v \mid v$ is an internal vertex in CRG_M reachable from a source vertex $u\}$.
Denote by SCC_M the set of nonempty strongly coupled components $SCC_M(u)$ for all source vertices $u \in V_{ao}$; that is, $SCC_M = \{SCC_M(u) \mid u \in V_{ao}\} \setminus \emptyset$.

Figure 6 illustrates an example of the development of SCC_{ABCD} from CRG_{ABCD}. To construct SCC_{ABCD}, candidate SCCs in CRG_{ABCD} should be identified by applying path-finding algorithms to the source vertices, which are $SCC_{ABCD}(A.o1)$, $SCC_{ABCD}(B.o2)$, $SCC_{ABCD}(A.o2)$, and $SCC_{ABCD}(C.o1)$. Among these four SCCs, $SCC_{ABCD}(A.o2)$ would be excluded from the SCC_{ABCD} because the vertices of $SCC_{ABCD}(A.o2)$ are identical to those of $SCC_{ABCD}(B.o2)$ (by Definition 3.4).

Definition 3.5 (*Bridges of SCC in* CRG_M ($B_M(SCC)$)).

$$B_M(SCC) = \langle inE_M(SCC), outE_M(SCC) \rangle,$$

where $inE_M(SCC) = \{(u, v) \mid u \in V_{ao}(CRG_M)$ and $v \in SCC$, and $(u, v) \in E(CRG_M)\}$ $outE_M(SCC) = \{(u, v) \mid u \in SCC$ and $v \in V_{ai}(CRG_M)$, and $(u, v) \in E(CRG_M)\}$.

Also, we defined *bridges of an SCC in* CRG_M ($B_M(SCC)$), where inE ($inE_M(SCC)$) and outE ($outE_M(SCC)$) are sets of the edges between (1) source and sink vertices in CRG_M and (2) the vertices in the SCC (see Definition 3.5). Using SCCs and their bridges, the number of vertices and edges of the CRG can be reduced: the vertices of an SCC are replaced with a single vertex, called the *representative vertex* (rv); the associated edges are replaced with new edges that are generated from the bridges by replacing its vertex in the SCC with rv.

In Figure 6, $SCC_{ABCD}(A.o1)$ and its bridges (i.e., $B_{ABCD}(SCC_{ABCD}(A.o1))$) are computed as $\{AB.o1, CD.i1\}$ and $\langle\{(A.o1, AB.o1)\}, \{(CD.i1, C.i1), (CD.i1, D.i1)\}\rangle$, respectively. Then, the vertices in $SCC_{ABCD}(A.o1)$ are replaced with a new single vertex, named $rv_{A.o1}$, and the associated edges of $rv_{A.o1}$ are derived from the bridges of the replaced SCC ($B_{ABCD}(SCC_{ABCD}(A.o1))$), such as $(A.o1, rv_{A.o1})$, $(rv_{A.o1}, C.i1)$, and $(rv_{A.o1}, D.i1)$. Then, FRs generated from an SCC are calculated by identifying pairs of (1) source vertices from incoming edges and (2) sink vertices from outgoing edges of the associated rv (see Figure 6).

By collapsing the CRG, we can exploit two advantages in constructing FRs: (1) avoiding repeated computations and (2) managing dynamic structural changes efficiently. Observe that, in Figure 6, by replacing $SCC_{ABCD}(B.o2)$ with $rv_{B.o2}$, the costs for the direct path from $A.o2$ are not required. The second advantage would be explained in the next section. The overall procedure for constructing FR from the CRG is described in Algorithm 1.

3.3. Time Complexity Analysis on Initial Construction of FR

Algorithm 1 uses a path-finding algorithm, such as depth-first search (DFS), to discover SCCs and FRs. Also, the TreeFR method applies a path-finding algorithm when constructing FRs. It means that both TreeFR and the proposed method have the identical asymptotic behavior in the worst-case time complexity. Hence, we take a deeper look into those methods to prove that the proposed algorithm outperforms TreeFR in the FR construction.

In the TreeFR method, DFS is applied to all output events of the atomic component models when constructing FRs for all event exchanges. It is well known that the time complexity of DFS is proportional to the number of vertices and edges of the searched graph.

Let T_{TreeFR} be the worst-case time complexity of the TreeFR method. Also, we define V_{ao} as the set of output events in the atomic component models, V as the set of input/output events in all the component models, and E as the set of coupling relations in all the component models. We then have the following equation:

$$T_{TreeFR} = O(|V_{ao}|(|V| + |E|)). \tag{1}$$

On the other hand, the proposed Algorithm 1 constructs FRs through three phases: *constructing CRG*, *identifying SCCs*, and *developing FR*. Hence, the time complexity of the proposed algorithm ($T_{Proposed}$) is represented by their summation:

$$T_{Proposed} = T_{CRG} + T_{SCC} + T_{FR}. \tag{2}$$

To construct the CRG, all events and coupling relations in a DEVS model should be traversed, so the time complexity of constructing CRG (T_{CRG}) is as follows:

$$T_{CRG} = O(|V| + |E|). \tag{3}$$

ALGORITHM 1: Construct Flattened Relations Using Coupling Relation Graph

Input: Coupling Relation Graph of DEVS model M, CRG_M
Output: Flattened Relations of model M, FR_M
Compute SCC_M
Let RV_M be the set of representative vertices
Let V_S and V_T be temporary containers for vertices
forall the *outgoing edge* (u, v) *from* $u \in V_{ao}(CRG_M)$ **do**
  if $v \in RV_M$ **then**
    continue
  end
  else if v *is a sink vertex* **then**
    Add (u, v) to FR_M
  end
  else if v *is an internal vertex* **then**
    Develop $B_M(SCC_M(u))$ as $\langle inE_M(SCC_M(u)), outE_M(SCC_M(u))\rangle$
    Remove $V(SCC_M(u))$ and the associated edges from $V(CRG_M)$ and $E(CRG_M)$
    Add a new vertex, rv_u, to $V(CRG_M)$, and add rv_u to RV_M
    forall the *edges* $(m, n) \in inE_M(SCC_M(u))$ **do**
      Add (m, rv_u) to $E(CRG_M)$
      Store m to V_S
    end
    forall the *edges* $(m, n) \in outE_M(SCC_M(u))$ **do**
      Add (rv_u, n) to $E(CRG_M)$
      Store n to V_T
    end
    forall the $s \in V_S$ *and* $v \in V_T$ **do**
      Add (s, v) to FR_M
    end
    clear V_S and V_T
  end
end

To identify SCCs and their bridges, DFS is applied to all source vertices in the constructed CRG. Hence, the time complexity of identifying SCC (T_{SCC}) is presented as follows:

$$T_{SCC} = O(|V_{ao}|(|V_{CRG}| + |E_{CRG}|)). \tag{4}$$

Note that, during identifying SCCs, the proposed algorithm reuses the identified SCCs: the proposed algorithm collapses CRG using the rv and the bridges of SCC (B(SCC)). Through the collapsed CRG, the searching scope in the CRG would be reduced (see Equation (5)).

