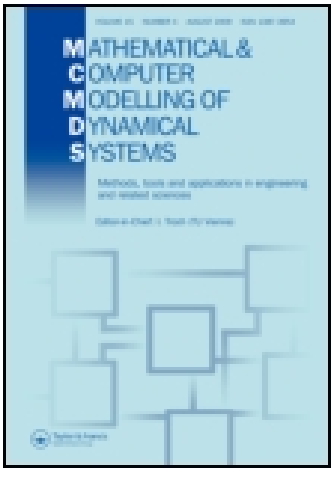

Mathematical and Computer Modelling of Dynamical Systems: Methods, Tools and Applications in Engineering and Related Sciences
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/nmcm20

Partitioned event graph: formalizing LP-based modelling of parallel discrete-event simulation

Bing Wang[a], Bo Deng[a], Fei Xing[b], Dongxia Wang[a] & Yiping Yao[b]
[a] Beijing Institute of Systems Engineering, Beijing, 100101, P.R. China
[b] National University of Defense Technology, Changsha, 410073, P.R. China
Published online: 06 May 2014.

To cite this article: Bing Wang, Bo Deng, Fei Xing, Dongxia Wang & Yiping Yao (2015) Partitioned event graph: formalizing LP-based modelling of parallel discrete-event simulation, Mathematical and Computer Modelling of Dynamical Systems: Methods, Tools and Applications in Engineering and Related Sciences, 21:2, 153-179, DOI: 10.1080/13873954.2014.911750

To link to this article: http://dx.doi.org/10.1080/13873954.2014.911750

Mathematical and Computer Modelling of Dynamical Systems, 2015
Vol. 21, No. 2, 153–179, http://dx.doi.org/10.1080/13873954.2014.911750

Partitioned event graph: formalizing LP-based modelling of parallel discrete-event simulation

Bing Wang[a]*, Bo Deng[a], Fei Xing[b], Dongxia Wang[a] and Yiping Yao[b]

[a]*Beijing Institute of Systems Engineering, Beijing, 100101, P.R. China;* [b]*National University of Defense Technology, Changsha, 410073, P.R. China*

(*Received 17 June 2012; accepted 29 March 2014*)

Logical process (LP) is a modelling paradigm widely used in parallel discrete-event simulation (PDES). However, effective methods for formalizing LP-based modelling of PDES are lacking. This prevents an unambiguous, platform-independent description of LP-based models. We present a formalism named partitioned event graph (PEG) as a solution. PEG extends classical event graph formalism towards a formal specification for LP-based PDES models. We map between PEG and LP-based models, define the structural operational semantics (SOS) in a timed-labelled transition system, and discuss the Wallclock time-based execution. We propose a PEG-based model transformation framework for PDES, which has three model representation phases and distinguishes amongst four kinds of personnel roles. Finally, we present a domain-specific language (DSL) for the PDES of a Lotka–Volterra system and obtain preliminary parallel simulation results using YinHe Simulation Utilities for Parallel Environment (YHSUPE). The case study shows that the PEG-based framework not only effectively transforms a DSL into the LP paradigm, but will also result in efficient parallel simulation on a specific platform. In summary, by setting out specific characteristics for event scheduling and state space partition in the LP paradigm, PEG provides a formal method for model behaviour analysis and cross-platform model transformation.

Keywords: parallel discrete-event simulation; event graph; logical process; structural operational semantics; model transformation

AMS Subject Classification: 00A72; 00A71; 93C65; 65Y05; 68Q55

1. Introduction

Parallel discrete-event simulation (PDES) has been recognized as a challenging field which bridges modelling and simulation, and high-performance computing [1]. PDES's superior performance compared to sequential discrete event system (DES) derives from its explicit decomposition of a model into sub-models which spreads the computing load over distributed processors and speculatively pushes the simulation time forward. Logical process (LP) is the most commonly used modelling paradigm in PDES. LP was introduced to PDES to model the physical processes a system consists of. Because of the intuitive depiction of time and space, as well as the commonality in implementation on parallel machines, LP paradigm has been the dominant PDES modelling method for a long time.

Recent research into the parallel simulation of LP-based models has usually attempted to overcome the performance bottleneck [2] of a sequential DES [3]. There has been

*Corresponding author. Email: wangbing.nudt@gmail.com

useful work on algorithm optimization, some novel hardware platform customization, and several effective modelling methodologies, but few effective methods for formalizing LP-based modelling of PDES.

There are many formalisms for DESs, such as DEVS [4], Timed Automata [5], Petri Net, Process Algebra GSMP [6], StateChart [7], and Activity Cycle Diagrams [8]. These formalisms differ in their usability due to their underlying semantic models; as to the domains they are suitable to describe; as to what property, aspect, or granularity they can be used to analyse; and, in their expressive power [9]. These formalisms cannot be directly used on LP-based models in the PDES as they either underspecify or overspecify the LP paradigm. There are two key characteristics for LP paradigm sharing by most PDES platforms: event scheduling and state space partition. A novel formalism precisely specifying these characteristics is required.

A recent paper presented a formalism named 'extended event graph' (EEG) [10] for PDESs. However, it incompletely reviewed the literature, and its extension of event graph (EG) formalism [11–13] requires a modification to correctly specify the LP paradigm. The DVE-like [14,15] modelling language in EEG does not fully describe the key characteristics of the LP paradigm. The 'parallel event process' [15] [16, p. 27] specified in its denotational semantics is a characteristic rarely shared by PDES time management protocols. Additionally, the formal definitions of EEG and LP in ref. [10] fail to distinguish the type and instance of specific elements in the formalism. This lack of distinction weakens its claim made for the rigorousness of the model transformation.

In this paper, we develop a formalism for PDES based on the EG formalism, called partitioned event graph (PEG), to provide an unambiguous and platform-independent description of LP-based models. PEG not only offers a formal representation for the LP-based model, but also provides a structural operational semantics (SOS) on timed-labelled transition system (TLTS) [17, p. 105]. Therefore, PEG can be used for cross-platform model transformation, automatic compilation to programming languages, and formal analysis of PDES-model behaviour. PEG accomplishes this by formally specifying event scheduling and the state space partition of the LP paradigm. These two characteristics correspond to synchronization and partitioning efforts which are keys to the correctness and performance of PDES.

The next section reviews the literature on formalizing LP-based modelling of PDES. Section 3 presents the PEG in detail. We construct a map between the PEG and the LP-based model, and define the SOS in a TLTS. Also, the Wallclock time execution and the related equivalence relationship are defined. Section 4 presents a PEG-based domain-specific modelling framework for PDES. Section 5 presents a domain-specific language (DSL) for the spatial stochastic simulation of chemical reactions based on PEG to demonstrate PEG-based framework's capability. Section 6 closes the paper by discussing the major findings and providing some direction for future work.

2. Related works

LP is a widely used term in PDES. Fujimoto et al. describe LP as a synonym for any sequential simulation or simulator [2, p. 40], while Curry et al. and Peschlow et al. introduce it to the sequential simulation branching [18,19]. The time-parallel simulation decomposes the model temporarily, and every computation unit is deemed an LP [2, p. 177]. In this paper, LPs are regarded as model entities and core execution units during the simulation [3,20]. They interact with each other by exchanging *events*.

The development of the PEG is based on the EG formalism. The EG, also known as the simulation graph [21] and the event relationship graph [22], explicitly expresses all the relationships between events, and is the most direct formalism for the event scheduling world view. EG has the same expressiveness as the Turing machine [11]. Infinitesimal perturbation analysis and mathematical programming are used to analyse the properties of EG models [23]. OMIGA and SIMKIT [13] are typical toolkits for EG-based modelling, but only with sequential execution. Viskit [24] provides a subscribe/publish mechanism based on bridges, which consist of communication ports between sub-graph models. J. de Lara et al. present the distributed event graph (DEG) [25] which extends classical event graphs for component-based models in the sequential DES, and the operational semantics is given by means of graph transformation.

The most common PDES modelling method is to directly use the application program interfaces (APIs) provided by PDES platforms, such as TWOS [26], GTW [27], SPEEDES [28], POSE [29], ROSS [30], and μsik [31]. API-based PDES modelling methods have low usability due to their tight platform coupling.

DSLs hide platform details, making modelling more friendly to domain experts. For example, the Maisie [32] language is based on GTW and frees users from having to know API details. However, most domain-specific PDES modelling methods remain tightly coupled to specific platforms and the proprietary execution semantics hampers cross-platform reusability and formal analysis.

Component-based modelling effectively reduces model construction difficulties and increases reusability. Sargent used visualization and a component-based method in the control flow graph [33] to construct LP-based models. Liu et al. [34] built a component-based modelling environment based on a variant of DEVS on top of the YinHe Simulation Utility of Parallel Environment (YHSUPE) [35]. Other EG-based modelling works include Jane [36] and Pave [37]. These efforts usually require direct transforming from component-based formalisms into underlying platforms because LP-based models are not natively component based. Mapping high-level formalisms into LP paradigms is another method of introducing formal method to PDES. Researchers from the University of Carleton have mapped cellular automata to LP paradigms [38] and implemented a Cell-DEVS based PDES platform [39]. Hybinnette mapped a multi-agent system into the LP-based model in Project SASSY [40]. Chen developed a middleware named HLA-Grid-RePast to execute agent-based model on the Grid [41]. Oguara et al., in Project MAS-PDES [42], organized LP-based models into reconfigurable tree structures based on influence conical and represented the transformation of information sharing among agents in the LP paradigm. However, each of these mappings is still directly linked to specific platforms. The lack of platform-independent representation makes it hard to do cross-platform migration and to verify transformation correctness.

Nutaro represents the simulation of LPs as a stack-based computing process in order to facilitate formal analysis [43], but its idiomatic semantics makes it hard to incorporate into commonly used formal methods. Tapus et al. discuss the formal model and the operational semantics of speculative execution in distributed environments [44], but without taking into consideration the equivalence to a sequential execution, which is of key importance in PDES.

In summary, formalisms for general discrete-event modelling do not yet fully depict certain key characteristics of the LP paradigm, including state space partition, event-driven semantics, and the partial order among events. Current LP-based modelling methods for PDES rely either on high-level programming language or artificial structure

to define the execution semantics. The lack of a formal semantics hampers further study on the transformation and the verification of LP-based models.

3. Partitioned event graph

The PEG is presented in this section. First, the assumptions in developing the formalism are listed; then the PEG is defined formally; then the map between the LP-based Model and the PEG-based model is constructed; then, the SOS on TLTS is presented; and finally, the Wallclock time-based execution is discussed.

