Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Activity-based simulation using DEVS: increasing performance by an activity model in parallel DEVS simulation[*]

Bin CHEN[†1], Lao-bing ZHANG[1], Xiao-cheng LIU[1], Hans VANGHELUWE[2,3]
(*[1]College of Information System and Management, National University of Defense Technology, Changsha 410073, China*)
(*[2]Department of Mathematics and Computer Science, University of Antwerp, Antwerp 2020, Belgium*)
(*[3]School of Computer Science, McGill University, Montréal H3A 2A7, Canada*)
[†]E-mail: nudtcb9372@gmail.com

Received May 9, 2013; Revision accepted Nov. 15, 2013; Crosschecked Dec. 16, 2013

Abstract: Improving simulation performance using activity tracking has attracted attention in the modeling field in recent years. The reference to activity has been successfully used to predict and promote the simulation performance. Tracking activity, however, uses only the inherent performance information contained in the models. To extend activity prediction in modeling, we propose the activity enhanced modeling with an activity meta-model at the meta-level. The meta-model provides a set of interfaces to model activity in a specific domain. The activity model transformation in subsequence is devised to deal with the simulation difference due to the heterogeneous activity model. Finally, the resource-aware simulation framework is implemented to integrate the activity models in activity-based simulation. The case study shows the improvement brought on by activity-based simulation using discrete event system specification (DEVS).

Key words: Activity tracking, Activity enhanced modeling, Discrete event system specification (DEVS), Resource-aware simulation framework
doi:10.1631/jzus.C1300121 **Document code:** A **CLC number:** TP312

1 Introduction

Much work has been done to improve the performance of simulation, especially parallel simulation. Simulation performance is predicted to make the partition of models more reasonable. The researches are divided roughly into two types: dynamic load balancing and model-based performance predicting (Balsamo *et al.*, 2004). The former type focuses on the performance data at runtime, which is observed and recorded to do the analysis using machine learning algorithms. The algorithms such as Q-learning and simulated annealing are applied to find the performance trend. The latter type builds performance models with the help of mathematical formalisms such as queuing networks (Petriu and Shen, 2002) and Petri nets. In some special cases, biological models like neural networks are imported. The performance model is built during the training process. Many samples are needed to attain the precise parameters that cover all the possibilities in simulation. In summary, the traditional algorithms are all performance centered. They try to predict the performance at the simulation level, without touching anything in the modeling aspect. Recently, the study on activity has become a novel field in performance optimization. Different from the traditional methods, activity is tightly bound up with models. With the help of an activity model, the partition of parallel simulation is adjusted at runtime to achieve higher performance. Activity tracking (Muzy and Zeigler, 2008) has been shown to contribute greatly in forest fire simulation (Hu and Ntaimo, 2008), pipeline of product transport (Shibata *et al.*, 2012), and pedestrian crowd

[*] Project supported by the National Natural Science Foundation of China (Nos. 71303252 and 91024030)

simulation (Qiu and Hu, 2013). The spatial activity information from tracking is used to reallocate resources. The performance is improved by the more reasonable resource allocation. Furthermore, weighted activity proposed by Hu and Zeigler (2013) connects the information from models and the energy consumption. However, both activity tracking and the quantization of activity discussed above are still limited in the discrete event system specification (DEVS) (Zeigler *et al.*, 2000) field. The advantages of domain specific modeling (DSM) are not applied. The activity-based techniques still meet three problems: (1) How to quantify activity of models? (2) How to apply activity-based techniques in specific domains? (3) How to construct a simulation framework for activity-based techniques in domain specific modeling?

We try to solve these problems in the paper. Resource usages such as CPU and memory occupations are used to quantify activity. So, the prediction of activity can be implemented by calculating the computation and memory occupation of the transitions in the near future. Activity enhanced modeling is proposed to support activity modeling in domain specific modeling. Model transformation of activity solves the model difference between specific domains and DEVS. The resource-aware simulation framework is designed to implement activity-based simulation in which a parallel DEVS simulator is modified to meet the requirements from activity prediction and load balancing.

2 Activity

2.1 What is activity?

It is known that activity should be taken into account in the modeling process. The sub-components of models are usually running at different activity levels during simulation. The activity is distributed in both the temporal and spatial dimensions. So, we give the definition of activity here.

Activity is the rate of change of the parameter in the temporal and spatial dimensions. It is the notion of locality in space and time.

Forest fire (Hu *et al.*, 2005; Hu and Ntaimo, 2006) and missile launch are seen as typical examples to describe activity in both spatial and temporal dimensions. Fig. 1 shows the activity intensity of forest fire. Similarly, Fig. 5 (see p.17) gives an example to describe the activity trace with time elapsing. The activity of the model usually does not stay at the same level during the simulation. The different activity levels indicate the different resource needs. The initial static resource allocation cannot satisfy the resource needs for the whole simulation. The resource disequilibrium deteriorates the simulation performance. So, it is necessary to reallocate the resource under the instruction of activity.

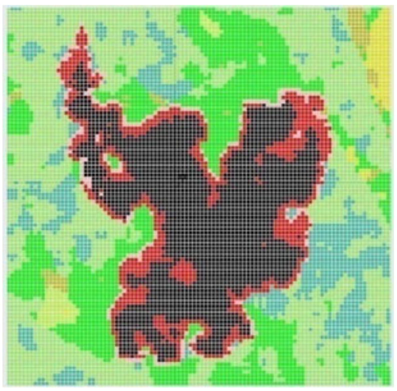

Fig. 1 Activity region of forest fire (Hu and Ntaimo, 2008)
The black and red spots represent the passive and active activity regions, respectively. References to color refer to the online version of this figure