When a CRG_M is identified, the vertices, the edges, and the identified SCCs in the CRG_M are also discovered as $V(CRG_M)$, $E(CRG_M)$, and SCC_M, respectively. Then, the vertices and the edges in the collapsed CRG_M ($V(C\_CRG_M)$ and $E(C\_CRG_M)$) are represented as follows:

$$\begin{gathered} V(C\_CRG_M) = (V(CRG_M) - \cup_{scc \in SCC_M} scc) \cup RV_M \\ E(C\_CRG_M) = (E(CRG_M) - rE_M) \cup E(RV_M), \\ \text{where } RV_M = \{v \mid v \text{ is an } rv \text{ associated with } SCC, \text{ and } SCC \in SCC_M\}, \\ rE_M = \{(u, v) \mid u, v \in E(CRG_M), \text{and either } u \text{ or } v \in SCC, \text{and } SCC \in SCC_M\}, \\ E(RV_M) = \{e \mid e \text{ is an edge incident to m} \in RV_M\}. \end{gathered} \tag{5}$$

Thus, the time complexity of T_{SCC} is reorganized using $V(C\_CRG_M)$ and $E(C\_CRG_M)$:

$$T_{SCC} = O(|V_{ao}|(|V(C\_CRG_M)| + |E(C\_CRG_M)|)). \tag{6}$$

Lastly, the FR of a DEVS model is directly developed using the bridges of SCCs, which means the time complexity of developing FR is $O(1)$. Hence, the total time complexity ($T_{Proposed}$) is as follows:

$$\begin{aligned} T_{Proposed} &= T_{CRG} + T_{SCC} + T_{FR} \\ &= O(|V| + |E|) + O(|V_{ao}|(|V(C\_CRG_M)| + |E(C\_CRG_M)|)) + O(1). \end{aligned} \tag{7}$$

By the definition of CRG (see Definition 3.1), the size of vertices and edges of model M is the same as that of $V(CRG_M)$ and $E(CRG_M)$, so we can say that the size of $V(C\_CRG_M)$ and $E(C\_CRG_M)$ is at most that of the V and the E in T_{TreeFR}, if the following statements are satisfied:

$$\begin{aligned} |\cup_{scc \in SCC} scc| &\geq |rv_M| \\ |rE_M| &\geq |E(RV_M)|. \end{aligned} \tag{8}$$

During collapsing CRG_M, multiple vertices in an SCC are substituted to a single rv. By the definition of SCC (see Definition 3.4), any SCC should contain more than one vertex. Moreover, rE_M contains edges between vertices in SCC, and the size of rE_M is greater than that of $E(RV_M)$. Hence, both statements in Equation (8) are always true. Therefore, the following statements are always true:

$$|V(C\_CRG_M)| \leq |V| \text{ and } |E(C\_CRG_M)| \leq |E|. \tag{9}$$

The first or the second term in $T_{Proposed}$ can be dominant according to the size of the collapsed vertices and edges. However, the proposed algorithm is faster than the *TreeFR* method (i.e., $T_{Proposed} \leq T_{TreeFR}$) in both cases, due to the statements in Equation (9).

4. COUPLING RELATION GRAPH IN DYNAMIC STRUCTURE MODEL

In classic DEVS models, the proposed method is utilized as a method for efficiently constructing FR, so its contribution may be limited. However, in dynamic structure models, it can guarantee the speedup of simulation executions by efficient FR updates. In this section, we explain how to efficiently reflect dynamic structural changes to FR using CRG and SCC.

4.1. Overview of Faster Simulation in Dynamic Structure Models

Based on hierarchical and modular modeling in DEVS, a dynamic structure model was first considered in Zeigler et al. [1990]. After this work, a lot of studies for representing dynamic structure models in a formal description, such as DSDEVS [Barros 1996], Mobile DEVS [Kim and Kim 2001], DynDEVS [Uhrmacher 2001], LDEF formalism [Bae et al. 2012], and DYS-DEVS [Muzy and Zeigler 2014], were introduced.

In a dynamic structure model, a small structural change may lead to partial or even entire changes in its FRs. However, the previous methods, including *TreeFR* search, cannot estimate how FRs would be changed by structural changes. Thus, they have to reconstruct the overall FRs whenever any structural change occurs (see the top of Figure 1). This is of course inefficient. Having said that, if reusable FRs are selected according to structural changes, the next FRs would be constructed by utilizing the reusable FRs (see the bottom of Figure 1). In the proposed method, CRG and its SCCs would help facilitate this efficient maintenance of FRs.

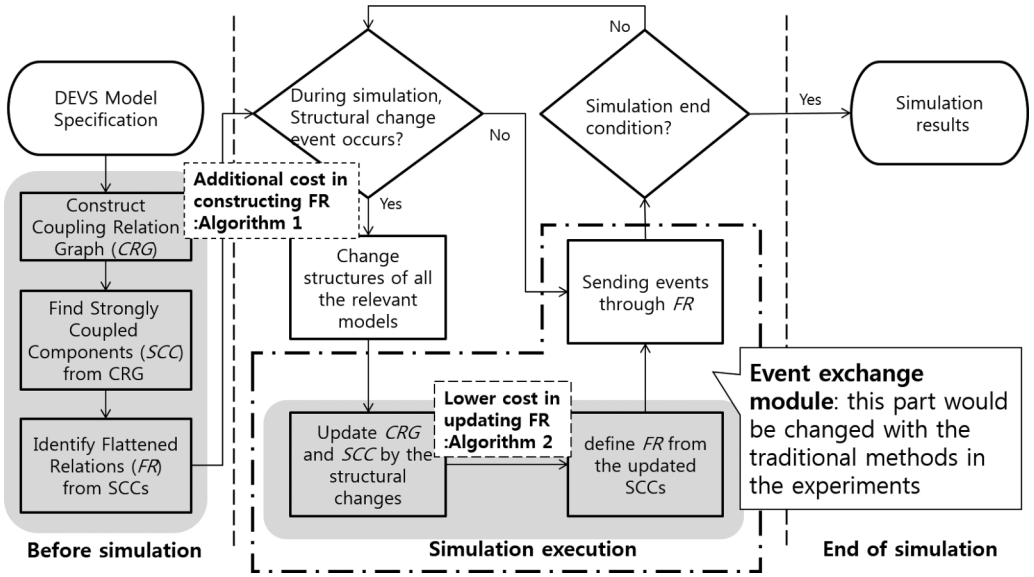

Fig. 7. Simulation execution procedure of a dynamic structure model using the proposed method: compared to ClassicDEVS, additional costs for initially constructing CRG, SCC, and FR is required before the simulation execution (which corresponds to Algorithm 1); on the other hand, compared to TreeFR, low costs for handling dynamic structural changes during the simulation execution (which corresponds to Algorithm 2).