3.1. Assumptions

Regulation of the system under investigation plays a central role in formal method. For the PEG formalism, the following assumptions are made:

(1) There are no simultaneous events. In simulation, we are interested in the state trace generated by the system behaviour and not high-level interleaving concurrency, and decoupling can be used to eliminate simultaneous events [45].
(2) The condition expression on edge should contain finite enumerable relations joined by Boolean operators, and thus the length of the edge condition is both enumerable and finite [21].
(3) The number of LP classes, of LPs (partitions), and of event types are each finite, and the number of event instances is enumerable, but not necessarily finite.
(4) The state space of the simulation model is enumerable, but not necessarily finite, the same as in the discrete-event dynamic system [6].
(5) The model structure is static, no increase or decrease, in the number of LP instances during simulation.

3.2. Logical process-based model

A general discrete-event model in event scheduling world view can be represented as follows:

Definition 3.1: Discrete-event model

$$Model \triangleq \langle EventTSet, StateVarSet, INIT, Time \rangle,$$

where

- *EventT Set* holds all of the event types,
- *StateVarSet* stands for the set of state variables,
- *INIT* stands for the initial configuration to *EventTSet* and *StateVarSet*,
- *Time* stands for the time base.

For *StateVarSet*, the set of all evaluation functions to the state variable in it, is defined as $EV\,AL(\{SV_1, ..., SV_M\}) \triangleq \prod_{0<j\leq_M} domain_{sv_j}$. The evaluation function $\eta \in EV\,AL(\cdot)$ maps every $SV_j \in StateVarSet$ to a certain value $v \in domain_{SVj}$, where $0 < j \leq M$.

The LP-based model of a DES consists solely of LPs as atomic model entities, the set of which is marked as Set_{LP}.

Let L be the set of indices. The L-indexed set over Set_{LP} is a triple (L, Set_{LP}, μ), where $\mu : L \to Set_{LP}$ be the index function of Set_{LP}. Since $\forall LP \in Set_{LP} \exists i \in L, \mu(i) = LP$, we mark the LP as LP_i.

The LP-based model of a DES should contain extra partition information on *EventTSet* and *StateVarSet*, respectively, and is formally represented in Definition 3.2.

Definition 3.2: LP-based model

$Model_{LP} \triangleq \langle Set_{LP}, INIT, Time \rangle$, where

- *Time* represents the time base, normally $\Re_0^+$ is used.
- $Set_{LP} \triangleq \{LP_i | i \in L\}$, where
  $\{LP_i \triangleq \langle EventTSubset_i, StateVarSet_i, \delta_i^S, \delta_i^\eta \rangle, i \in L\}$ where

  (1) $EventTSubset_i$ is a subset of *EventTSet*, and satisfies:

  $$\forall i \in L. EventTSubset_i \subseteq EventTSet$$
  $$\cup_{i \in L} EventTSubset_i = EventTSet$$
  $$\forall i \neq j \in L. EventTSubset_i \cap EventTSubset_j = \phi$$

  (2) $StateVarSet_i$ is a subset of *StateVarSet*, and satisfies:

  $$\forall i \in L. StateVarSet_i \subseteq StateVarSet$$
  $$\cup_{i \in L} StateVarSet_i = StateVarSet$$
  $$\forall i \neq j \in L. StateVarSet_i \cap StateVarSet_j = \phi$$

  (3) δ_i^S is the state transition function, and is represented as $\delta_i^S : EventTSubset_i \times S_i \to S_i, i \in L$.

  (4) δ_i^η is the event scheduling function, and is represented as $\delta_i^\eta : EventTSubset_i \times S_i \to \wp(EventTSet \times Time), i \in L$, where S_i is the state space of LP_i:
  $S_i \triangleq EV\,AL(StateVarSet_i) \triangleq \prod_{SV} \in StateVarSet_i domain_{SV}$ and the state space of $Model_{LP}$ is defined as $S \triangleq EV\,AL(StateVarSet) \triangleq \prod_{i \in L} S_i$.

- *INIT* stands for the initial configuration to $EventTSet_i$ and $StateVarSet_i$, where $i \in L$.

The schedule relationship for event types of LP_i, marked as $ScheduleRelationship_i$, is a set of tuples consisting of an event type in LP_i and those event types scheduled by it.

$$ScheduleRelationship_i \triangleq \{(eT_{from}, eT_{to}) |$$
$$(eT_{from} \in EventTSubset_i) \wedge (\exists s \in S_i, \exists t \in Time, s.t. (eT_{to}, t) \in \delta_i^\eta (eT_{from}, s))\}$$

The schedule relationship for $Model_{LP}$ is

$$ScheduleRelationship \triangleq \cup_{i \in L} ScheduleRelationship_i$$

3.3. Formal description

The EG formalism describes the elements of an EG model, which is shown in Definition 3.3 [11].

Definition 3.3: Event graph

$M \triangleq < V, E, S, F, C, T, A, B >$, where

- V, the set of vertices corresponding to the events of M,
- E, the set of directed edges $e_{od} = < v_o, v_d >$ that describe the scheduling/cancelling relationships between pairs of events in M,
- S, the state space of M, it is depicted by a vector of state variable in DES,
- $F = \{f_v : S \rightarrow S, \forall v \in V\}$, the set of state change functions, each function associates with a vertex v,
- $C = \{c_e : S \rightarrow \{True, False\}, \forall e \in E\}$, the set of condition functions, each function associates with an edge e,
- $T = \{t_e : S \rightarrow Time, \forall e \in E\}$, the time delay on each edge e(normally we use $\Re_0^+$ as Time Base),
- $A = \{A_e, \forall e \in E\}$, the set of attribute lists, if any, and each list associates with an edge e,
- $B = \{B_v, \forall v \in V\}$, the set of parameter lists, if any, and each list associates with a vertex v.

$\wp(H)$ marks a set consisting of the set of all subsets of a set H, the so-called powerset: $\wp(H) = \{x | x \subseteq H\}$.

Definition 3.4: Partition

For any set H, set $Par \subseteq \wp(H)$ is a partition of H, if $(\forall I, I' \in Par. I \neq I' \Rightarrow I \cap I' = \phi) \wedge (\sqcup_{I \in Par} I = H)$

The partition of set H is marked as P_H.

Definition 3.5: Projection

For event graph M, $Projection : V \rightarrow StateVarSet$ is defined as the set of all state variables accessed by a subset of V.

Definition 3.6: Partitioned event graph

The PEG corresponding with an event graph $M \triangleq < V, E, S, F, C, T, A, B >$ is $PEG_M \triangleq \langle M, P_V \rangle$, where $\{Projection(I) | I \in P_V\}$ is a partition of $StateVarSet$.

The equivalence between the PEG and the LP paradigm is proved by generating a bidirectional map between them.

The introduction of attribute A and parameter B in Definition 3.3 is a notational convenience and does not bring change to the modelling power of the EG [22]. The method for removing parameters from the EG is presented in [11] and the EG-based model with cancelling edges can be transformed into an equivalent form with only scheduling edges [46]. Therefore, we only consider EG without attributes, parameters, and cancelling edges in the discussion. First, transformation from LP-based model to the corresponding EG-based model is performed. The corresponding EG model of the Set_{LP} is defined as:

$$EG_{Set}\underline{\underline{\Delta}}_{LP}\langle EventTSet, ScheduleRelationship, StateSpace, \cup_{i\in L}F_i, \cup_{i\in L}C_i, \cup_{i\in L}T_i\rangle.$$

.

Since each event type belongs, and only belongs, to a single LP, which means that

$$\forall_e T \in EventTSet \Rightarrow (\exists i \in L \wedge eT \in EventTSubset_i) \wedge \big(\forall i, j \in L \Rightarrow \big(eT \in EventTSubset_i \wedge eT \in EventTSubset_j \Rightarrow i = j\big)\big),$$

there exists a function Γ: $EventTSet \rightarrow L$, which maps each $eT \in EventTSet$ to the index i of the $EventTSubset_i$ it belongs to.

By applying a curry operation to δ_i^S, the state transformation function for LP_i is defined as:

$$F_i\underline{\underline{\Delta}}\big\{f_{eT} : S_i \rightarrow S_i | \forall eT \in EventT\,Set, f_{eT} = \delta_i^S(eT), i = \Gamma(eT)\big\}.$$

.

The corresponding event scheduling function on the edge set of LP_i is defined as:

$$C_i\underline{\underline{\Delta}}\{c_r : S_i \rightarrow \{True, False\} | \forall r \in Schedule\,Relationship_i, \forall s \in S_i, c_r(s) = IsTrue(r.to \in \pi_{EventT\,Set}\delta_i^\eta(r.from, s))\},$$

where $\pi_{EventT\,Set}$ is the projection function from $EventTSet \times Time$ to $EventTSet$.

The time delay function on the edge set of LP_i is defined as:

$$T_i\underline{\underline{\Delta}}\,t_r : S_i \rightarrow Time | \forall r \in Schedule\,Relationship_i \forall s \in S_i t_r(s) = \pi_{Time}\delta_i^\eta(r.from, s),$$

where π_{Time} is the projection function from $EventTSet \times Time$ to $Time$.

Through the transformation procedure described above, an EG-based model $EG_{set_{LP}}$ is generated from Set_{LP}.

Define $P_E\underline{\underline{\Delta}}\{EventT\,Subset_i, \forall i \in L\}$, according to the definition of $EventTSubset_i$ and Definition 3.4, P_E is a partition of $EventTSet$. On the one hand, since $StateVarSet_i$ and $EventTSubset_i$ belong and only belong to a single LP_i, there is a bidirectional map from $P_{StateVarSet}$ to P_E, on the other hand,

$$\{Project(StateVarSet, x) | x \in P_E\} = \{StateVarSet_i, i \in L\}$$

is a partition of the state variable set *StateVarSet*, the constraint in Definition 3.6 is satisfied.

The PEG-based model constructed from Set_{LP} is $PEG_{Set_{LP}} \underline{\underline{\Delta}} \langle EG_{Set_{LP}}, P_E \rangle$, and $PEG_{Set_{LP}}$ is a PEG-based representation of Set_{LP}.

We have proved that any LP-based discrete-event model can be transformed into an EG-based model.

Second, a transformation is constructed from an EG-based model to its corresponding LP-based model. Since for the PEG-based model $\{Projection(I)|I \in P_V\}$ is a partition of *StateVarSet*, the construction of the map is similar to the procedure above. The *LP*s are constructed according to P_V and $\{Project(StateVarSet, x)|x \in P_V\}$. Particularly, $M = \langle V, E, S, F, C, T, A, B \rangle$ corresponds to a discrete-event model in event scheduling world view, which is $Model_{LP}$ with $|L| = 1$.