2.2 Activity formalization

The activity discussed here is the abstraction of the physical activity such as the aircraft taking off, missile launch, and platoon moving. The physical activity describes the active operations or status of real systems. Accordingly, we define the activity for models that are the abstraction of systems in the real world. The formalization of activity in Jammalamadaka (2003), Zeigler *et al.* (2004), and Muzy *et al.* (2010; 2011) is given below:

$$A(t)=\int_0^T\left|\frac{\partial \Phi(t)}{\partial t}\right|\partial t, \tag{1}$$

where $\Phi(t)$ represents the mathematical abstraction of the real system that is either the continuous system or discrete system (discrete time/event). In the computer world, models are constructed based on the mathematical description. All the continuous models are discretized to be discrete time or discrete event models. It is because models can be simulated only in a discrete manner in a computer. Therefore, the mapping from a mathematical model to a simulation model is an approximation process. Likewise, the continuous activity model $A(t)$ is discretized to meet the simulation requirements. Actually, the activity is measured by the resource usage in the simulation. So, the activity model is also approximated by the activity of resource usage. According to the definition of activity, the activity model means the intensity of evolution during the model's life cycle. The evolution consists of a large number of transitions in the model. So, the transition frequency well reflects the quantum of activity of resource usage. The activity formalization by transition in the simulation view is listed below:

$$A_H^R = \int_H^R v_H(t)\mathrm{d}t, \quad (2)$$

where $v_H(t)$ is the frequency of transition i at t from physical time H to R. From the aspect of implementation, the integration is realized by the summary of the quantized function. Many algorithms have been devised to make the quantized results consistent with the original continuous models. In Muzy *et al.* (2008), quantization works well by quantizing the state variables of a continuous system in equal quanta. As shown in Fig. 2, four kinds of transitions (from 1 to 4) are triggered from physical time H to R. The frequencies of transitions are discrete in the computer world. So, $v_H(i)$ is represented by a piecewise step curve along the physical time t. Referring to the weighted activity proposed by Hu and Zeigler (2013), we can also formalize activity in a quantized manner:

$$A_H^R = \sum_{i=1}^{n} v_H(i)\Delta t_{\text{phys}}(i), \quad (3)$$

where $v_H(i)$ is the frequency of transition i at physical time interval $\Delta t_{\text{phys}}(i)$. In the view of implementation, however, the transitions of models bring the computational intensity, which is reflected by CPU and memory occupations in the computer. So, the resource usage such as CPU and memory occupations is also considered in the quantification of $v_H(i)$. This will be discussed later in our work. n is the total amount of the different kinds of transitions. $\Delta t_{\text{phys}}(i)$ is the physical time cost of transition i from physical time H to R.

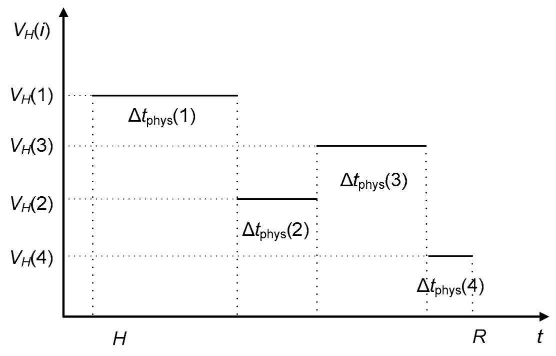

Fig. 2 Approximation of $v_H(i)$

Corresponding with Fig. 2, Fig. 3 gives a simple example of a car moving in a 5×3 map. The moving from the start point to the end point is composed of four transitions ($v_H(1)$, $v_H(2)$, $v_H(3)$, and $v_H(4)$) represented in different colors. The frequency of transitions is computed by the amount of the atomic transition Move from one grid to another. So, the values of $v_H(1)$, $v_H(2)$, $v_H(3)$, and $v_H(4)$ are 4, 2, 3, and 1, respectively. The corresponding $\Delta t_{\text{phys}}(i)$'s are shown in Fig. 2. Obviously, the larger the transition frequency, the longer the physical time will be. The physical time cost is decided by the computational intensity of transition. Note that the computational intensity of atomic transition Move is an important parameter in the quantification of the activity model in the moving car example.

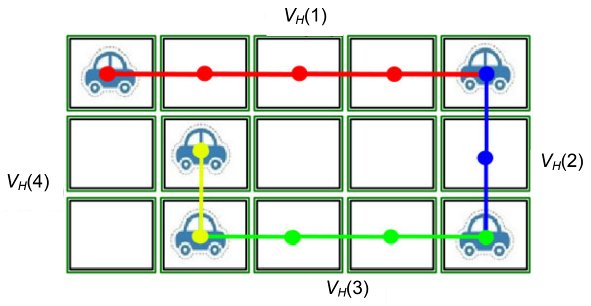

Fig. 3 A simple moving car example of $v_H(i)$

Fig. 4 illustrates the whole story of activity formalization in a global view. Parallel with the modeling of real systems, the idea of how to model activity is shown in the bottom of the figure. Based on study of the real systems such as aircraft, missile, and platoon, the modelers can also find physical activities. The behaviors like aircraft taking off, missile launch, and platoon moving are considered to be the physical

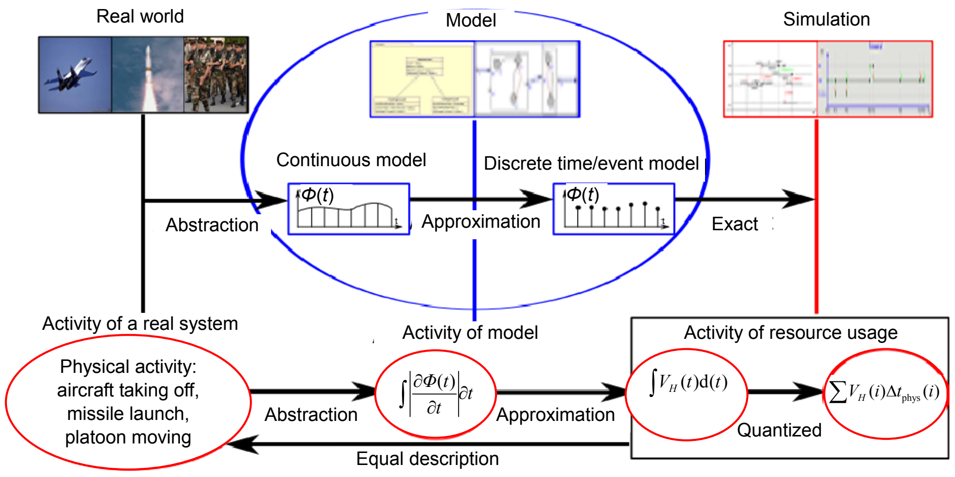
Real world
Model
Simulation
Continuous model
Discrete time/event model
Φ(t)
Φ(t)
Abstraction
Approximation
Exact
Activity of a real system
Activity of model
Activity of resource usage
Physical activity: aircraft taking off, missile launch, platoon moving
$\int\left|\frac{\partial\Phi(t)}{\partial t}\right|\partial t$
$\int V_H(t)\mathrm{d}(t)$
$\sum V_H(i)\Delta t_{\text{phys}}(i)$
Abstraction
Approximation
Quantized
Equal description