Figure 7 illustrates the overall procedure of simulation execution of dynamic structure models using the CRG and SCCs. Before the simulation execution, the CRG, SCCs, and FRs are constructed from DEVS specifications, which required additional costs compared with the ClassicDEVS approach. When structural changes happen during the simulation execution, the associated model changes their model structures, and these changes make the existing FRs updated. For the FR updates, CRG and its SCCs identify FRs that can be reused in the next FRs. By the reuse of those FRs, FR updates would be handled with a lower cost.

From the additional cost for the initial construction of FR and the partial updates of FRs, the proposed method enables one to exploit the direct event exchanges with the low cost for the FR updates so that the overall simulation execution time would be reduced.

4.2. Managing Dynamic Structural Changes Using Coupling Relation Graph

In order that the CRG and SCCs identify reusable FRs, all dynamic structural changes should be reflected to the CRG. From the formal specifications of dynamic structure models [Barros 1996; Kim and Kim 2001; Uhrmacher 2001; Bae et al. 2012], dynamic structural changes are reduced to the addition or removal of events, coupling relations, and component models. Transforming those changes into the CRG terms, they correspond with the changes of the vertices and the edges. In other words, structural changes in a model are reduced to the additions or the removals of vertices and edge in a CRG.

Moreover, we note that removing a vertex is handled by a series of removing the edges that are incident to the removed vertex, while only adding a new vertex does not cause any changes in the SCCs and FRs. Thus, it is sufficient to discuss how to handle edge removals and additions associated with dynamic structural changes. Since updating the CRG is rather direct and simple, we focused on describing how to update SCCs and FRs with respect to the edge changes.

Algorithm 2 represents how to handle edge (u, v) addition and removal for FR updates. The FR updates in this algorithm are carried out through three phases: detecting the SCCs to be changed, identifying FRs to be updated in the detected SCCs, and updating the SCCs and FRs. A detailed explanation of how these phases are executed in the algorithm follows.

When an edge (u, v) in a CRG_M changes, the algorithm computes the subgraph T_v of CRG_M, which is defined as a graph that consists of reachable vertices from v and edges between the reachable vertices in CRG_M. Then, if the edge change is about an edge removal, the handling process differs with where the edge is positioned: if the edge is positioned between internal vertices, the algorithm finds SCCs that hold the associated vertices. By removing the edge, some vertices in the identified SCCs are disconnected, so they should be removed from the SCCs. Also, FRs that consist of the source vertices linked to the SCCs and the sink vertices in T_v are also removed.

If the edge is positioned between a source vertex (u) and an internal vertex (v), the algorithm removes $SCC(u)$ when u is the only source vertex linked to $SCC(u)$. Then, FRs that consist of u and the sink vertices in T_v are removed. If the edge is positioned between an internal vertex (u) and a sink vertex (v), the algorithm searches SCCs that hold u and deletes FRs that are composed with the source vertices linked to the identified SCCs and v. Otherwise, the edge is positioned between a source vertex and a sink vertex, so (u, v) is added to FR_M.

Similarly, the edge addition is handled differently according to the edge position: when the edge (u, v) is located between internal vertices, the algorithm discovers SCCs that contain u and not v because the edge addition in SCCs that contain both vertices does not cause any changes. Then, all vertices excluding sink vertices in T_v are added to the discovered SCCs, and all pairs of the source vertices linked to the SCCs and the sink vertices in T_v are added to FR_M.

When the edge is started from a source vertex (u) and T_v contains any sink vertices, the algorithm generates $SCC(u)$ as the set of vertices excluding sink vertices in T_v and adds it to SCC_M. Also, pairs of u and the sink vertices in T_v are added to FR_M. When the edge is located between an internal vertex (u) and a sink vertex (v), the algorithm searches SCCs that have u and adds all pairs of the source vertices linked to the SCCs and v to FR_M. Otherwise, the edge (u, v) is added to FR_M.

Contrary to the TreeFR method, Algorithm 2 reduces the searching scope affected by structural changes using the CRG and SCCs, so it can identify which FR to be updated and kept. Such partial FR updates provide more time gains than the TreeFR method as structural changes frequently occur.

4.3. Time Complexity Analysis on Dynamic Updates to FR

In this section, the proposed Algorithm 2 is compared with the *TreeFR* method from the perspective of FR updates. For the FR updates, the *TreeFR* method executes the identical algorithm as the initial construction of the FR, that is, DFS. Hence, the time complexity of the TreeFR method for dynamic updates to FRs is the same as in Equation (1).

On the other hand, FR updates in the proposed algorithm are performed in three phases: (1) *detecting SCCs to be changed by structural changes*, (2) *identifying FRs to be updated in the detected SCCs*, and (3) *updating the identified FRs* (see Algorithm 2). Hence, the time complexity of the proposed algorithm ($T_{Proposed}$) is as follows:

$$T_{Proposed} = T_{detect} + T_{identify} + T_{update}. \tag{10}$$

Structural changes in a model correspond to edge changes in the CRG, and these edge changes incur changes of some SCCs, which can be checked by vertex-level comparisons.