We have proved that any discrete-event model in PEG formalism can be described by the LP Paradigm.

In summary, the bidirectional map between LP-based model and PEG-based model exists.

3.4. The structural operational semantics

The SOS of the PEG is defined on an abstract machine with TLTS, which is an extended Labelled-Transition System.

Definition 3.7: Labelled-transition system (LTS)

$LTS \underline{\underline{\Delta}} \langle S, s_0, Label, \rightarrow \rangle$, where

- S is the state space.
- $s_0 \in S$ is the initial state.
- *Label* is the set of all actions, which stands for the event types.
- $\rightarrow$ is the set of time-passage transitions, and $\rightarrow \subseteq S \times Label \times S. s_a \overset{action}{\longrightarrow} s_b$ stands for $(s_a, action, s_b) \in \rightarrow$, in which $s_a, s_b \in S$, $action \in Label$.

Definition 3.8: Timed-Labelled Transition System

$$TLTS = \langle S, s_0, Label, \rightarrow, \mapsto \rangle,$$

where

- S, s_0, *Label* and $\rightarrow$ are defined in the same way as in Definition 3.7.
- $\mapsto$ is the set of time-passage transitions, and $\mapsto \subseteq S \times R^+ \times S. s_a \overset{delay}{\mapsto} s_b$ stands for $(s_a, delay, s_b) \in \mapsto$, in which $s_a, s_b \in S$, $delay \in R^+$.

A TLTS contains two kinds of transitions: time-passage transitions corresponding to R^+ and discrete transitions.

Based on the definitions above, we derive the definition of the SOSs of EG and PEG. The execution of an EG model is on an abstract machine whose configuration (global state) keeps track of the state of the component and the time of the last transition. The configuration of a PEG-based model is composed from the configuration of all partitions. The time used in the following definition refers to simulation time [2].

3.4.1. Operational semantics of EG

The SOS of the EG model is represented by

$$TLTS(EG) = <Config, config_0, Label, \rightarrow, \mapsto>,$$

in which:

- *Config*
  $\triangleq STATE \times Time \times \wp(EventT\ Set \times Time) \times (EventT\ Set \times Time) \times PHASE$

  Config is the state space, and $\forall state \triangleq \langle s, \tau, \Lambda, Active, Phase\rangle \in Config$:

  (1) $s \in STATE$, $STATE = EVAL(StateVarSet).\ EVAL(StateVarSet)$ corresponds to the model's *StateSpace* in the LP paradigm. *StateVarSet* stands for the set of state variables in EG-based model. The type of the state variable v is defined as $dom(v)$. $\eta \in EVAL(StateVarSet)$ is an evaluation function for the *StateVarSet*. *Cond(StateVarSet)* is the set of Boolean expressions on *StateVarSet*.
  (2) $\tau \in Time$ stands for the simulation time, *delay* stands for the simulation time delay.
  (3) $\Lambda \in \wp(EventT\ Set \times Time)$ is the set of pending events. Define $\Lambda.head$ as the reference to the event in Λ with the lowest timestamp, if the set is empty.
  (4) $Active \in EventTSet \times Time$ is the cursor to the currently executing event if any. $Active.Type \in V \cup \{NULL\}$, *Type* stands for the Event Type of the cursor, $Type = NULL$ means that there is no event executing currently.
  (5) $Phase \in PHASE \triangleq \{Elapse, Scheduling\}$, in which the event retrieval operation from the pending event set and the execution operation are treated as a single transaction.
  (6) The time base is represented by Time, which is $\Re_0^+$ in this paper.

- $config_0$ is the initial state.
- $Label \triangleq V \cup E$, which is the union of the edge set and the vertex set of the PEG.
- Transition relationships $\rightarrow$ and $\mapsto$ are defined in below:
  (1) Event Execution

$$\frac{(Phase = Elapse) \wedge (\Lambda.head.TYPE = EGvertex) \wedge (\tau = \Lambda.head.timestamp)}{\langle s, \tau, \Lambda, Active, Phase\rangle \overset{EGvertex}{\longrightarrow} \langle s', \tau, \Lambda', Active', Phase'\rangle},$$

$$\text{where } (s' = f_{EGvertex}(s), Active' = \Lambda.head, \Lambda' = \Lambda - \{Active'\}, Phase' = ExeSch, EGvertex \in V);$$

(2) Event Scheduling

$$\frac{(Phase = Scheduling)\Lambda(Active.TYPE = EGvertex_i) \wedge \left(\eta\left(EGedge_{ij}.c\right) = True\right)}{\langle s, \tau, \Lambda, Active, Phase\rangle \overset{EG_{edge_{ij}}(c,t)}{\longrightarrow} \langle s, \tau, \Lambda', Active', Phase'\rangle},$$

where $\left(\Lambda' = \Lambda \cup \left\{\left\langle EGvertex_j, \tau + 1\right\rangle\right\}, Active'.TYPE = NULL, Phase' = Elapse, EGvertex_i \text{ and } EGvertex_j \in V, EGedge_{ij} \in E\right)$;

(3) Time Elapsing

$$\frac{(Phase = Elapse) \wedge (\tau + delay < \Lambda.head.timestamp)}{\langle s, \tau, \Lambda, Active, Phase\rangle \overset{delay}{\longrightarrow} \langle s, \tau + delay, \Lambda, Active, Phase\rangle},$$

where *(delay ∈ Time).*

3.4.2. *Operational semantics of PEG*

In order to define the SOS of the PEG-based model, we first transform $PEG_M \underline{\underline{\Delta}} \langle M, P_{StateVarSet}, P_E\rangle$ into the equivalent style $PEG \underline{\underline{\Delta}} \{LP_0, ..., LP_{n-1}\}$, where

$$LP_i \underline{\underline{\Delta}} \{<StateVarSet_i, EventTSubset_i> | i \in L\}, (0 \leq i < n).$$

The SOS of the PEG-based model is represented as a TLTS:

$$TLTS(PEG) = <Config, config_0, Label, \rightarrow, \mapsto>,$$

in which

- *Config*
  $\underline{\underline{\Delta}} SRATE \times Time \times \wp(EventSet \times Time) \times (EventSet \times Time) \times PHASE$

  Config is the state space, and $\forall state \underline{\underline{\Delta}} \langle s, \tau GVT, \Lambda, Active, Phase\rangle \in Config$:
  (1) $s = \left[s_1, \ldots, s_{|L|}\right]^T, s_i \in S_i, 1 \leq i \leq |L|$.
  (2) $\Lambda = \left[\Lambda_1, \ldots, \Lambda_{|L|}\right]^T, 1 \leq i \leq |L|$, which means that Λ is the composition of pending event sets of all *LP*s, and globally unique.
  (3) $Active \in EventTSet \times Time$ is the cursor to the currently executing event if any.
  (4) The notation, $\tau \underline{\underline{\Delta}} \tau GVT = Min_{i \in L}(lvt_i)$, is the global simulation time in which lvt_i is the local simulation time of LP_i.
  (5) $Phase \in PHASE = \{\text{Elapse}, \text{Scheduling}\}$.
- $config_0$ is the initial state.
- $Label \underline{\underline{\Delta}} V \cup E$.
- The time base is represented by *Time*, which is $\Re_0^+$ in this paper
- Transition relationships $\rightarrow$ and $\mapsto$ are defined in the following rules:

(1) Event Execution

$$\frac{(EGvertex \in EventT\ Subset_i) \wedge (Phase = Elapse) \wedge (\Lambda.head.TYPE = EGvertex) \wedge (\tau = \Lambda.head.timestamp)}{\left\langle \begin{bmatrix} \cdots \\ s_i \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ lvt_i \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ \Lambda_i \\ \cdots \end{bmatrix}, Active,\ Phase \right\rangle \overset{EGvertex}{\longrightarrow} \left\langle \begin{bmatrix} \cdots \\ s_i^{'} \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ lvt_i \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ \Lambda_i^{'} \\ \cdots \end{bmatrix}, Active',\ Phase' \right\rangle}$$

where $(s' = f_{EGvertex}(s),\ Active' = \Lambda.head, \Lambda' = \Lambda - \{Active'\}, Phase' = Scheduling, EGvertex \in V)$.

.

(2) Event Scheduling

$$\frac{(Phase = Elapse) \wedge (\tau_{GVT} + delay < \Lambda.head.timestamp)}{\left\langle \begin{bmatrix} s_1 \\ s_i \cdots \\ s_{|L|} \end{bmatrix}, \begin{bmatrix} lvt_1 \\ \cdots \\ lvt_i \\ \cdots \\ lvt_{|L|} \end{bmatrix}, \begin{bmatrix} \Lambda_1 \\ \cdots \\ \Lambda_i \\ \cdots \\ \Lambda_{|L|} \end{bmatrix}, Active,\ Phase \right\rangle \overset{delay}{\mapsto} \left\langle \begin{bmatrix} s_1 \\ \cdots \\ S_i \\ \cdots \\ S_{|L|} \end{bmatrix}, \begin{bmatrix} lvt_1 + delay \\ lvt_i \cdots + delay \\ \cdots \\ lvt_{|L|} + delay \end{bmatrix}, \begin{bmatrix} \Lambda_1 \\ \cdots \\ \Lambda_i \\ \cdots \\ \Lambda_{|L|} \end{bmatrix}, Active,\ Phase \right\rangle}$$

where $(delay \in Time)$.

(3) Time Elapsing

If the vertex of the scheduling edge crosses the partition boundary it is identified as external event scheduling. If not, then it is internal.

External event scheduling:

If $(EGvertex_i \in Event\ T\ Subset_m \wedge EGvertex_j \in EventTSubset_k \wedge k \neq m)$

$$\frac{(Phase = Scheduling) \wedge (Active.TYPE = EGvertex_i) \wedge \left(\eta\left(EGedge_{ij}.c\right) = True\right)}{\left\langle \begin{bmatrix} \cdots \\ s_m \\ \cdots \\ s_k \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ lvt_m \\ \cdots \\ lvt_k \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ \Lambda_m \\ \cdots \\ \Lambda_k \\ \cdots \end{bmatrix}, Active,\ Phase \right\rangle \overset{EGedge_{ij}(c,t)}{\longrightarrow} \left\langle \begin{bmatrix} \cdots \\ s_m \\ \cdots \\ s_k \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ lvt_m \\ \cdots \\ lvt_k \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ \Lambda_m \\ \cdots \\ \Lambda'_k \\ \cdots \end{bmatrix}, Active',\ Phase' \right\rangle},$$

where $\left(\Lambda'_k = \Lambda_k \cup \left\{\left\langle EGvertex_j, lvt_m + t\right\rangle\right\}, Active'.TYPE = NULL,\ Phase' = Elapse,\ EGvertex_i \in V, EGedge_{ij} \in E\right)$.