Fig. 4 Activity formalization from the real world to simulation

activities of real systems. As discussed before, the physical activities are modeled in Eq. (1) in the modeling view. Similar to the approximation from a continuous model to a discrete event or discrete time model, the activity model is defined in Eq. (2) by approximation. Because the performance is measured by resource usage in the computer, the activity model is finally formalized in Eq. (3) with $v_H(i)$ represented by resource usage. The quantified definition is considered to be equal to the abstraction of physical activities in the simulation view.

2.3 Activity in the compositional model

The activity model defined before considers only the activity in the temporal dimension at the atomic level. It does not take account of the compositional model. So, we propose the activity set to define the integral activity for the compositional model. The activity set is integration of activities from active sub-components in the physical time span (from H to R). The definition of the activity set is given as

$$\begin{aligned}&\mathrm{AS}_H^R=\{A_{c_j}^{\mathrm{AC}}\mid(v_H^{c_j}(t))>0\},\\&c_j\in C_s, s\in\mathrm{AC}, H<t<R,\end{aligned}\tag{4}$$

where AC is the subset of the compositional model's state space. It represents the active coordinate collection for sub-components. s is a collection of the states of sub-components, and $s\in$AC. C_s is the set of active sub-components with state s. c_j is an active sub-component belonging to C_s. $v_H^{c_j}(t)$ represents the transition intensities (frequencies) of c_j in the state belonging to s. AS_H^R gives a generalized activity representation relying on AC. AC is a general abstraction for active components. It contains the relationship between activity and computational intensity of transition. Furthermore, the definition illustrates a principle for the modeler to predict the transition intensity in future.

Additionally, in the view of visual modeling, the geographically spatially connected atomic components constitute the more complex model. The geographical locations are also information of the model outside the components. Consequently, the activity in the spatial dimension is defined corresponding to these spatial connection emphasized models. Region is a smaller group of atomic components with similar activities. The activity region defines how these regions evolve with time. The definition is given as

$$\begin{aligned}&\mathrm{AR}_H^R=\{A_{c_{(x,y)}}^{\mathrm{AC}}\mid(v_H^{c_{(x,y)}}(t))>0\},\\&c_{(x,y)}\in C_s, s\in\mathrm{AC}, H<t<R,\end{aligned}\tag{5}$$

where $c_{(x,y)}$ is a sub-component at location (x, y). The activity region can be refined with spatial information from the atomic activity model in the temporal dimension.

Take the missile launch in Fig. 5 as an example. The activity evolves with time elapsing. The activity stays at zero before the launch time and after the exploding time, but jumps to a high level at launch time or exploding time. Obviously, the missile activities at different levels (intensities) are located at different locations. So, the X-axis is a complement to the description of missile activity that evolves with distance to launch position. The bottom of Fig. 5 shows the refinement by distance information. The activity-time-distance axis is used to show the activity evolution along both time and distance. Actually, the activity spots with time and distance are the activity regions. In the opposite way, the concise and portable activity expression is more appropriate in some specific cases. So, the abstraction named collapse from the activity region to temporal atomic activity is necessary. Collapse can be seen as the projection from the activity region to temporal dimension. As shown in the figure, the activity region is projected to the activity-time plane with the loss of distance information.

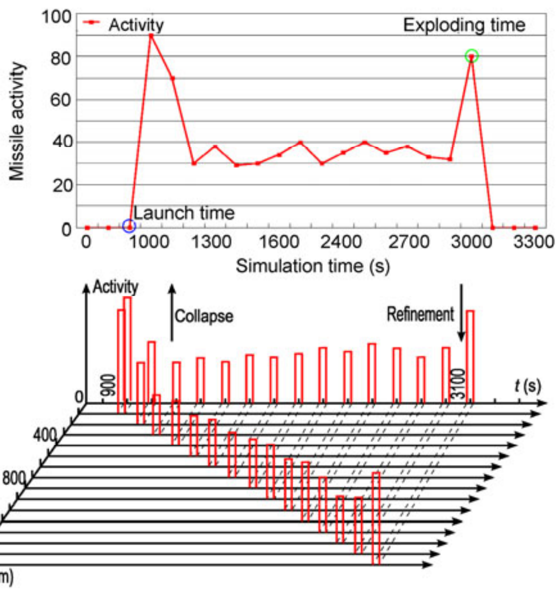

Fig. 5 Temporal activity of missile launch

3 Activity enhanced modeling

Based on the activity formalization discussed before, we build the model to predict the activity in the temporal or spatial dimension. Instructed by the activity prediction, the computational resource is reallocated to improve the simulation performance. It is the principle of activity enhanced modeling. The activity models come from the frequency and intensity of transition associated tightly to models. So, it is necessary to build the frequency and intensity of transition together with the model itself. Domain specific modeling (DSM) minimizes accidental complexity of modeling. The models in the domain are smaller, more efficient, and easier to understand. Likewise, the activity models built by modelers are attached in the domains. According to the theory of the meta-model, it is a good way to build the activity meta-model to extract the common features in the activity. Then the activity meta-model can be merged into the domain meta-model. Therefore, the activity combined meta-model can model activity for the models in the domain. Actually, the activity meta-model provides the interfaces for modelers to implement the methods to do activity prediction.

3.1 Activity meta-model

Considering the hierarchical cases in the domain, the activity meta-model consists of two super classes: atomic activity (AA) and model activity (MA). AA is the atomic component that cannot be decomposed again. It supports only temporal activity prediction for the atomic component. In contrast, MA predicts the activity of the whole model in an entire manner. The hierarchy, structure, and spatial information is considered in MA. MA receives the sub-activities from AA models by association reporting, and then synthesizes the sub-activities with the structural information. The activity of the whole model is finally presented by MA. The activity meta-model packages represented in Unified Modeling Language (UML) are shown in the left of Fig. 6.