ALGORITHM 2: Handling Dynamic Structural Events

Input: An event e associated with an addition/removal of edge (u, v) in CRG_M.
Compute the subgraph T_v of CRG_M defined as a graph that consists of reachable vertices from v and edges between the reachable vertices
if *u is a source vertex and v is a sink vertex* **then**
    Add/remove (u, v) to/from FR_M according to the edge change
end
if *e corresponds to a removal of an edge (u, v)* **then**
    if *both u and v are internal vertices* **then**
        forall the $SCC \in SCC_M$ *such that* $u, v \in SCC$ **do**
            Remove the vertices in T_v from SCC
            forall the *source vertices s linked to SCC and sink vertices $t \in V(T_v)$* **do**
                Remove (s, t) from FR_M
            end
        end
    end
    else if *u is a source vertex* **then**
        Remove $SCC(u)$ from SCC_M, if u is the only source vertex linked to $SCC(u)$
        forall the *sink vertices $t \in V(T_v)$* **do**
            Remove (u, t) from FR_M
        end
    end
    else if *v is a sink vertex* **then**
        forall the $SCC \in SCC_M$ *such that* $u \in SCC$ **do**
            Remove all pairs (s, v) such that s is a source vertex linked to SCC from FR_M
        end
    end
end
else if *e corresponds to an addition of an edge (u, v)* **then**
    if *both u and v are internal vertices* **then**
        forall the $SCC \in SCC_M$ *such that* $u \in SCC$ *and* $v \notin SCC$ **do**
            Add all vertices excluding sink vertices in T_v to SCC
            forall the *source vertices s linked to SCC and sink vertices $t \in T_v$* **do**
                Add (s, t) to FR_M
            end
        end
    end
    else if *u is a source vertex and T_v contains sink vertices* **then**
        Generate SCC(u) as the set of the vertices excluding sink vertices in T_v
        Add SCC(u) to SCC_M
        forall the *sink vertices $t \in T_v$* **do**
            Add (u, t) to FR_M
        end
    end
    else if *v is a sink vertex* **then**
        forall the $SCC \in SCC_M$ *such that* $u \in SCC$ **do**
            forall the *source vertices s linked to SCC* **do**
                Add (s, v) to FR_M
            end
        end
    end
end

Hence, the time costs for those investigations (T_{detect}) are expressed as follows:

$$T_{detect} = O\left(\sum_{scc \in SCC_M} |scc|\right). \tag{11}$$

After changing SCCs are detected, the proposed algorithm investigates whether FRs from the detected SCCs need to be updated or not. This test also is performed by applying DFS to the source vertex in the detected SCCs. Hence, the time complexity to identify FRs to be updated ($T_{identify}$) is represented as follows:

$$T_{identify} = O\left(\sum_{scc \in detSCC_M} (|scc| + |E_{scc}|)\right), \tag{12}$$

where $detSCC_M = \{scc \mid scc \in SCC_M$ contains vertices of changing edges$\}$

$$E_{SCC} = \{(u, v) \mid u, v \in SCC, \text{and } (u, v) \in E(CRG_M)\}.$$

When changing FRs are identified, the FRs are added to or removed from FR_M in time proportional to the number of the FRs, which is bounded by $T_{identify}$. Therefore, the total time complexity for the dynamic updates is expressed by summing up the previous terms:

$$\begin{aligned} T_{Proposed} &= T_{detect} + T_{identify} + T_{update} \\ &= O\left(\sum_{scc \in SCC_M} |scc|\right) + O\left(\sum_{scc \in detSCC_M} (|scc| + |E_{scc}|)\right). \end{aligned} \tag{13}$$

By the definition of SCC_M, the size of SCC_M is smaller than or equal to the size of V_{ao}, and the size of $detSCC_M$ is smaller than or equal to that of SCC_M because $detSCC_M$ is a subset of SCC_M. Also, the size of the summation of all SCCs and of all E_SCC in $detSCC_M$ is smaller than the size of $V(CRG_M)$ and $E(CRG_M)$ (see Definition 3.4):

$$\begin{aligned} O\left(\sum_{scc \in detSCC_M} |scc|\right) \le O\left(\sum_{scc \in SCC_M} |scc|\right) \le |V_{ao}| \\ O\left(\sum_{scc \in detSCC_M} |scc|\right) \le |V(CRG_M)| \\ O\left(\sum_{scc \in detSCC_M} |E_{scc}|\right) \le |E(CRG_M)|. \end{aligned} \tag{14}$$

Hence, the proposed algorithm is faster than the *TreeFR* method in dynamic updates to FRs (i.e., $T_{Proposed} \le T_{TreeFR}$). In particular, the expected speedup in the dynamic updates is more than that in the initial construction because the size of the searching scopes is significantly reduced in the dynamic updates.

5. CASE STUDY: EVALUATION OF THE PROPOSE METHOD

To evaluate the efficiency of the proposed method, we designed and performed two virtual experiments, the multiqueuing model (MQM) and evacuation-agent-based model (EABM). However, due to the page limit, this article introduces the EABM case only; simulation results and analyses about MQM are offered by the online supplementary.

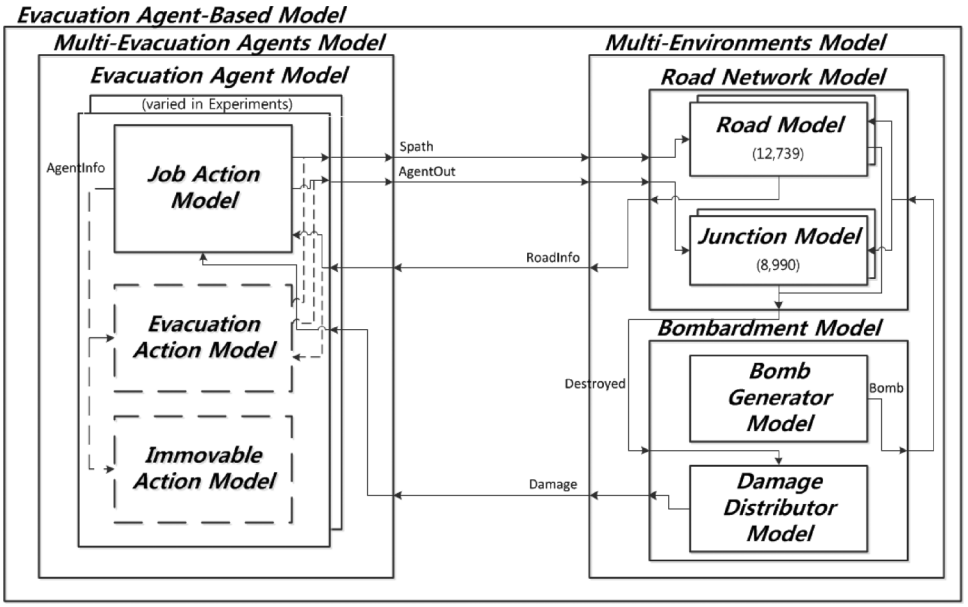

Fig. 8. Hierarchical structure and coupling relations of the evacuation-agent-based model (the number in the parentheses below a model name indicates the number of the model instances and dashed lines indicate dynamic structural changes in EABM).