Internal event scheduling:

If $EGvertex_i,\ EGvertex_j \in EventTSubset_m$

$$\frac{(Phase = Scheduling) \wedge (Active.TYPE = EGvertex_i) \wedge \left(\eta\left(EGedge_{ij}.c\right) = True\right)}{\left\langle \begin{bmatrix} \cdots \\ s_m \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ lvt_m \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ \Lambda_m \\ \cdots \end{bmatrix}, Active,\ Phase \right\rangle \overset{EGedge_{ij}(c,t)}{\longrightarrow} \left\langle \begin{bmatrix} \cdots \\ s_m \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ lvt_m \\ \cdots \end{bmatrix}, \begin{bmatrix} \cdots \\ \Lambda'_m \\ \cdots \end{bmatrix}, Active',\ Phase' \right\rangle},$$

where

$$\left(\Lambda'_m = \Lambda_m \cup \left\{\left\langle EGvertex_j lvt_m + t\right\rangle\right\}, Active'.TYPE = NULL, Phase' = Elapse, EGvertex_i, \in V, EGedge_{ij} \in E\right).$$

.

3.4.3. Wallclock time-based execution

The SOS of PEG in Section 3 is strictly sequential, hence the sequence of event execution in Wallclock time strictly conforms to the sequence in simulation time, which is non-decreasing timestamp order.

The Wallclock time execution of TLTS and its equivalence relationship based on local causality constraint (LCC) [2] are introduced in order to study the concurrent execution of the LP-based model in PEG formalism. The following definitions are also introduced:

Definition 3.9: *I – Trace*
ω: $I \rightarrow S$, in which I represents the closed interval start from time 0, and S represents the state space of the system, in which $\forall t$, then t', if $t \leq t'$ then $\omega(t) \xrightarrow{t'-t} \omega\ (t')$, ωlt represents the supremum of I, and ω. ft represents the infimum of I. The $\omega.ls$ represents the final state, and $\omega.fs$ the initial state $\omega(0)$.

For example, $config \xrightarrow{d} config'$ corresponds to the $[0, d]$ – *Trace*, in which $\omega.fs = config$, $\omega.ls = config'$.

Definition 3.10: Timed Execution of TLTS
The Timed Execution of TLTS is an alternating sequence $\gamma = \omega_0 a_1 \omega_1 a_2 \ldots \omega_{n-1} a_n \ldots$ in which ω_i represents I-trace, a_i represents event type, and $\omega_i.ls \xrightarrow{a_i+1} \omega_{i+1}.fs$. The duration and initial state of γ is marked as $\gamma.lt$ and $\gamma.fs$, respectively. $\gamma.fs = \omega_0.fs$, $\gamma.ft = \omega_0.ft$. For limited Timed Execution, formulae $\gamma.lt = \omega_n.lt$, $\gamma.ls = \omega_n.ls$ hold. The set of all possible Timed Execution of *TLTS* is marked as *execs(TLTS)*.

Definition 3.11: Wallclock time execution of TLTS
The Wallclock time execution of TLTS is formed by attaching the Wallclock timestamp $a_i.wct$ to each discrete-event type a_i in $\gamma = \omega_0 a_1 \omega_1 a_2 \ldots \omega_n - 1 a_n \ldots$ The actual execution sequence of TLTS in Wallclock time is $\gamma_{WC} = \omega_{WC_0} a_{WC_1} \omega_{WC_2} \ldots \omega_{WC_{n-1}} a_{WCn} \ldots$. Define the set of all possible Wallclock time execution of *TLTS* as $execs_{WC}$*(TLTS)*.

If each LP adheres to the LCC, then the parallel execution will yield exactly the same results as a sequential execution of the same model, provided that simultaneous events are processed in the same order in both cases [2, p. 53].

This allows LCC to be regarded as a selection of all executions without causal error from $execs_{WC}$ (*TLTS*). The set of all Wallclock time executions of *TLTS* qualifying LCC is defined as $execs_{WC}^{LCC}(TLTS) \subseteq execs_{WC}(TLTS)$, and event is defined as $e_i \underline{\Delta} \langle a_i, \omega_i ft \rangle$, with event type $e_i.type = a_i$, and execution time $e_i.st = \omega_i.ft$.

If there is no causality between e_i and e_j, which is marked as $e_i \| e_j$, then exchanging e_i with e_j in the Wallclock time execution, does not alter the result. This is stated by the following formula.

For executions

$$\gamma_{WC} = \omega_{WC_0} a_{WC_1} \omega_{WC_1} \ldots a_{WC_i} \omega_{WC_i} \ldots a_{WC_j} \omega_{WC_j} \ldots$$
$$\gamma'_{WC} = \omega_{WC_0} a_{WC_1} \omega_{WC_1} \ldots a_{WC_j} \omega'_{WC_j} \ldots a_{WC_i} \omega'_{WC_i} \ldots$$

The following solutions are held to be true:

$$\omega_{WC_j}.fs = \omega'_{WC_i}.fs, \gamma WC, \gamma'_{WC} \in execs^{LCC}_{WC}(TLTS), \text{and } \gamma WC \neq \gamma'_{WC}.$$

In conclusion, even though parallel executions using different parallel time management algorithms can generate different Wallclock time executions, these executions result in the same final system state as the SOS describes in Section 3, as long as the time management algorithms all adhere to the LCC.

4. PEG-based model transformation framework for PDES

In this section, a model transformation framework for PDES is proposed. In this framework, models transform through three phases, which are DSL, PEG formalism, and platform-specific API.

Four personnel roles will be described. First, the abstract Grammar of PEG language, named $Grammar_{PEG}$, is laid out; second, the Framework is discussed in detail; third, the transformation from the general EG to a PEG-based Model is accomplished by implementing imparametrization and partition. This approach is key to the application of the PEG.

4.1. Grammar of the PEG language

The PEG language grammar, $Grammar_{PEG}$, is described in this section. $Grammar_{PEG}$ specifies how to analyse the model and provides the partition information of the PEG. A use case of $Grammar_{PEG}$ is presented.

The Extended Backus–Naur Form (EBNF)-styled grammar of PEG is shown in Figure 1, in which *Function* employs the ECMAScript grammar [47].

The single airport model [2, p. 34], in PEG language, is demonstrated in Algorithm 1.

4.2. The PEG-based domain-specific modelling framework for PDES

In this framework, the model in DSL is transformed into an executable model through the PEG formalism, as shown in Figure 2. The transformation has three phases. First, the transformation from DSL to EG is performed; second, the resulting EG-based model is partitioned into a PEG-based model; and, third, the PEG-based model is transformed into an executable model on specific PDES platform.

Simulation experiments are conducted after further configuration to Random Number Generator, Queue Algorithms, and Time Management Algorithms. Briefly, with PEG, the tripartite phasing decouples DSL from specific simulation platform API.

Algorithm 1: Single Airport Model in PEG Language

```
1 model airport;
2 StateVar {
3           integer ln_.The.Air 10;
4           integer On_The_Ground 0;
5           boolean Runway_Free True;
6           double R 10.0;
7           double G 15.0;
8}
9
```

```
10 EventPartition EventSetl {
11          Event ArrivalEvent = "ARRIVAL" ();
12          Event DepartureEvent = "DEPARTURE" ();
13          Event LandedEven t = "LANDED" ();
14 }
15
16 Edges {
17      Edge Arrival_Landed {
18          type schedule
19          events (ArrivalEvent, LandedEvent)
20          condition [isRunwa_Free = True]
21          priority default
22          delay 0
23      }
24      Edge Landed_Landed {
25          type schedule
26          events (LandedEvent, LandedEvent)
27          condition [True]
28          priority default
29          delay 0
30      }
31      Edge Landed_Departure {
32          type schedule
33          events (LandedEvent, DepartureEvent)
34          condition [get In _The_Air >0]
35          priority default
36          delay 0
37      }
38 }
```


Four roles are involved in the PEG-based PDES of domain-specific models (DSMs):

(1) Expert A: an expert with Domain Knowledge, familiar with domain knowledge, models the domain problem using the selected DSL.
(2) Expert B: an expert with DES Modelling Knowledge, familiar with the grammar, semantics, and analysis of EG formalism, models the domain problems using EG.
(3) Expert C: an expert with Language Analyser Construction Knowledge develops language interpreter, parser and lexical analyser in order to further transform the DSL into the target language.
(4) Expert D: an expert with Parallel Simulation Platform Knowledge, who is familiar with the experiment configuration and software/hardware deployment customized towards a specific simulation platform.

Generally speaking, the model transformation is divided into three phases: model-to-model transformation, model reconstruction, and model to code transformation [48].

The transformation from a DSM to the general EG-based model is a model-to-model transformation. In the cooperation of experts of four roles, information is passed through several abstraction levels and aspects of the system.

```
1  //a model has a name and a number of definitions (states, event partitions, edges)
2  MODEL ::= 'model' Identifier ';'STATEVARIABLES EVENTPARTITION EDGELIST
3
4  //the state variables
5  STATEVARIABLES ::=    'StateVar {' StateVariableDeclaration (';'
       StateVariableDeclaration) '}'
6  StateVariableDeclaration ::= Variable Assignment
7
8  //a variable has a type and a name
9  Variable      ::= BasicType Identifier
10 BasicType    ::= 'string' | 'double' | 'integer' | 'boolean'
11 Identifier     ::= ('a'.. 'z' | 'A'.. 'Z' | '_') ('a'.. 'z' | 'A'.. 'Z' | ' '|'0'..
       '9')*
12 Assignment ::= Number | STRING | Function
13 STRING       ::= (any character)*
14 Number       ::= intNumber | realNumber
15
16 //an event partition definition is being composed from the events defined for it
17 EVENTPARTITION::= 'EventPartition' Identifier '{' Event* '}'
18
19 //each event has a name, and a string representation and a set of parameters
20 EVENT::=   'Event' Identifier '=' STRING '('Identifier ')''('ParameterSet')' ';'
21 // state transition function F
22 STATETRANSITION::= " StateTransition"  Identifier '=' Function;
23 ParameterSet ::= (Variable';')*
24
25 //edge definitions
26 EDGELIST::=   'Edges' '{' EDGE* '}'
27 EDGE::=   'Edge' Identifier '{'  'type' EdgeType 'events ('eventPair')' 'condition
       [' Condition ']' 'priority' Priority 'delay' TimeDelay ('functions ('
       (Function ';')* )')+ '}'
28 EdgeType ::= 'Schedule' |'Cancel'
29 EventPair ::= Identifier Identifier
30 Condition ::= 'True' | 'False' | Function
31 Priority ::= 'default' | 'lowest' | 'lower' | 'low' | 'high' | 'higher' | 'highest'
32 TimeDelay ::= Number | Function  // lvt+a.random()
```

Figure 1. Grammar of PEG.