AA is designed based on Eq. (3), which defines the activity in two aspects: frequency with intensity of transition v_H and physical time (Δt_{phys}) consumed by each type of transition. AA concerns the types of transitions and the transition frequency with intensity. So, we define TransitionType to enumerate the types of transitions. The AtomicTransitionTypes in AA are typed by TransitionType. This attribute lists all the types of transitions in the sub-component. PredictAtomicActivity is the virtual interface for modelers to give the function to predict transition frequency and intensity. The results returned by the interface are the transitions that will be triggered in future. LastTransitionType is used to indicate the last transition type in order to match the physical time cost when the simulation is running. The model activity predicts activity in a global perspective. The TransitionTypes contains all the transition types occurring in this model.

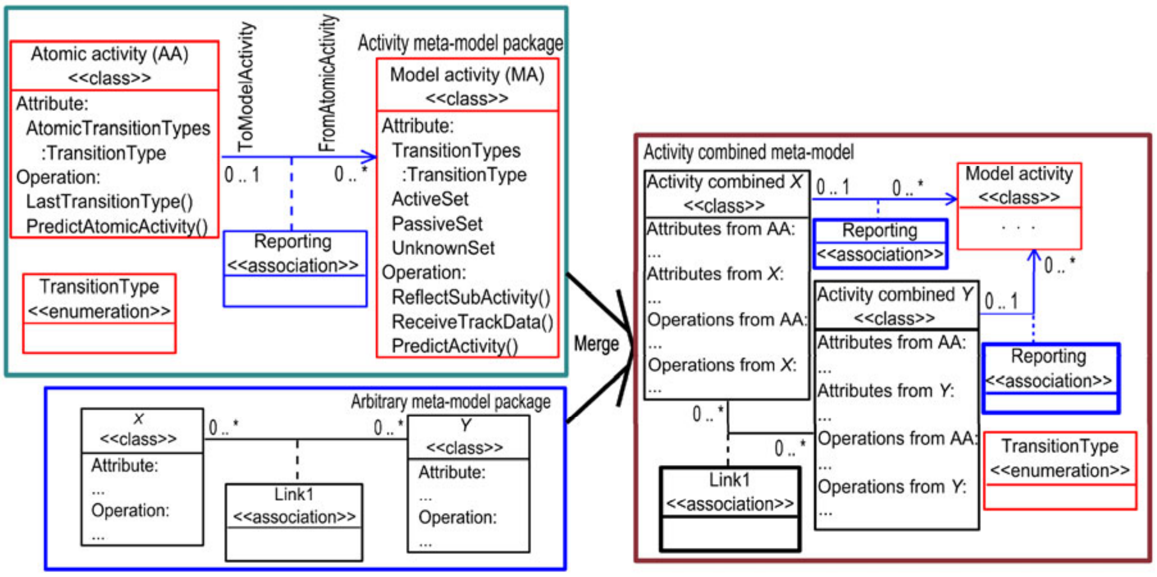

Fig. 6 Integrating activity meta-model into an arbitrary meta-model

The transition types are marked with the sub-component. They represent the different activity frequencies and intensities. Compared to AA, MA emphasizes the ActiveSet that is the projection of the active region in the spatially connected models. ActiveSet holds the model's current active part to indicate the resource need. On the contrary, PassiveSet means the models that will not be active again. Meanwhile, UnknownSet consists of the uncertainty in activity prediction. ReflectSubActivity is used to receive activity from sub-components. PredictActivity calculates the activity of the entire model, including spatial and hierarchical information. LastTransitionAtomic and LastTransitionType output the information of the last active transitions. They are used to construct the mapping table (MT) between resource usage and transition, which will be discussed in the next section together with LastTransitionType in AA. Atomic activity is linked to model activity by the association reporting. As a result, all the AA models report their activities to the MA model after each transition.

3.2 Activity combined meta-model

We combine the activity meta-model and arbitrary meta-model with meta-model merging techniques (Emerson and Sztipanovits, 2006; Lagerström *et al.*, 2008). The two packages are merged to be the activity combined meta-model, as shown in the right of Fig. 6. The merging operations are list below:

1. Atomic activity (AA) is merged into each atomic class in an arbitrary meta-model. The attributes and operations are directly copied to the new atomic class. AA does not exist in the activity combined meta-model any more.

2. Model activity (MA) and TransitionType are kept in the activity combined meta-model.

3. The reporting associations between the AA model and MA model are resumed. All the atomic classes within activity parts are linked to MA by reporting associations.

4. The associations in the arbitrary meta-model are still kept in the activity combined meta-model.

We execute the operations as shown in Fig. 6. MA and TransitionType are copied to the activity combined meta-model. The activity attributes from AA are embedded into atomic classes X and Y. The association reporting is linked to each atomic class to connect MA. The association Link1 between atomic classes X and Y is retained.

3.3 Activity combined model in general purpose simulation formalism (GPSF)

The activity combined meta-model is still an application of domain specific modeling (DSM). DSM reduces modeling complexity but brings out the difference in simulation (Kelly and Tolvanen, 2008). It is impossible to establish a universal simulator to run all kinds of models in the domain. Compared to solving the differences in the simulator, model transformation is more immediate and much work has been done (D'Abreu and Wainer, 2005). Fig. 7 shows the idea of how to do the modeling with activity from the domain to efficient simulation. The resource can be aware of simulators under the instruction of the activity model. As a result, we have to find a general-purpose simulation formalism with a highly efficient simulator. Thanks to the precise definition, modularity, and hierarchy, the discrete event specification system (DEVS) is chosen to be the general-purpose simulation formalism. The introduction of DEVS was detailed in Concepcion and Zeigler (1988) and Vangheluwe (2000). This portable formalism can be expanded to more than a discrete event field. Much work has been done on the modeling of continuous and discrete time systems using DEVS. Thus, the transformation from domain to DEVS is inherently convenient compared to other formalisms. Obviously, the model transformation from an activity combined meta-model to DEVS is composed of two parts: transformation from the meta-model in a specific domain to DEVS and transformation from the activity meta-model in a specific domain to DEVS. The former has been discussed in Czarnecki and Helsen (2003), Sendall and Kozaczynski (2003), and Syriani and Vangheluwe (2007). We discuss the activity model transformation in detail in the next subsection.