5.1. Example Model: Evacuation-Agent-Based Model

The EABM describes individual-level evacuations from the Gangnam region in Seoul, Korea, due to bombardments by enemies [Bae et al. 2014]. The EABM consists of multiple-agents models and environment models in a hierarchical form. The multi-evacuation agents model (see Figure 8) consists of multiple evacuation agent models, and each evacuation agent model contains three action models: job action model (for normal agents), evacuation action model (for evacuating agents), and immovable action model (for damaged agents). The multi-environments model (see Figure 8) holds one road network model and one bombardment model. The road network model contains 12,730 road and 8,990 junction models based on real GIS data for the Gangnam region. The bombardment model contains a bomb generator model to generate bombs and their target points, and a damage distributor model to spread bomb damages to the road network and the evacuation agents. Figure 8 shows the hierarchical structure and the coupling relations in the EABM.

An evacuation scenario applied to the EABM is as follows: when the bombs are generated from the bomb generator model, they head to the predefined spots in the Gangnam region. After the bombardments begin, all agents change their actions to be disaster response actions using dynamic structural changes. Specifically, job action models in the agents can be changed to either an evacuation action (when the agent is far enough from the bombing spots) or an immovable action (when the agent is damaged from the bombing spots). Agents with the evacuation action escape the Gangnam region through the damaged road network, and agents with the immovable action stay and wait for a rescue.

5.2. Experimental Design

The EABM was developed by LDEF formalism [Bae et al. 2014], which is based on DEVS formalism but supports describing dynamic structure discrete event models

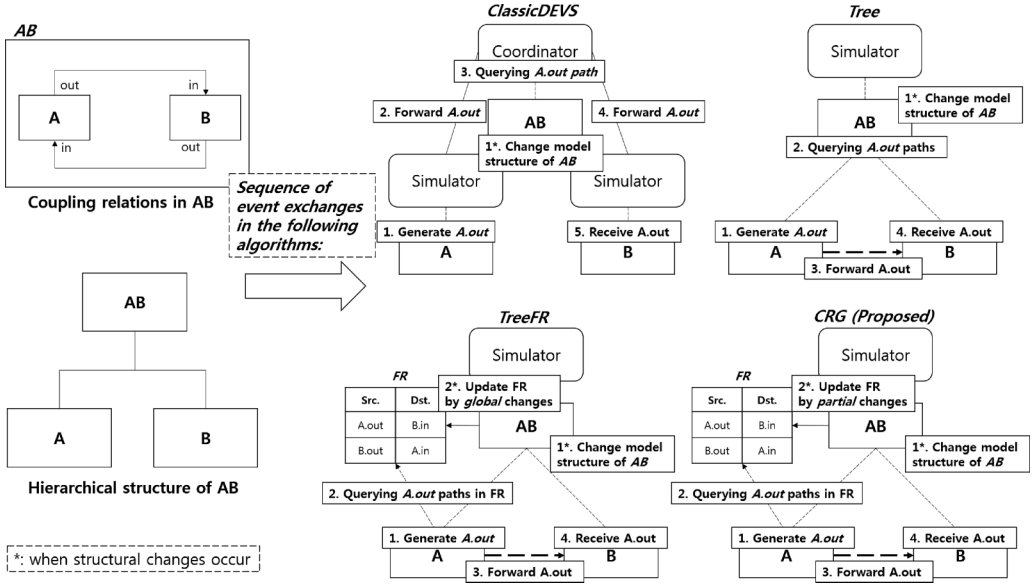

Fig. 9. Event exchange algorithms from ClassicDEVS, Tree, TreeFR, and CRG applied to the experiments.

with embedding agent-based model contexts. To simulate the EABM, we developed a simulation environment based on the DEVS abstract simulation algorithm [Zeigler et al. 2000]. Moreover, to handle dynamic structural changes during simulation execution, the procedure for dynamic structural changes is added to the simulation environment: (1) when a model generates an event, the parent component identifies whether the event triggers structural changes from its specification; (2) if the event is about the structural changes, the event changes coupling structures of the relevant models; (3) if not, the event is forwarded through the existing coupling structures.

Furthermore, the simulation environment is designed to separate its *event exchange module* (see Figure 7), which handles event exchanges during simulation execution, so we can simulate the example models with changing event exchange algorithms only. Through this separation, the traditional event exchange algorithms can be applied to the experiments, even if they are not concerned about dynamic structural changes. Among them, we developed four event exchange algorithms including the proposed one: *ClassicDEVS*, *Tree*, *TreeFR*, and *CRG* (as the proposed method).

Figure 9 illustrates the differences of the four event exchange algorithms: ClassicDEVS indicates an event exchange procedure that is proposed in the DEVS abstract simulation algorithm: models send and receive events through the hierarchically associated abstract simulators (i.e., simulator and coordinator). In the Tree method, models exchange events through the hierarchical structure of their parent model, not through multiple abstract simulators. These algorithms utilize the current coupling structures in the event exchanges. Hence, even if dynamic structural changes occur, these event exchange methods are applicable.

Contrary to those algorithms, TreeFR and CRG build the FR before simulation execution and utilize the FR in event exchanges. TreeFR builds the FR by traversing hierarchically structured coupling relations with path-finding algorithms [Kim et al. 2000]. When dynamic structural changes occur, the TreeFR method has no choice but to reconstruct new FRs using the initial traversing method. On the other hand, the CRG

Table II. Experimental Designs for Comparing the Efficiency of the Proposed Algorithm Using Evacuation-Agent-Based Model (EABM): Four Event Exchange Algorithms, Including the Proposed One, and Parameters About the Model Structure Are Considered as Experimental Variables

Model	Variable Name	Experimental Cases	Implications
EABM	Algorithms for event exchange	*ClassicDEVS*, *Tree*, *TreeFR*, and *CRG* (4 cases)	Event exchange algorithms to be changed in the simulation environment
	Number of evacuation agents	1,000, 2,000, 3,000, 4,000, and 8,000 (5 cases)	Number of evacuation agents in evacuation-agent-based model
	Number of structural changes	1, 30, 60, and 100 (4 cases)	Number of structural changes in EABM calibrated by maximum evacuation response time of evacuation agents
	Total cases	$4 \times 5 \times 4 = 80$ cases	10 replications on each cases ($80 \times 10 = 800$ simulations)

method constructs a CRG and utilizes the CRG to build the FR. Moreover, it supports partially updating FRs using the CRG and its SCCs.

To compare the performance of those algorithms, these four event exchange algorithms are concerned as experimental variables. Also, the parameters for the model structures are added to the experimental variables, such as the number of evacuation agents and the frequency of structural changes. The details about the experimental design are presented in Table II. In addition, the experimental variables from the EABM are derived from its model parameters, but their value ranges are arbitrarily determined.

6. PERFORMANCE ANALYSIS

This section shows the performance of the proposed method compared to other event exchange algorithms in simulating the EABM. To see the general trends of the four event exchange algorithms, we drew response surfaces of the overall elapsed time of simulation executions with respect to varying the experimental variables (see Figure 10).