The transformation from PEG-based model to simulation platform-specific API description is a model to code transformation. First, the general EG-based model is transformed into a parameter-free EG-based model by the parameter expansion algorithm. Next, the parameter-free EG-based model is transformed to a PEG-based model. Both are model reconstructions.

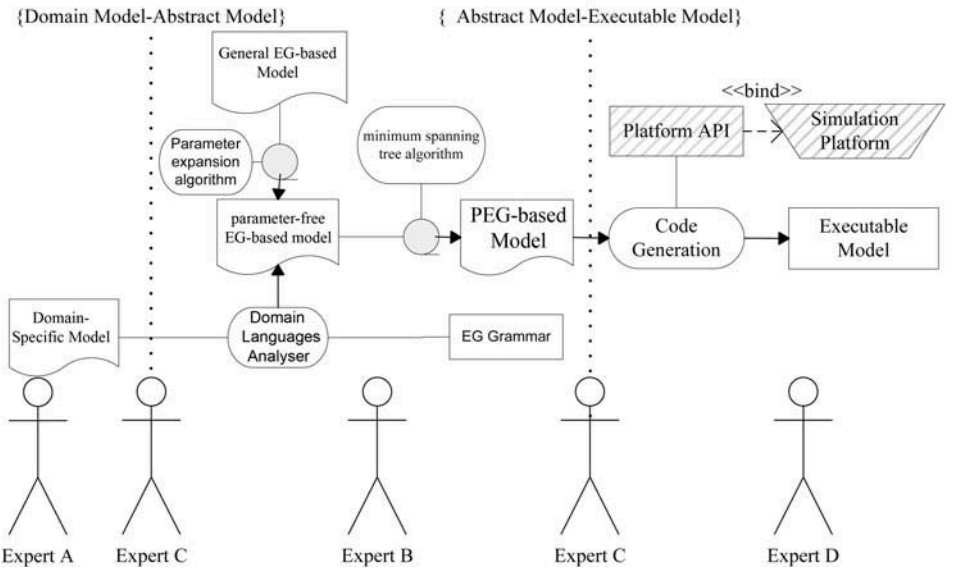

Figure 2. PEG-based model transformation framework for PDES.

The tripartite transformation provides flexibility at two points. The first phase enables a platform-independent representation and a formal analysis of the model's property. The second phase eases the cross-platform migration by focusing the map from PEG formalism, instead of reforming the models each time from the very beginning.

4.3. *Transforming general EG- to PEG-based model*

Transforming general EG formalism into PEG formalism is carried out as follows:

- Imparametrization of the parametric EG-based model

This procedure eliminates the attribute and the parameter. The preconditions of this operation are the domain of parameter on the node, and the *domain* of the attribute on the edge are, respectively, either finite, or can be transformed into a partition with finite cardinality. A partition is a set of nonempty subsets of the *domain* such that every *domain* element is in exactly one of these subsets. The parameter expansion method is shown in Algorithm **2**, in which the first loop is an expansion through *Domain*([*i*]). Since *Domain* ([*i*]) and *Domain*([*j*]) are both finite sets, this loop contains finite operations.

Algorithm 2: Imparametrization Algorithm for General EG-based Model

```
1  V_temp = ∅, E_temp = ∅
2  for each v in V with parameter [i] //Iteration through the vertex Set
3      add v to V_temp
4      for each i in Domain([i]) //Expand the Parameter Set of single vertex
5          if (v_i ∈ V = = False)
6              add V_i to V
7              //Generate vertex, state change function, and state variable set according to i
8              generate V_i, f_vi, SV_i
9          endif
10         for each e ∈ < v, * > //expand the attributes on each edge starting from v
11             add e to E_temp
12             //Domain([j]) is the expression set defined on V × E × Domain([i])
13             if e has attribute as j(i) //if attribute [j] of e is defined on [i].
14                 domain = Domain([j (i)])
15             else                    //if attribute [j] of e is not defined on [i].
16                 domain = Domain([j])
17             endif
18             for each j in domain
19                 generate e_j = < v_i,v'_j >
20                 if (e_j ∈ E = = false)
21                     t_ej = t_e(j)
22                     c_ej = c_e(j)
23                     add e_j to E
24                 endif
25             endfor
26         endfor
27     endfor
28 endfor
29 V = V – V_temp
30 E = E – E_temp
```

- Generating partitions for state variable set and event set

The influencer functions (*influencer*) and influencee functions (*influencee*) in all event vertexes to state variables are generated for the EG-based model in order to reflect the causality between state variables and events in literature [49,50]. The notation f_v *.write* represents the set of state variables modified by the state transition function of event type vertex *v*. The notation f_v *.read* represents the set of state variables read by the state transition function of *v*. The notation t_e*.read* represents the set of state variables read by the time increment function *t*, on edge *e*. The notation c_e*.read* represents the set of state variables read by the condition function *c* on edge *e*. Iterating through the EG-based model obtains these sets. $OutCom_v$ stands for the set of outcoming edges started from vertex *v*.

$$influencer = \{V \to \wp(SV) | \forall sv \in SV, sv \in influencer(v) \Leftrightarrow sv \in f_v\} \tag{1}$$

The notation *influencer* stands for the set of state variables modified by state transition function f_v of vertex *v*.

$$\begin{aligned} influencee \ = \Big\{ & V \to \wp(SV) | \forall sv \in SV, sv \in influencee(v) \\ & \Leftrightarrow sv \in f_v.read \cup \bigcup_{e \in OutCom_v} (t_e.read \cup c_e.read \Big\}) \end{aligned} \tag{2}$$

$$\text{Project} = \{V \to \wp(SV) | \forall sv \in SV,\ Project\,(v) = influencee(v) \cup influencer(v)\} \tag{3}$$

Construct directed bipartite $G_b = (V, SV, E_b), E_b = E_{write} \cup E_{read}$ based on EG.

$$E_{write} = \{<u_v, u_{sv}> \in V \times SV | u_{sv} \in influencer(u_v)\} \tag{4}$$

$$E_{read} = \{<u_{sv}, u_v> \in SV \times V | u_{sv} \in influencee(u_v)\} \tag{5}$$

- Construct the Adjacency Matrix of bipartite *G* as $A(G)_{|V| \times |SV|} = (a_{ij})$, in which $a_{ij} \in \{-1, 0, 1\}$.

$$a_{ij} = \begin{cases} -1 \Leftrightarrow <u_i, sv_j> \in E_{read} \cup E_{read} \\ 0 \Leftrightarrow <u_i, sv_j> \in \{V \times SV - E_{read} - E_{write}\} \\ 1 \Leftrightarrow <u_i, sv_j> \in E_{write} - E_{read} \end{cases} \tag{6}$$

- Partition Generation.

The adjacent matrix $A(G') = (|a_{ij}|)$ of the base graph G' of *G* is also with $|V| \times |SV|$ rank, but does not distinguish between influencer and influencee. The set of all connected components of G' is generated by the minimum spanning tree algorithm [51].

Given that G' has ω connected components as, $G_1, G_2, ..., G_\omega (\omega \geq 2)$ where G_i is not empty, by proper arranging the row/line of *G*, we can get:

$$M(G') = \begin{pmatrix} M(G'_1) & & 0 \\ & M(G'_2) & \\ & & \ddots \\ 0 & & M(G'_\omega) \end{pmatrix} \tag{7}$$

For example, the adjacent matrix of the corresponding undirected underlying graph G' of the discrete-event model in Figure 3(a) is:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$\omega = 2$ corresponds to the situation in Figure 3(b).

- Further partition of a single connected component

If $\omega = 1$ or, the estimated computational load of a single connected component exceeds the single criteria, then further partitions are required which will result in (Read/Write) access to the stat variables across different partitions. Shadow events and state variables are introduced in order to enforce boundary constraints requiring that each event in each partition access only its own state variables.

(1) Write access across partitions: a shadow event is created for each write access to state variable that crosses partitions, shown in Figure 3(c).
(2) Read access across partitions: two shadow events and a shadow state variable are created for each read access to state variable that crosses partitions, shown in Figure 3(d).

5. Case study

In this section, we present a DSL for the spatial stochastic simulation of chemical reactions using Abstract Next Subvolume Method (ANSM) [52,53], to demonstrate the capability of the PEG-based framework presented in Section 4. A spatial variation of the Lotka–Volterra System (LVSystem) [54,55] is used as the benchmark model, and described in Figure 4. First, a DSL for the LVSystem model is presented. Second, the DSM is transformed into a generic PEG-based model, and then into the executable code on YHSUPE. Third, we evaluate the performance of parallel simulations.