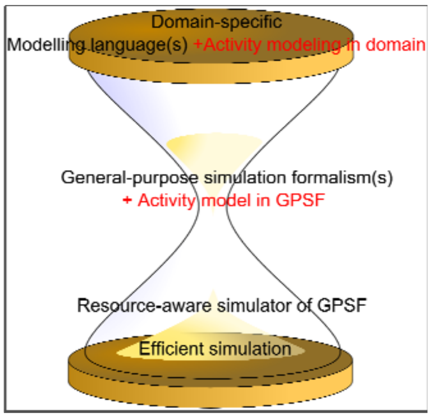
Domain-specific
Modelling language(s) +Activity modeling in domain
General-purpose simulation formalism(s)
+ Activity model in GPSF
Resource-aware simulator of GPSF
Efficient simulation

Fig. 7 Activity modeling: from the modeling domain to efficient simulation
The wide top of the sand glass indicates much activity combined with modeling formalisms in the domains, the narrow middle of the glass means that the multi-formalisms have to be transformed to unified GPSF, and the wide glass bottom represents the resource-aware simulator which is implemented with many techniques to improve efficiency

3.4 Activity model transformation

As discussed in Section 3.2, the activity meta-model consists of AA and MA. Therefore, activity model transformation is composed of AA transformation and MA transformation. Note that AA has been merged into the atomic class of activity combined meta-model. So, the source part of AA transformation is composed of the activity attributes (such as AtomicTransitionTypes) and operations (such as OutputAtomicActivity()) while the target part consists of the corresponding attributes and operations imbedded inside the atomic DEVS models. In the meantime, MA transformation obtains a default activity DEVS model called the activity predictor transformed from the MA in a specific domain. According to the theory of model transformation (Czarnecki and Helsen, 2003; Vangheluwe and de Lara, 2004), the transformation rules for AA and MA are listed, respectively. The AA transformation rules are as follows:

1. Find the attributes such as AtomicTransitionTypes belonging to AA in each class of activity combined meta-model.
2. Find the corresponding target atomic DEVS models transformed from the original class in the meta-model in a specific domain.
3. Add these attributes to the states of corresponding atomic DEVS models.
4. Map the operation PredictAtomicActivity() of AA to a sub-function in internal transition in the corresponding atomic DEVS model.
5. Map the operations OutputAtomicActivity() and LastTransitionType() of AA to the sub-functions in the λ function in the corresponding atomic DEVS model.
6. Transform the type TransitionType to the built-in type in the DEVS model.

The rules of MA transformation are listed below:

1. Build an atomic DEVS model, the activity predictor, which is the target model of MA.
2. Copy the attributes in MA to the model state in the activity predictor.
3. Transform the operation PredictActivity() to a sub-function in the internal transition.
4. Embed the operations ReflectSubActivity() and ReceiveTrackData() into the external transition.
5. Integrate LastTransitionType() and OutputActivity() into the λ function.

The reporting associations between the activity model and sub-components are mapped to the activity connections. These connections between activity ports are used to report atomic activities to the activity predictor.

These rules are generated based on source target relationship. The application scope of the rules is limited in the models typed by the activity combined meta-model. Activity model transformation also covers the model related intrinsic activity model. The intrinsic model does not evolve with the changes of states. Take the missile model as an example. The activity before launch time and exploding time is zero, without regard to the model states evolution. The model characteristics of activity are named model related intrinsic activity model; it is also extracted during the transformation. These intrinsic activity prediction algorithms are finally merged together with the attributed related activity sub-functions in the internal transition of the activity predictor.

After activity model transformation, the standard definition of the activity predictor is listed below:

$$\delta_P=\{X_P, S_P, Y_P, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a\},$$

where X_P={Sub-Activity, Query, TrackData}, S_P= ActiveSet∪PassiveSet∪UnknownSet∪AllTransitions-Mappings, Y_P={LastTransitionActivity, Activity-Prediction}, δ_{int}={PredictActivity()} is the internal activity prediction calculation, δ_{ext}={ReflectSub-Activity(), ReceiveTrackData()} is the sub-activities collection or activity prediction calculation using track data, λ={LastTransitionAtomic(), LastTransition-Type(), OutputActivity()} outputs results after activity prediction, and t_a returns the time stamp for the next prediction.

With the support of activity model transformation, the models in the specific domain are transformed to the models within atomic activity and the activity predictor represented in DEVS. Activity-based simulation using DEVS is proposed to simulate these models in order to improve the simulation performance. This will be discussed in detail in the next section.

4 Activity-based simulation using DEVS

The principle of activity-based simulation is described in Fig. 8a. Models, simulation environment, and resource are the basic elements in simulation. The load balance module is added as the management part to find the most reasonable partition dynamically at runtime. The simulation environment is used to simulate both the models within atomic activity and the activity predictor. Therefore, the simulation environment is composed of the original simulator, resource predictor, and tracker. The original simulator simulates the models within atomic activity while the resource predictor simulates the activity predictor. Tracker is a tracking system which tracks the resource usage by transitions of models at runtime.

Note that the implementation of the original simulator supports parallel simulation so that the load balance module is able to reallocate the resource. Resource such as CPU and memory occupations is considered in activity-based simulation because the performance is measured mainly by the indexes.