Among the response surfaces in Figure 10, we regarded the ClassicDEVS case as the baseline of this graphical performance comparison. The baseline shows a tendency in which the execution time is more rapidly increased by the number of agents than by the number of structural changes. It infers that the time overheads from the event exchange method in ClassicDEVS (i.e., indirect event exchanges) take a large portion of the overall simulation execution time. Also, the baseline illustrates that as the structural changes increase, the growth rate of the execution time is reduced a little. We note that this reduction is caused by the change of the EABM dynamics from the increase of the structural changes, not by the applied algorithms. As proof of this statement, the response surface of the Tree algorithm, which also applies the indirect event exchanges, shows a similar growth rate according to the structural changes as well.

Compare to the baseline, the response surface of Tree shows better performance in all the cases due to its reduced indirect event exchanges (please refer to Figure 9). Meanwhile, we can see that the response surfaces of TreeFR and CRG show better performance than CRG and Tree, which represents the efficiency of the direct event exchanges in reducing simulation execution time. Also, in the comparison between the response surfaces of Tree and CRG, we can see that there is a different tendency: their performance gap is getting larger when the structural changes increased. This difference is induced from the efficient FR updates of CRG rather than TreeFR; while TreeFR rebuilds FRs when a structural change occurs, CRG reuses a portion of the existing FRs and adds new constructed FRs from the structural changes.

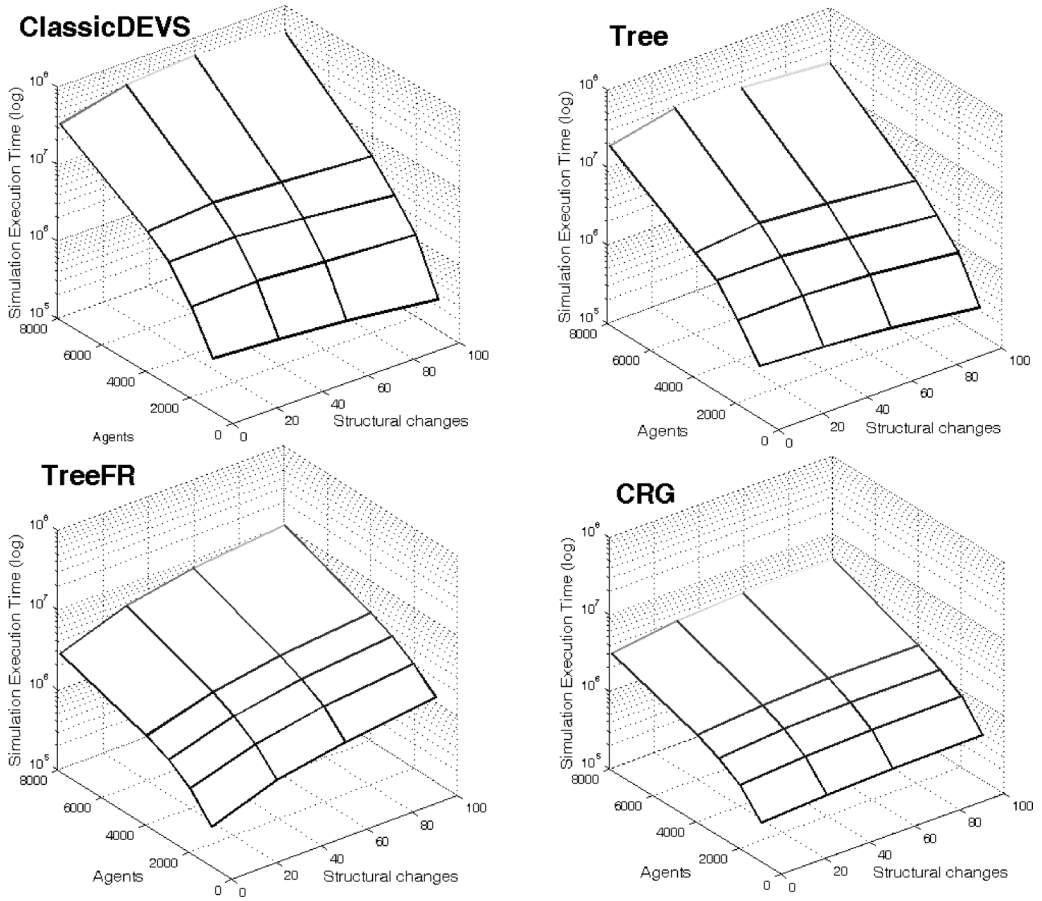

Fig. 10. Response surfaces for the elapsed time of the simulation executions with four event exchange algorithms (ClassicDEVS (as the baseline), Tree, TreeFR, and CRG) by varying the number of evacuation agents and the frequency of structural changes in the EABM.

In the supplementary document, we presented a detailed investigation of simulation execution time—the simulation execution time is separated into three phases: initial construction of FR (*InitialFR*), update of FR (*updateFR*), and event exchanges (*EventExchanges*)—and we compared the execution time of each phase in the four event exchange algorithms. As a result, we confirmed that the comparison results from the three phases are identical to the results from the response surfaces.

7. DISCUSSION

Through the demonstrations in this article, we illustrated that the proposed method is more efficient than other traditional methods. In particular, it presents outstanding performance on the FR updates over the previous FR-utilizing method, that is, TreeFR. However, at the same time, the experimental results also revealed several limitations.

First, in the MQM example (please refer to the supplementary document), the proposed method still shows better performance on the overall execution time, yet its InitialFR takes more time than TreeFR's. It indicates that the FR construction from the proposed method is dependent on the model structure; while the proposed method affords time gains as in Algorithm 1, it also requires extra time for constructing extra

data structure, such as the CRG and SCC. Therefore, when a model has too simple a structure that it cannot provide enough time gains against the extra costs, the proposed method may be inefficient.

The second limitation is that the proposed algorithm cannot guarantee efficiency when a simulation model holds a small number of components but a large number of structural changes. This limitation is exposed in the EABM example (e.g., in the case of 1,000 agents and 100 structural changes, the simulation execution time of CRG takes 382,754.5msec, but that of ClassicDEVS takes 247,533msec); even if the proposed method always secures time gains from the FR utilizations, this time gain does not always outplay the costs from the traditional methods that do not utilize FRs in event exchanges, such as ClassicDEVS and Tree.