5.1. Model description

Pronounced oscillatory behaviour is produced using the setup in Gillespie's work [56]. An $N \times N$ *2D* grid is taken as the space for the Lotka–Volterra Model. N denotes the number of subvolumes in the model and represents the scale of the model. $Num_{predator}$ and Num_{prey} stand for the predator and prey the population in a subvolume. Initially there

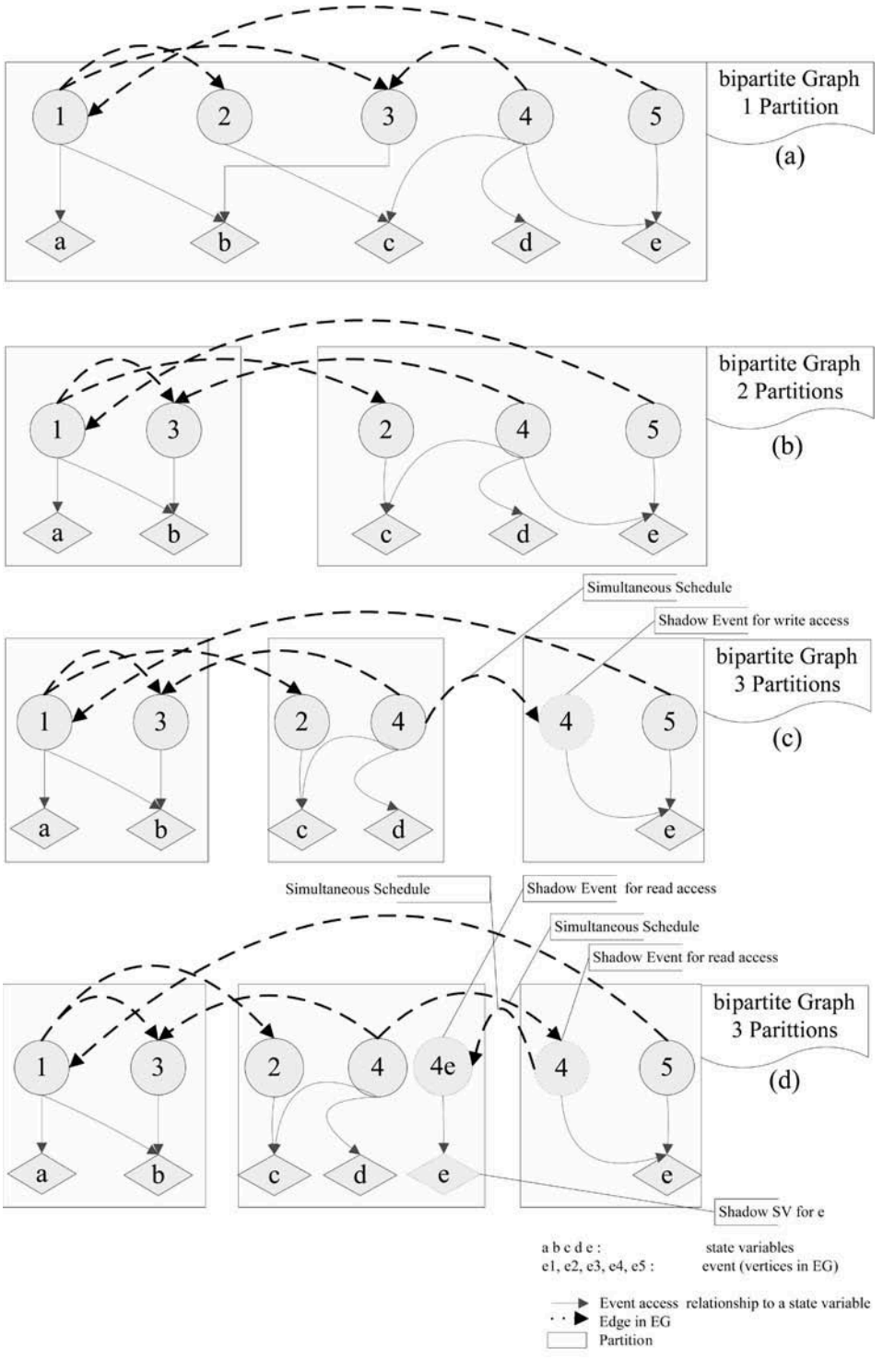

Figure 3. Event graph partition based on bipartite graph.

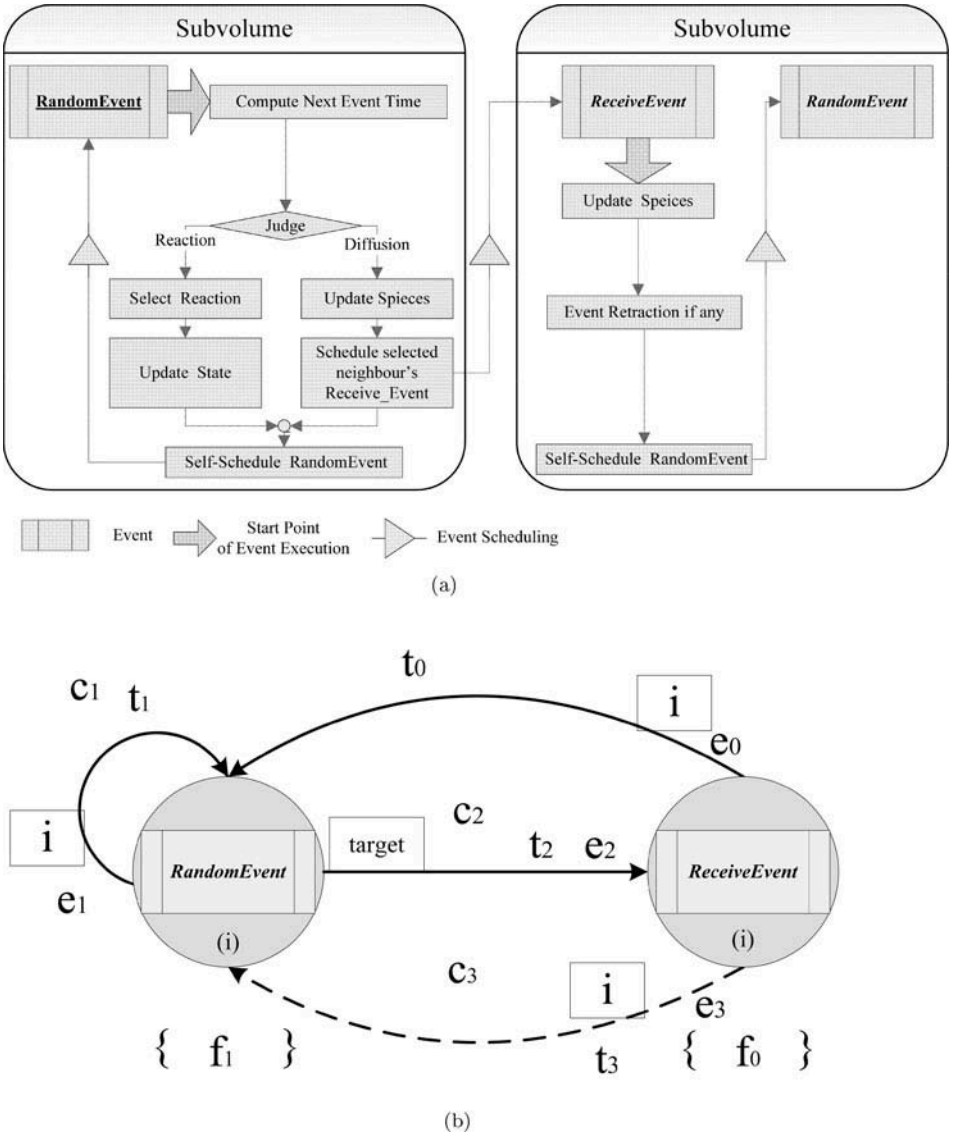

Figure 4. EG-based model description of LVSystem in ANSM. (a) Event processing and scheduling in ANSM. (b) EG-based model description.

are 1000 of each in every subvolume of the *2D* grid. The diffusion rate constants of the predators and preys are noted as $d_{predator}$ and d_{prey}. The reaction constant is noted as c_i. The grass population is assumed to never change. The parameter setup for the LVSystem model is in Table 1.

Figure 4(b) is the parameterized EG-based model for LVSystem in ANSM, in which the *target* is the number of the subvolume. A single subvolume contains the state variables and the event types. State variables define the quantity and quality of the reactants in the subvolume. The event processing and scheduling is shown in Figure 4

Table 1. Model parameters setup.

Population	Diffusion rate constant	Reaction	Reaction rate constant
$Num_{predator} = 1000$	$d_{predator} = 2.5$	$Grass + Prey \xrightarrow{c_1} 2Prey$	$c_1 = 0.01$
$Num_{prey} = 1000$	$d_{prey} = 5$	$Predator + Prey \xrightarrow{c_2} 2Predator$	$c_2 = 0.01$
$Num_{grass} = 1000$	$d_{grass} = 0$	$Predator \xrightarrow{c_3} Null$	$c_3 = 10$

(a) [53], in which *RandomEvent* is an event type which corresponds to self-scheduling, and acts as described in the appendix of [57]. *ReceiveEvent* is an event type corresponding to diffusion, is assumed to be scheduled by an adjacent subvolume, and addresses the reactants diffusing in.

5.2. *Domain-specific language for LVSystem*

The DSM transformation of the LVSystem is performed in the following procedure:

(1) Specify the data and behaviour provided by the DSM, and construct the EG-based model, according to ANSM.
(2) Formulate the grammar of the DSL, and implement the interpreter from that language to the EG formalism using ANTLR [58].
(3) Initialize the EG-based model, and partition the general EG-based model to PEG-based model using Algorithm 4.3.
(4) Implement the interpreter from the PEG formalism to YHSUPE using ANTLR.
(5) Configure the simulation experiment and deploy software/hardware environment according to requirements.

The grammar of the DSL is shown in Figure 5. The model description of the LVSystem is shown in Figure 6.

The domain-specific parallel spatial stochastic simulation subsystem is built on top of the YHSUPE.

After the parameter and attribute expansion of the EG-based LVSystem model using Algorithm **2**, a parameter-free EG-based model is obtained as shown in Figure 7. By dividing the EG-based model into a set of connected components, we get the PEG-based model; the partition is shown in Figure 8.

5.3. *Simulation experiments*

The simulation experiments are executed on a PowerLeader Baode PR4310D Cluster (Shenzhen, Guangdong, P.R. China) with 20 computing nodes. Each computing node is equipped with two Intel 1.6 GHz Quad-Core Xeon Processors (Santa Clara, CA, USA), 8 GB RAM and interconnected with a 10 Gbps Infiniband connection. The operating system on each node is Red Hat 3.4.6–8, with kernel 2.6.9–67.0.7, *EL4ustre*.1.6.5*smp*. The GCC version is 4.1.1. The discs are shared by all computing nodes. All the experiments in this paper were conducted atop YHSUPE. Every logical processor is mapped to a single OS process, and occupies the CPU core assigned as much as possible.

```
1  DomainSpecificMODEL ::= 'name' Identifier ';' SCALE  SPECIES REACTIONS STATE RC RD
2  SCALE          ::=    'scale'  intNumber
3  SPECIES        ::= 'species:' VarSet
4  REACTIONS      ::= 'reactions:' ReactionSet
5  STATE          ::= 'state:' StatevarSet
6  RC             ::= 'rc:'    RcSet
7  RD             ::= 'dc:'    RdSet
8  VarSet         ::= '{'(Identifier ';')* '}'
9  ReactionSet    ::=(Equation)*
10 Equation       ::='{'Identifier '=' intNumber Identifier ('+' intNumber Identifier )+
       '->' intNumber Identifier ('+' intNumber Identifier )+ '}'
11 StatevarSet    ::= VarvalueSet
12 RcSet          ::= VarvalueSet
13 RdSet          ::= VarvalueSet
14 VarvalueSet    ::= '{'(varvalue',')* '}'
15 varvalue       ::= Identifier ':' intNumber
16 Identifier     ::= ('a'..'z' | 'A'..'Z' | '_') ('a'..'z' | 'A'..'Z' | '_'| '0'..'9')*
```

Figure 5. Grammar of the script language for the LVSystem model.

```
1  name: {SaptialLotkaVolterra}
2  scale X
3  \\[Subvolume]
4  species: {A;B;C}
5  reactions:
6          {r1 = 1A + 1B -> 2B+A}
7          {r2 = 1C + 1B -> 2C}
8          {r3 = 1C -> 0C}
9  \\[parameters]
10 state: {A:1000, B:1000, C:1000}
11 rc: {r1:0.01, r2:0.01, r3:10}
12 dc: {d1:0, d2:5, d3:2.5}
```

Figure 6. Model description for the LVSystem.