Based on the principle discussed above, it is necessary to answer the following four questions in order to implement the activity-based simulation:

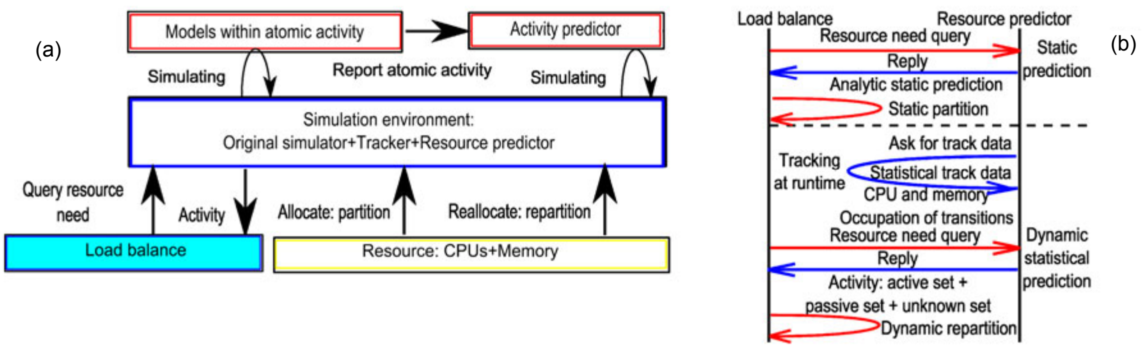

Fig. 8 Activity tracking: (a) principle of activity tracking; (b) query and reply between load balance and resource predictor

1. How to track activity in activity-based simulation?

2. How to quantify resource usage in order to compute activity?

3. How to predict the activity with the support of the activity predictor at runtime?

4. How to construct the resource-aware simulation framework?

4.1 Activity tracking

Activity tracking discussed in Muzy and Zeigler (2008) is used to collect the activity data. The track data of activity includes:

1. Resource usage, i.e., the resource occupied by models which evolves with physical time.

2. Performance, i.e., the model activity with physical time, such as the frequency of model transitions.

3. Resource allocation, i.e., the current resource allocation status.

4. Resource available, i.e., the resource available to be allocated. The query and answer mechanism of activity tracking is illustrated in Fig. 8b. The interactions between the load balance module and the resource predictor are used to obtain the most optimal resource allocation. As mentioned before, the resource predictor is a specific simulator to simulate the activity predictor at runtime. The resource predictor provides two types of prediction: static analytical prediction and statistical prediction. Static analytical prediction comes from the modelers in activity modeling. Take missile launch as an example. The activity before launch and after exploding stays at zero. Statistical prediction is computed with the help of track data from the tracker. Track data indicates the resource usage of transitions during the whole simulation, so the activity can be obtained by the computation of statistical track data of transitions.

At the beginning of simulation, the load balance module queries the static analytical prediction from the resource predictor. The predictor calculates the whole activity trace at the preparation phase. Then the static activity distribution along the simulation is obtained. The load balance module allocates the resource based on the static analytical prediction replied to by the resource predictor.

At runtime, the resource predictor queries the track data from the tracker first. Then it generates the activity prediction including activity trace, active set, passive set, and unknown set. The computation of prediction is supported by the combination of analytical and statistical prediction. Static analytical prediction is corrected by the statistical track data during the simulation. When the query from the load balance module is received, the prediction is fed back and a new partition is obtained by the mapping from activity to resource usage.

4.2 Quantization of resource usage

As discussed in activity tracking, track resource usage of transitions of models in simulation is a key technique in activity-based simulation. So, it is important to find measurement of resource usage. In the computer world, the computing power is decided by the performance of CPU and memory, especially CPU. Thus, the quantification of resource usage is realized by tracking of CPU and memory occupations.

Because DEVS is chosen to be the target formalism in implementation, the activity model is also realized in a compositional manner. Thus, Eq. (4) can be rewritten as follows:

$$\mathrm{AR}_H^R = \{A_j(s) \mid (v_H^{A_j(s)}(t)) > 0\}, \quad s \in S_j, j \in (1, m), \tag{6}$$

where $v_H^{A_j(s)}(t) = \{F_\delta^{A_j(s)}(i), C_\delta^{A_j(s)}(i), M^{A_j(s)}(i)\}$, A_j is an atomic DEVS model in models, m is the count of the atomic DEVS models in models, s is a sub-state of A_j while S_j is the state space of A_j, and $F_\delta^{A_j(s)}(i)$ is the frequency of transition i in A_j in state s. $F_\delta^{A_j(s)}(i)$ is the kernel parameter in the activity model. The prediction of activity is the prediction of transition frequency. $C_\delta^{A_j(s)}(i)$ and $M^{A_j(s)}(i)$ are the quantifications of the computation intensity of transition i. The quantifications are represented by computation and memory occupation of transition i. First, the definition of the transition computation is listed as follows:

$$\begin{gathered} C_\delta^{A_j(s)}(i) = K_{\text{proc}} \int_H^R p_{\text{occ}}^{A_j(s)}(i) \mathrm{d}t, \\ 0 < i < n, 0 < j < m, s \in S_j, \end{gathered} \tag{7}$$

where i is a transition belonging to A_j. There are n transitions in A_j in total. $A_j(s)$ is an atomic DEVS model with state s, $s \in S_j$. δ_i is the internal function of

A_j with transition i. δ_e is the external function of A_j with transition i. H is the physical time just before δ_i or δ_e. R is the physical time after δ_i or δ_e. K_{proc} is the coefficient of CPU performance, used to quantify the computational capability of CPU. $p_{\text{occ}}^{A_j(s)}(i)$ is the CPU occupation by transition i changing with time of model A_j.

The precise values of H, R, and $p_{\text{occ}}^{A_j(s)}(i)$ can be measured at runtime. However, K_{proc} is not a measurable parameter that can be represented by the performance data such as million instructions per second (MIPS) (MacNeil, 2004) and CPU clock speed. Due to the inconsistency of measurements, MIPS and CPU clock speed are not the reliable parameters to represent the computational capability. They are not operation system independent. The same test case will give different results in different operation systems. To solve the problem, we use a common standardized test model to acquire K_{proc}. According to Eq. (7),

$$K_{\text{proc}} = \frac{C_{\delta}^{A_j(s)}(i)}{\int_H^R p_{\text{occ}}^{A_j(s)}(i)\mathrm{d}t}. \tag{8}$$

Here the computation of the common standardized test model is known while other parameters can be collected in testing. All the machines involved in simulation are tested to obtain the performance coefficient. Moreover, the standardized test models are customized to satisfy the requirements from the models in different categories. The customized models can obtain accurate K_{proc} with consideration of the computation types such as float point operation and integer operation.