Despite these limitations, the two examples obviously depict that the proposed method is more efficient in the FR updates (than TreeFR) and the event exchanges (than ClassicDEVS and Tree). Moreover, the examples represent that those time gains are proportional to events, structural changes, and model hierarchy level and width. Therefore, based on the example results and limitations, we argue that the proposed method is efficient when a simulation model has a complex model structure with intricate interactions and frequent structural changes, such as models about complex systems.

8. CONCLUSION

The hierarchical model structure provides considerable advantages in the model development, but it also causes many time costs by inducing the indirect event exchanges. Several traditional approaches resolved this time loss by flattening the hierarchical model structure using model specifications. However, considering that model structure could change during simulation execution, the traditional approaches have no choice but to flatten the changed model structure from scratch. Such simple reconstruction might miss the opportunity of an efficient update because some elements in the flattened structure could be reused. Therefore, we present a better-optimized data structure and algorithms to flatten the model hierarchy from the execution time perspective.

This article proposes algorithms that efficiently construct and manage the flattened structure, or flattened relations (FRs). To this end, this article suggests Coupling Relation Graph (CRG) and Strongly Coupled Component (SCC) concepts to reduce the time costs from constructing and updating FRs. To prove the efficiency of the proposed algorithm, theoretical and empirical analyses are performed: theoretically, it was compared with another FR-utilized algorithm by the time complexity analyses; empirically, we applied traditional event exchange algorithms and the proposed algorithm to simulate the multiqueuing model and the evacuation-agent-based model. These analysis results proved that the proposed algorithm is not always the best performer, but when a model contains lots of components and frequent structural changes, it guarantees better performance on the reduction of the simulation execution time than the traditional event exchange methods. Hence, we expect that this flattening method would support faster simulation executions of a complex hierarchical and dynamic structure model.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

Adedoyin Adegoke, Hamidou Togo, and Mamadou K. Traoré. 2013. A unifying framework for specifying DEVS parallel and distributed simulation architectures. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 90, 7 (2013), 1293–1309.

Amin Alvanchi, SangHyun Lee, and Simaan AbouRizk. 2011. Modeling framework and architecture of hybrid system dynamics and discrete event simulation for construction. *Computer-Aided Civil and Infrastructure Engineering* 26, 2 (2011), 77–91.

Jang Won Bae, Jeong Hoon Kim, Il-Chul Moon, and Tag-Gon Kim. 2016. Accelerated simulation of hierarchical military operations with tabulation technique. *Journal of Simulation* 10, 1, 36–49.

Jang Won Bae and Tag Gon Kim. 2010. DEVS based plug-in framework for interoperability of simulators. In *Proceedings of the 2010 Spring Simulation Multiconference (SpringSim'10)*. Society for Computer Simulation International, San Diego, CA, 127:1–127:7. http://dx.doi.org/10.1145/1878537.1878670.

Jang Won Bae, GeunHo Lee, and Il-Chull Moon. 2012. Formal specification supporting incremental and flexible agent-based modeling. In *Proceedings of the 2012 Winter Simulation Conference (WSC'12)*. 1–12.

Jang Won Bae, SeHoon Lee, Jeong Hee Hong, and Il-Chul Moon. 2014. Simulation-based analyses on massive evacuation from metropolis during bombardment. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 90, 11 (2014), 1244–1267.

Yaneer Bar-Yam. 1997. *Dynamics of Complex Systems*. Addison-Wesley, Reading, MA. 213 pages.

Fernando J. Barros. 1996. The dynamic structure discrete event system specification formalism. *Transactions of the Society for Computer Simulation International* 13, 1 (1996), 35–46.

Federico Bergero and Ernesto Kofman. 2014. A vectorial DEVS extension for large scale system modeling and parallel simulation. *Simulation* 90, 5 (2014), 522–546.

Federico Bergero, Ernesto Kofman, and François Cellier. 2013. A novel parallelization technique for DEVS simulation of continuous and hybrid systems. *SIMULATION* (Aug. 2013), 663–683. Issue 6. DOI:http://dx.doi.org/10.1177/0037549712454931

Jong Hyuk Byun, Chang Beom Choi, and Tag Gon Kim. 2009. Verification of the DEVS model implementation using aspect embedded DEVS. In *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, San Diego, CA, 151.

Bin Chen and Hans Vangheluwe. 2010. Symbolic flattening of DEVS models. In *Proceedings of the 2010 Summer Computer Simulation Conference*. Society for Computer Simulation International, 209–218.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. Transitive closure of a directed graph. In *Introduction to Algorithms*. MIT Press and McGraw-Hill, 632–634.

Richard M. Fujimoto. 1999. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, New York, NY.

Judith L. Gersting. 2007. *Mathematical Structures for Computer Science*. W. H. Freeman. 807 pages.

Ezequiel Glinsky and Gabriel A. Wainer. 2005. DEVStone: A benchmarking technique for studying performance of DEVS modeling and simulation environments. In *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05), October 10-12, 2005, Montreal, Canada*. 265–272. DOI:http://dx.doi.org/10.1109/DISTRA.2005.18

Kumar K. Goswami and Ravishankar 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)*. Society for Computer Simulation International, San Diego, CA, 197–202.

Gang Guo, Bin Chen, Xiao Gang Qiu, and Zhen Li. 2012. Parallel simulation of large-scale artificial society on CPU/GPU mixed architecture. In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 174–177.

Alexander Helleboogh, Giuseppe Vizzari, Adelinde M. Uhrmacher, and Fabien Michel. 2007. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems* 14, 1 (2007), 87–116. DOI:http://dx.doi.org/10.1007/s10458-006-0014-y

Jan Himmelspach, Roland Ewald, Stefan Leye, and Adelinde M. Uhrmacher. 2007. Parallel and distributed simulation of parallel DEVS models. In *Proceedings of the 2007 Spring Simulation Multiconference,* Vol. 2. Society for Computer Simulation International, 249–256.

Thomas Homer-Dixon, Jonathan Leader Maynard, Matto Mildenberger, Manjana Milkoreit, Steven J. Mock, Stephen Quilley, Tobias Schröder, and Paul Thagard. 2013. A complex systems approach to the study of ideology: Cognitive-affective structures and the dynamics of belief systems. *Journal of Social and Political Psychology* 1, 1 (2013), 337–363.

Xiaolin Hu and Bernard P. Zeigler. 2004. A high performance simulation engine for large-scale cellular DEVS models. In *Proceedings of the High Performance Computing Symposium (HPC'04)*. 3–8.

Xiaolin Hu, Bernard P. Zeigler, and Saurabh Mittal. 2005. Variable structure in DEVS component-based modeling and simulation. *SIMULATION: Transactions of the Society for Modeling and Simulation International2* 81, 2 (2005), 91–102.