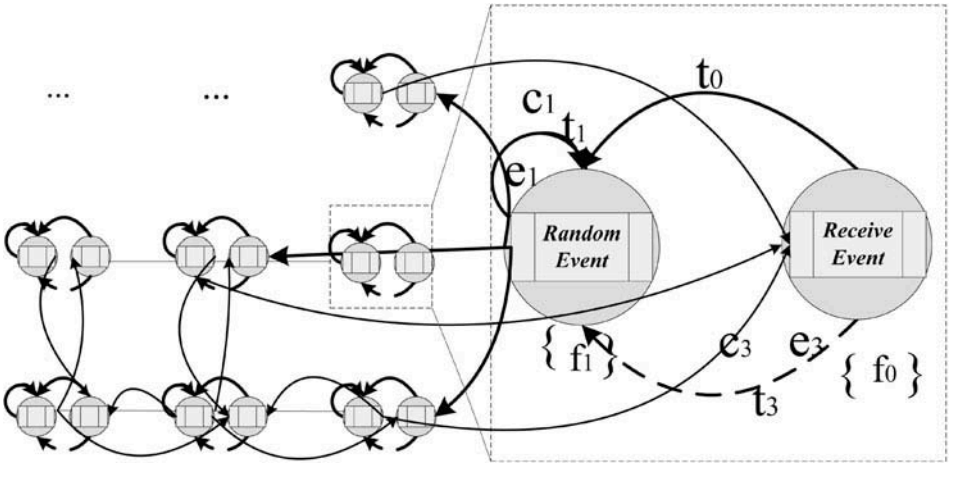

Figure 7. The parameter-free EG-based model for LVSystem.

The DSM of the LVSystem is transformed into an executable model of YHSUPE using the model transformation framework we proposed. The performance of parallel simulation using Breathing Time Warp (BTW) [59] as time management algorithm was

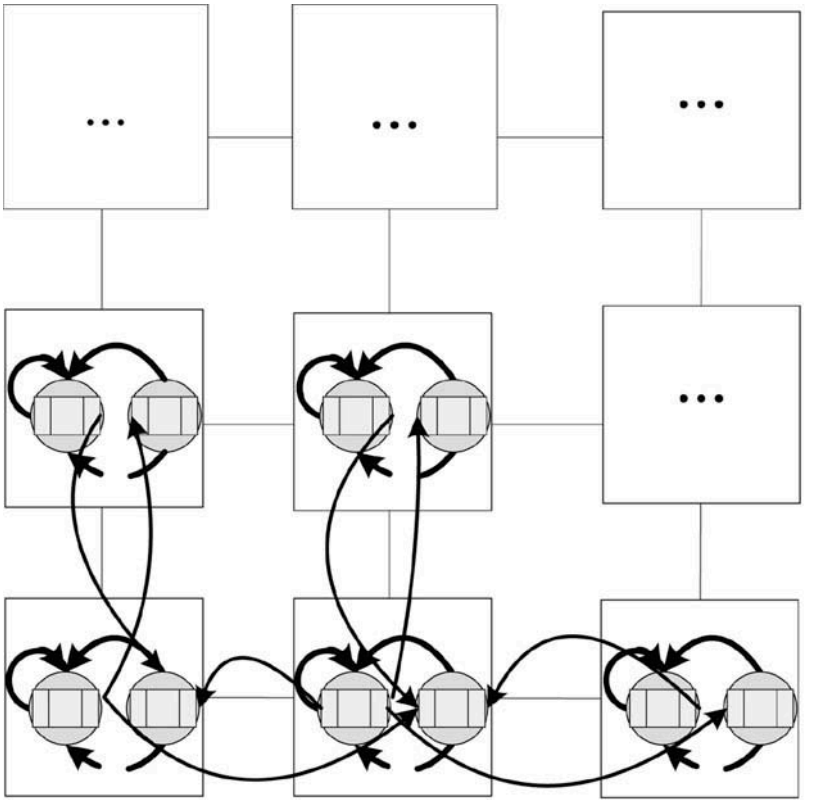

Figure 8. The partition of the parameter-free EG-based model for LVSystem.

investigated. Rollbackable pseudo random number generators are used. The Qheap [60] is used as the event queue. A block style static partition is employed, in which nearby LPs are dispatched to the same logical processor, or if not the same, as nearby as possible.

Figure 9 shows the comparison of speedup on a 100 × 100 grid and a 64 × 64 grid of LVSystem. It also shows the speedup variation according to the number of processors used in the experiments.

The relationship between the speedup and the number of processors deployed is shown. The model size for the blue line is 100 × 100 and for red line is 64 × 64. The speedup grows quickly at first, and then declines slowly as the number of processors involved increases from 1 to 18. The inflection points for 100 × 100 model is 8 processors, while for 64 × 64 model is 4 processors. The speedup growth comes from computing power utilization, while the decline is due to the model size not matching the scale of physical platform. Performance penalty is more severe for models of smaller size.

6. Conclusion and outlook

In this work, we have extended the classical event graphs to formalize the LP-based model of PDES. PEG formalism is a well-defined interpretation of the elements in the LP paradigm and serves as an unambiguous and platform-independent description of the LP-based model. The SOSs rigorously represent the timed behaviour of the LP-based model.

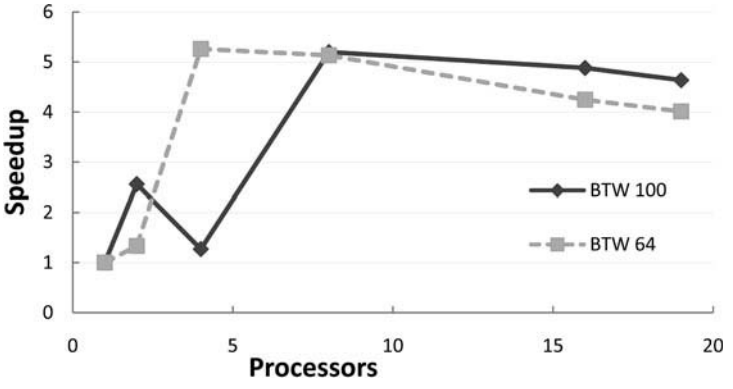

Figure 9. Simulation results: speedup of BTW under different hardware scales and model sizes.

Though the SOS is sequential, it generates exactly the same result as parallel executions do with any time management algorithms. In our case, Wallclock time execution and its equivalence relationship based upon LCC is introduced. In the model transformation framework for PDES, the models are transformed using PEG formalism, from a DSL into a platform-specific API, while four personnel role types are distinguished. Based on this framework, a DSL for the PDES of LVSystem is presented, and preliminary parallel simulation results using YHSUPE are obtained. This case study shows that the PEG-based framework cannot only effectively transform a DSL into the LP paradigm, but also result in efficient parallel simulation on a specific platform. In short, PEG formally specifies the characteristics of event scheduling and state partition in the LP paradigm and thus provides modellers with a mean to formally analyse an LP-based model's behaviour and a cross-platform model transformation.

There are several directions for further work. First, PEG formalism is not designed as a high-level specification of DESs and does not allow create/delete vertexes/edges at run-time. This would make mapping high-level formalisms (such as DEVS [4]) with an indeterministic and dynamic structure to PEG an important direction for the future. Second, a more specialized language that only allows coding valid models of PEG formalism, and incorporates automatic model partitioning for general EG-based model should be developed. Third, the SOS of the PEG presented in this paper formed a basis for formal verification. By transforming the PEG-based model into a timed automata-based model, mainstream model checking tools such as DiVinE (Brno, Moravia, Czech Republic) [14] can be used to study the properties of LP-based model as a necessary supplement to the simulation approach. Fourth, the current sequential operation semantics of the PEG yields exactly the same results as any parallel execution adhering to the LCC does, and serves as a basis for further study of formalizing parallel execution. To develop an abstract machine with concurrent and speculative operation semantics for PEG, such as introducing Tapus's semantics [44], is of great interest.

Acknowledgements

The authors would like to show their gratitude to Prof. Lee W. Schruben, Prof. Adelinde M. Uhrmacher, Dr Jan Himmelspach, and Dr Roland Ewald for their help throughout the development

of this work, and to Stephen Laudig J.D. for his language editing assistance. Authors are greatly thankful to the reviewers for their valuable comments to improve this paper.

Funding

This work is partly supported by the China Scholarship Council [grant number 2007U39612]; the National Natural Science Foundation of China (NS-FC) [grant number 61271252], [grant number 61003075].