Memory occupation is also an important resource factor. The performance will be lowered greatly when the available memory cannot handle the memory requirements from the models. The memory occupation for transition i of A_j is

$$M^{A_j(s)}(i) = \int_H^R m_{\text{occ}}^{A_j(s)}(i)\mathrm{d}t, \tag{9}$$

where $m_{\text{occ}}^{A_j(s)}$ is the memory occupation that changes with time. The memory is occupied by transition i of model A_j.

Based on these definitions, the measurement becomes manipulable. The mapping table is constructed to apply the activity model in the simulation. The table matches the activity intensity and resource usage. The activity intensity is determined by the transition types while the resource usage is measured at runtime. The mapping table holds and updates the mapping relationships when the resource is allocated or reallocated. Note that the mapping relationship is organized by the unique activity ID $\text{ID}_{A_j(i)}^C$. The mapping table (MT) and ID are defined as

$$\begin{aligned} &\text{MT} = \{(\text{ID}_{A_j(i)}^C, (C_{\delta}^{A_j(s)}(i), M^{A_j(s)}(i)))\}, \\ &\text{ID}_{A_j(i)}^C = N_C + "\_" + N_{A_j} + "\_" + N_i, \end{aligned} \tag{10}$$

where N_C is the name of coupled DEVS model C, N_{A_j} is the name of atomic DEVS model A_j, and N_i is the name of transition i. During the simulation, CPU and memory occupations of transition i are tracked first. Then the quantified results are computed using Eqs. (7) and (8). Finally the two-tuples are assigned to each $\text{ID}_{A_j(i)}^C$ in the mapping table. The generation of the table is finished when all the transitions are covered. With the help of quantization, the resource needs can be calculated by activity prediction and the mapping table. The prediction outputs the activity frequency which lists the transitions to be triggered from H to R. With the track data in the mapping table, the transition intensities represented by resource usages such as CPU and memory occupations of each transition are found. By calculation of transition information and resource usages, the resource need from H to R is obtained in the end.

4.3 Resource-aware simulation framework

Fig. 9 shows the framework of resource-aware simulation. The framework is divided into two steps: modeling and simulation. In the activity enhanced modeling step, the meta-model for both the model itself and activity is built for activity-based simulation. Instantiated by the meta-model, the model generated in the domain is composed of the original model and activity model. With the help of model transformation, the model is transformed to the GPSF (DEVS) domain. The model in GPSF consists of the original model within atomic activity models represented in DEVS and the activity model called the activity predictor with connections with atomic activity models inside the original model. The activity predictor includes the model attribute related activity model and model related intrinsic activity. As discussed in Section 3.4, modelers do not have to describe the intrinsic activity features in the activity meta-model in the specific domain, and the transformation collects the instinct information and adds them to the activity model in GPSF (DEVS).

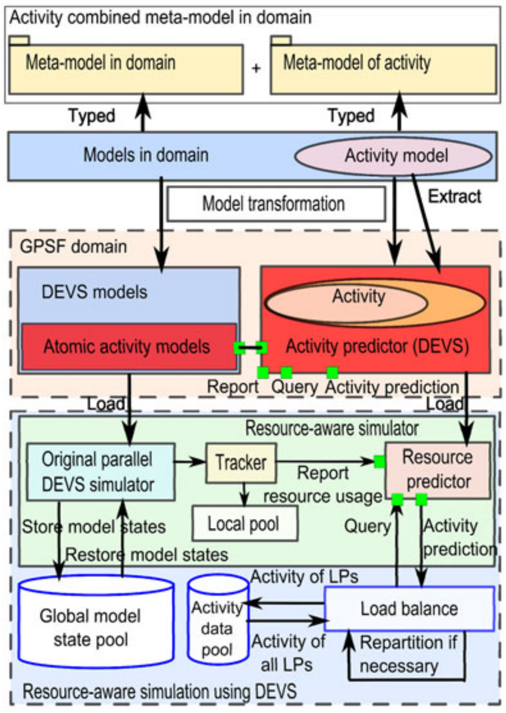

Fig. 9 Framework of resource-aware simulation

In the activity-based simulation step, the original parallel DEVS simulator is extended to construct the resource-aware simulator. The tracker module, resource predictor module, and local pool are added to the new simulator. The original models in DEVS are still simulated by the original parallel DEVS simulator. The tracker module tracks the CPU and memory occupations (resource usage) by transitions' computation. The track data is stored in the local pool and reported to the resource predictor. As a specific simulator for the activity predictor, the resource predictor receives the track data, makes activity prediction, and answers the activity query from the load balance module. Note that the activity discussed here is the activity of a local process in parallel simulation. The load balance module collects activity information from all local processes in parallel simulation. Meanwhile, the activity information is stored in the activity data pool in case of global activity computation for the load balance module. Repartition is made by the load balance module when the conditions of overload are met at runtime. Additionally, the global model state pool is implemented to store the states for models located at all local processes. When the load balance module triggers the repartition, the states of all models are stored in the pool. The states are restored from the pool after repartition.

4.4 Resource reallocation with activity prediction

With the help of activity prediction in the resource aware simulation framework, the load balance module is able to make resource reallocation (repartition) during simulation in order to avoid local process overload and improve the performance. The load balance module queries activity predictions from the activity predictor. Activity predictions provide the resource usage of each local process in future. So, the resource usage of all local processes can be predicted by the load balance module. Then the module adjusts the resource allocation to find the most optimal partition. Fig. 10 illustrates the work process in detail.