Moon Ho Hwang. 2005. Tutorial: Verification of real-time system based on schedule-preserved DEVS. In *Proceedings of 2005 DEVS Symposium*. 2–8.

Shafagh Jafer and Gabriel Wainer. 2009. Flattened conservative parallel simulator for DEVS and cell-DEVS. In *Proceedings of the International Conference on Computational Science and Engineering, 2009 (CSE'09)*. Vol. 1. IEEE, 443–448.

Jae-Hyun Kim and Tag Gon Kim. 2001. DEVS-based framework for modeling/simulation of mobile agent systems. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 76, 6 (2001), 345–357.

Kihyung Kim, Wonseok Kang, Bong Sagong, and Hyungon Seo. 2000. Efficient distributed simulation of hierarchical devs models: Transforming model structure into a non-hierarchical one. In *Proceedings of the Simulation Symposium 2000 (SS'00)*. IEEE Computer Society, Washington, DC, 227–233.

Se Jung Kwon and Tag Gon 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 - DEVS Integrative M&S Symposium (TMS/DEVS'12)*. Society for Computer Simulation International, San Diego, CA, 14:1–14:8.

Wan Bok Lee and Tag Gon Kim. 2003. Simulation speedup for DEVS models by composition-based compilation. In *Proceedings of the Summer Computer Simulation Conference*.

Seong Yong Lim and Tag Gon Kim. 2001. Hybrid modeling and simulation methodology based on DEVS formalism. In *Summer Computer Simulation*.

Qi Liu and Gabriel Wainer. 2010. Accelerating large-scale DEVS-based simulation on the cell processor. In *Proceedings of the 2010 Spring Simulation Multiconference on (SpringSim'10)*. ACM Press, New York, NY, 124:1–124:8. DOI:http://dx.doi.org/10.1145/1878537.1878667

Martina Maggio, Kristian Stavåker, Filippo Donida, Francesco Casella, and Peter Fritzson. 2009. Parallel simulation of equation-based object-oriented models with quantized state systems on a GPU. In *Proceedings of the 7th International Modelica Conference*.

Alexander Muzy and James J. Nutaro. 2005. Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators. In *Proceedings of the 1st Open International Conference on Modeling & Simulation (OICMS'05)*. 1–8.

Alexandre Muzy and Bernard P. Zeigler. 2014. Specification of dynamic structure discrete event systems using single point encapsulated control functions. *International Journal of Modeling, Simulation, and Scientific Computing* 5, 3 (2014), 1450012. DOI:http://dx.doi.org/10.1142/S1793962314500123.

Ernesto Posse, Jean-Sébastien Bolduc, and Hans Vangheluwe. 2003. Generation of DEVS modelling and simulation environments. In *Proceedings of the 2003 Summer Computer Simulation Conference*. 139–146.

H. Pranevicius, L. Simaitis, M. Pranevicius, and O. Pranevicius. 2011. Piece-linear aggregates for formai specification and simulation of hybrid systems: Pharmacokinetics patient-controlled analgesia. *Elektronika ir Elektrotechnika* 110, 4 (2011), 81–84.

Gauthier Quesnel, Raphaël Duboz, and Éric Ramat. 2009. The virtual laboratory environment–An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory* 17, 4 (2009), 641–653.

Hesham Saadawi and Gabriel Wainer. 2009. Verification of real-time DEVS models. In *Proceedings of the 2009 Spring Simulation Multiconference (SpringSim'09)*. Society for Computer Simulation International, San Diego, CA, Article 143, 8 pages.

Jeyaveerasingam G. Shanthikumar and Robert G. Sargent. 1983. A unifying view of hybrid simulation/analytic models and modeling. *Operations Research* 31, 6 (1983), 1030–1052. http://www.jstor.org/stable/170837.

Fahad A. Shiginah and Bernard P. Zeigler. 2011. A new cell space DEVS specification: Reviewing the parallel DEVS formalism seeking fast cell space simulations. *Simulation Modelling Practice and Theory* 19, 5 (2011), 1267–1279.

Eugene Syriani, Hans Vangheluwe, and Amr Al Mallah. 2011. Modelling and simulation-based design of a distributed devs simulator. In *Proceedings of the Winter Simulation Conference (WSC'11)*. 3007–3021.

Mamadou K. Traoré. 2009. A graphical notation for DEVS. In *Proceedings of the 2009 Spring Simulation Multiconference (SpringSim'09)*. Society for Computer Simulation International, San Diego, CA, Article 162, 7 pages.

Adelinde M. Uhrmacher. 2001. Dynamic structures in modeling and simulation: A reflective approach. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 11, 2 (2001), 206–232.

Hans L. M. Vangheluwe. 2000. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *IEEE International Symposium on Computer-Aided Control System Design, 2000 (CACSD'00)*.. IEEE, 129–134.

Gabriel Wainer. 2000. Improved cellular models with parallel cell-DEVS. *Transactions of the SCS* 17, 2 (2000), 73–89.

Gabriel Wainer and N. Giambiasi. 2001. Application of the cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation* 76, 1 (2001), 22–39.

Gabriel A. Wainer. 2004. Modeling and simulation of complex systems with cell-DEVS. In *Proceedings of the 36th Conference on Winter Simulation*. 49–60.

Gregory Zacharewicz, Maâmar El-Amine Hamri, Claudia Frydman, and Norbert Giambiasi. 2010. A generalized discrete event system (g-DEVS) flattened simulation structure: Application to high-level architecture (HLA) compliant simulation of workflow. *Simulation* 86, 3 (2010), 181–197.

Bernard P. Zeigler, Tag Gon Kim, and Chilgee Lee. 1990. Variable structure modelling methodology: An adaptive computer architecture example. *Transactions of the Society for Computer Simulation* 7, 4 (1990), 291–320.

Bernard P. Zeigler and Tuncer I. Ören. 1986. Multifaceted, multiparadigm modeling perspectives: Tools for the 90's. In *Proceedings of the 18th Conference on Winter Simulation*. 708–712.

Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. 2000. *Theory of Modeling and Simulation* (2nd ed.). Vol. 132. Academic Press. DOI:http://dx.doi.org/10.1159/000074301

Bernard P. Zeigler and Hessam S. Sarjoughian. 2013. Dynamic structure: Agent modeling and publish/subscribe. In *Guide to Modeling and Simulation of Systems of Systems*. Springer, 125–143.

Bernard P. Zeigler and Guoqing Zhang. 1990. Mapping hierarchical discrete event models to multiprocessor systems: Concepts, algorithm, and simulation. *Journal of Parallel and Distributed Computing* 9, 3 (1990), 271–281.

Received November 2014; revised November 2015; accepted November 2015