References

[1] J. Liu, J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh, and J.C. Smith, *Parallel Discrete-Event Simulation*, John Wiley & Sons, Inc., Hoboken, NJ, 2011.
[2] R.M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley and Sons, Hoboken, NJ, 2000.
[3] J. Misra, *Distributed discrete-event simulation*, ACM Comput. Surv. 18 (1986), pp. 39–65.
[4] B.P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation*, 2nd ed., Vol. 2, Academic Press, Waltham, MA, 2000.
[5] R. Alur, *Timed Automata*, Computer Aided Verification, Springer-Verlag, New York, 1999, pp. 688.
[6] C.G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed., Springer-Verlag, New York, 2008.
[7] D. Harel, *Statecharts: a visual formalism for complex systems*, Sci. Comput. Programming. 8 (1987), pp. 231–274.
[8] R.J. Paul, *Activity cycle diagrams and the three-phase method*, in *Proceedings of the 25th Conference on Winter Simulation*, WSC '93, Los Angeles, CA, ACM, New York, 1993, pp. 123–131.
[9] J.M. Wing, *FAQ on PI-calculus*, Microsoft Internal Memo, 2002.
[10] W. Xia, Y. Yao, and X. Mu, *An extended event graph-based modelling method for parallel and distributed discrete-event simulation*, Math. Comput. Model. Dyn. Syst. 18 (2012), pp. 287–306. doi:10.1080/13873954.2012.655697.
[11] E.L. Savage, L.W. Schruben, and E. Yücesan, *On the generality of event-graph models*, INFORMS J. Comput. 17 (2005), pp. 3–9. doi:10.1287/ijoc.1030.0053.
[12] L. Schruben, *Simulation modeling with event graphs*, Commun. ACM. 26 (1983), pp. 957–963.
[13] A. Buss, *Component Based Simulation Modeling with Simkit*, Vol. 1, San Diego, CA, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 243–249.
[14] J. Barnat, L. Brim, M. Češka, and P. Rockai, *DiVinE: parallel distributed model checker (Tool paper)*, in *Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology (HiBi/PDMC 2010)*, Twente, IEEE Computer Society, Washington, DC, 2010, pp. 4–7.
[15] P. Šimeček, DIVINE – distributed verification environment; Master's thesis, Faculty of Informatics, Masaryk University, Czech Republic, 2006.
[16] R. Eshuis, *Reconciling statechart semantics*, Sci. Comput. Programming. 74 (2009), pp. 65–99. doi:10.1016/j.scico.2008.09.001.
[17] E. Posse, *Modelling and Simulation of Dynamic Structure Discrete-Event Systems*, School of Computer Science McGill University, Montreal, QC, 2008.
[18] R. Curry, C. Kiddle, R. Simmonds, and B. Unger, *Sequential performance of asynchronous conservative PDES algorithms*, in *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, IEEE Computer Society, Washington, DC, 2005, pp. 217–226.
[19] P. Peschlow, M. Geuer, and P. Martini, *Logical process based sequential simulation cloning*, in *Simulation Symposium, ANSS 2008, 41st Annual (2008)*, IEEE Computer Society, Washington, DC, 2008, pp. 237–244.
[20] K.M. Chandy and J. Misra, *Distributed Simulation: A Case Study in Design and Verification of Distributed Programs*, in *Software Engineering, IEEE Transactions on SE-5*, IEEE Computer Society, Washington, DC, 1979, pp. 440–452.

[21] E. Yücesan and L. Schruben, *Structural and behavioral equivalence of simulation models*, ACM Trans, Model. Comput. Simul. 2 (1992), pp. 82–103.
[22] L.W. Schruben, *Graphical Simulation Modeling and Analysis: Using SIGMA for Windows*, 1st Course Technology Press, Boston, MA, 1995.
[23] L.W. Schruben, T.M. Roeder, W.K. Chan, P. Hyden, and M. Freimer, *Advanced event scheduling methodology*, in *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*, WSC '03, New Orleans, Louisiana Winter Simulation Conference, IEEE Computer Society, Washington, DC, 2003, pp. 159–165.
[24] A.H. Buss and P.J. Sanchez, *Building complex models with LEGOs (Listener Event Graph Objects)*, Winter Simulation Conference, Vol. 1, Los Alamitos, CA, IEEE Computer Society, Washington, DC, 2002, pp. 732–737.
[25] J. de Lara, *Distributed event graphs: formalizing component-based modelling and simulation*, Electron. Notes. Theor. Comput. Sci. 127 (2005), pp. 145–162.
[26] D.O. Rich and R.E. Michelsen, *An assessment of the ModSim/TWOS parallel simulation environment*, in *WSC '91: Proceedings of the 23rd Conference on Winter Simulation*, Phoenix, AZ, IEEE Computer Society, Washington, DC, 1991, pp. 509–518.
[27] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, *GTW: a time warp system for shared memory multiprocessors*, in *Simulation Conference Proceedings*, *Winter*, Lake Buena Vista, FL, IEEE Computer Society, Washington, DC, 1994, pp. 1332–1339.
[28] J.S. Steinman, *SPEEDES: synchronous parallel environment for emulation and discrete event simulation*, in *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, January, IEEE Computer Society, Washington, DC, 1991, pp. 95–101.
[29] T.L. Wilmarth, *POSE: Scalable General-purpose Parallel Discrete Event Simulation*, University of Illinois, Champaign, IL, 2005.
[30] C.D. Carothers, D. Bauer, and S. Pearce, *ROSS: a high-performance, low memory, modular time warp system*, Workshop on Parallel and Distributed Simulation, Los Alamitos, CA, IEEE Computer Society, Washington, DC, 2000, pp. 53–60.
[31] K. Perumalla, *μsik: a micro-kernel for parallel/distributed simulation systems*, in *Workshop on Parallel and Distributed Simulation*, June, IEEE, Monterey, CA, 2005, pp. 59–68.
[32] R.L. Bagrodia and W.T. Liao, *Maisie: a language for the design of efficient discrete-event simulations*, IEEE Trans. Software. Eng. 20 (1994), pp. 225–238. doi:10.1109/32.277572.
[33] R.G. Sargent, *Some recent advances in the process world view*, in *Proceedings of the 36th Conference on Winter Simulation*, WSC '04, Washington, DC, Winter Simulation Conference, IEEE Computer Society, Washington, DC, 2004, pp. 293–299.
[34] G. Liu, Y. Yao, and B. Liu, *VISICOM: a component-based parallel discrete event modeling framework*, International Conference on Advances in System Simulation, (2010), pp. 158–163.
[35] B. Hou, Y. Yao, B. Wang, and D. Liao, *Modeling and simulation of large-scale social networks using parallel discrete event simulation*, Simulation. 89 (2013), pp. 1173–1183.
[36] K.S. Perumalla and R.M. Fujimoto, *Interactive parallel simulations with the Jane framework*, Future. Gener. Comput. Syst. 17 (2001), pp. 525–537.
[37] R.L. Bagrodia, *Parallel languages for discrete-event simulation models*, IEEE Computational. Sci. Eng. 5 (1998), pp. 27–38. doi:10.1109/99.683737.
[38] G.A. Wainer and N. Giambiasi, *N-dimensional cell-DEVS models*, Discrete Event Dynamic Systems. 12 (2002), pp. 135–157. doi:10.1023/A:1014536803451.
[39] Q. Liu, *Algorithms for Parallel Simulation of Large-Scale DEVS and Cell-DEVS Models*, Carleton University, Ottawa, ON, 2010.
[40] M. Hybinette, E. Kraemer, Y. Xiong, G. Matthews, and J. Ahmed, *SASSY: a design for a scalable agent-based simulation system using a distributed discrete event infrastructure*, in *Proceedings of the 38th Conference on Winter Simulation*, WSC '06, Monterey, California Winter Simulation Conference, IEEE Computer Society, Washington, DC, 2006, pp. 926–933.
[41] D. Chen, G.K. Theodoropoulos, S.J. Turner, W. Cai, R. Minson, and Y. Zhang, *Large scale agent-based simulation on the grid*, Future. Gener. Comput. Syst. 24 (2008), pp. 658–671.
[42] T. Oguara, D. Chen, G. Theodoropoulos, B. Logan, and M. Lees, *An adaptive load management mechanism for distributed simulation of multi-agent systems*, Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications. 1 (2005), pp. 179–186. doi:10.1109/DISTRA.2005.9.
[43] J. Nutaro and H. Sarjoughian, *Design of distributed* simulation *environments: a unified system-theoretic and logical processes approach*, Simulation. 80 (2004), pp. 577–589.

[44] C. Tapus, *Distributed Speculations: Providing Fault-Tolerance and Improving Performance*, California Institute of Technology, Pasadena, CA, 2006.
[45] F. Wieland, *The threshold of event simultaneity*, in *Proceedings of the eleventh workshop on Parallel and distributed simulation*, PADS '97, Lockenhaus, Austria IEEE Computer Society, Washington, DC, 1997, pp. 56–59.
[46] R.G. Ingalls, D.J. Morrice, E. Yucesan, and A.B. Whinston, *Execution conditions: a formalization of event cancellation in simulation graphs*, INFORMS J. Comput. 15 (2003), pp. 397–411.
[47] ECMA International, *ECMA-262: ECMAScript Language Specification*, 5.1 ed., Geneva, 2011.
[48] A.W. Brown, S. Iyengar, and S. Johnston, *A rational approach to model-driven development*, IBM Syst. J. 45 (2006), pp. 463–480. doi:10.1147/sj.453.0463.
[49] E.H. Page, *Beyond speedup: PADS, the HLA and Web-based simulation*, in *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, PADS '99, Atlanta, GA, IEEE Computer Society, Washington, DC, 1999, pp. 2–9.
[50] E.H. Page, *Simulation Modeling Methodology: Principles and Etiology of Decision Support*, Department of Computer Science, Virginia Tech, Blacksburg, 1994.
[51] M.L. Fredman and D.E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. Syst. Sci. 48 (1994), pp. 533–551. doi:10.1016/S0022-0000(05)80064-9.
[52] B. Wang, Y. Yao, Y. Zhao, B. Hou, and S. Peng, *Experimental analysis of optimistic synchronization algorithms for parallel simulation of reaction-diffusion systems*, in *Proceedings of the 2009 International Workshop on High Performance Computational Systems Biology*, Vol. 0 of *HIBI '09*, Oct., IEEE Computer Society, Washington, DC, 2009, pp. 91–100.
[53] B. Wang, B. Hou, F. Xing, and Y. Yao, *Abstract next subvolume method: a logical process-based approach for spatial stochastic simulation of chemical reactions*, Comput. Biol. Chem. 35 (2011), pp. 193–198. doi:10.1016/j.compbiolchem.2011.05.001.
[54] R.B. Schinazi, *Predator-prey and host-parasite spatial stochastic models*, The Annals of Applied Probability. 7 (1997), pp. 1–9. doi:10.1214/aoap/1034625250.
[55] J.O. Loffler, W. Rand, and U. Wilensky, *A spatial model of the red queen effect*, in *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, London, ACM, New York, 2007, pp. 490–491.
[56] D.T. Gillespie, *Exact stochastic simulation of coupled chemical reactions*, J. Phys. Chem. 81 (1977), pp. 2340–2361. doi:10.1021/j100540a008.
[57] J. Elf and M. Ehrenberg, *Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases*, IEE Proc. Syst. Biology. 1 (2004), pp. 230–236. doi:10.1049/sb:20045021.
[58] T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*, Pragmatic Bookshelf, Dallas, TX, 2007.
[59] J.S. Steinman, *Breathing time warp*, in *PADS '93: Proceedings of the Seventh Workshop on Parallel and Distributed Simulation*, San Diego, CA, ACM, New York, 1993, pp. 109–118.
[60] J.S. Steinman, C.A. Lee, L.F. Wilson, and D.M. Nicol, *Global virtual time and distributed synchronization*, SIGSIM Simul. Dig. 25 (1995), pp. 139–148.