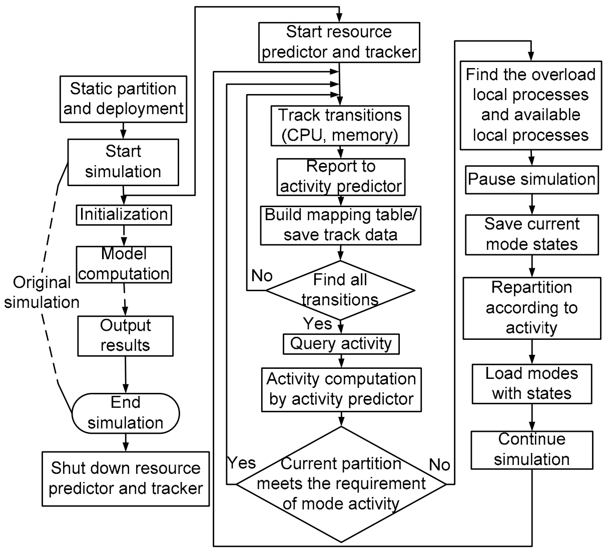
Start resource predictor and tracker
Static partition and deployment
Start simulation
Initialization
Model computation
Original simulation
Output results
End simulation
Shut down resource predictor and tracker
Track transitions (CPU, memory)
Report to activity predictor
Build mapping table/ save track data
No
Find all transitions
Yes
Query activity
Activity computation by activity predictor
Yes
Current partition meets the requirement of mode activity
No
Find the overload local processes and available local processes
Pause simulation
Save current mode states
Repartition according to activity
Load modes with states
Continue simulation

Fig. 10 Work process of the activity predictor

Different from the original simulation shown in the left part of the figure, activity prediction and resource reallocation are coupled into resource-aware simulation. Just after the starting of simulation, the resource predictor and tracker are activated. The tracker tracks the resource usage of transitions of models simulated in the original simulation. The track data is reported to the activity predictor simulated by the resource predictor. The mapping table, which records the mapping between the transitions and resource usage, is built gradually along the simulation. When all the transitions defined in activity models are covered, the preparation of activity prediction is finished. In response to the query from the load balance module, the activity predictor makes the prediction of the current local process and sends it back. If the current partition cannot meet the requirements of activities received from all the local processes, the load balance module finds the overload and available local processes. Then the resource allocations are triggered. The process of reallocation falls into five steps: pause the simulation, save the states of models in the global model state pool, repartition the models according to the activity, restore the state from the global state pool, and continue the simulation.

However, there are two problems still not interpreted in Fig. 10: how to do the load balance and how to find the overload local process in resource-aware simulation. For the first problem, we reuse the load balance algorithms in Deelman and Szymanski (1998) in the implementation of the resource-aware simulator. It is because our case study is a typical spatial distribution model. For the second problem, we give the definitions of average computation and memory occupation of the local process with the support of quantification of resource usage.

$$C_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)=\frac{1}{R-H}\int_H^R\sum_{i=1}^{n}\sum_{j=1}^{m}F_\delta^{A_j(s)}(i)\cdot C_\delta^{A_j(s)}(i)\mathrm{d}t, \quad (11)$$

$$M_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)=\frac{1}{R-H}\int_H^R\sum_{i=1}^{n}\sum_{j=1}^{m}m_{\mathrm{occ}}^{A_j(s)}(i)\mathrm{d}t, \quad (12)$$

$$\mathrm{Partition}=\sum_{i=1}^{k}P(l). \quad (13)$$

LP$_l$ is a local process in parallel simulation; there are k local processes in total. $P(l)$ is the partition located in LP$_l$. $A_j(s)$ is an atomic DEVS model belonging to $P(l)$ with state s. i is a transition of A_j. H is the input start time of LP$_l$ for computation. R is the input end time of LP$_l$ for computation. Overload means the resource need exceeds the local process limitations. With the help of $C_{\mathrm{LP}_l}^{\mathrm{Ave}}$ and $M_{\mathrm{LP}_l}^{\mathrm{Ave}}$, the conditions of local process overload from time H to R are

$$\begin{aligned}\mathrm{LP}_l^O(H,R)=\{&C_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)>>1,\\ &M_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)>>\mathrm{Mem}(\mathrm{LP}_l)\},\end{aligned} \quad (14)$$

where Mem(LP$_l$) is the memory allocated for LP$_l$.

$M_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)>>\mathrm{Mem}(\mathrm{LP}_l)$ means that the local process is overloaded when the average CPU occupation is much larger than 100% or the average memory occupation is much larger than the allocated memory. The reallocation is triggered when the local processes are in the conditions mentioned before. Note that the reallocation is sometimes time consuming. Thus, the reallocation cost is possibly larger than the improved performance. We have to take account of it before reallocation; otherwise, the simulation performance is even deteriorated.

5 Case study

5.1 Model description

We model the march map and its activity to testify the performance improvement brought on by activity enhanced modeling and activity-based simulation using DEVS. The march map constituted by grids is usually a typical case to simulate the combat simulation. The composition of grids is seen as the battlefield. The platoons move, rest, and fight in the transitable grid with the terrain such as field, pool, and road. When a grid meets the collision by platoons in different colors, the fight happens. In the modeling view, the march map is modeled in the spatial-oriented representation. Grid is an atomic model that cannot be decomposed again. The grids are spatially linked together to construct the march map. Platoons are modeled as the events sent and received by grids.

5.2 Activity model of the march map

Obviously, the march map is a typical spatial compositional model. So, the quantified definition of the activity model in Eq. (6) can be improved with the help of the activity region definition in Eq. (5). The definition for the activity model of the march map is listed as

$$\mathrm{AR}_H^R=\{G_{(x,y)}(s)\,|\,(v_H^{G_{(x,y)}(s)}(t))>0\},\qquad(15)$$
$$s\in S_{(x,y)},\ x\in(1,m),\ y\in(1,n),$$

where

$$v_H^{G_{(x,y)}(s)}(t)=\left\{F_\delta^{G_{(x,y)}(s)}(i),C_\delta^{G_{(x,y)}(s)}(i),M^{G_{(x,y)}(s)}(i)\right\}.$$

$G_{(x,y)}$ is an atomic grid model with location (x, y). The width of grids is m while the length is n.

According to the description of the march map, a grid can be linked with other grids in four directions. The states of the grid are composed of terrain, platoons, platoon state, and abut grids. It is easy to build the meta-model of the march map with these attributes. In the view of activity enhanced modeling, the grids with platoons inside are active because of the computation for platoon tasks. So, the transitions of the grid are the basic elements in atomic activity for the activity model. Based on the merging operations discussed in Section 3.2, the activity combined meta-model of the march map is devised as Fig. 11. The attributes are listed in Table 1. The complementary

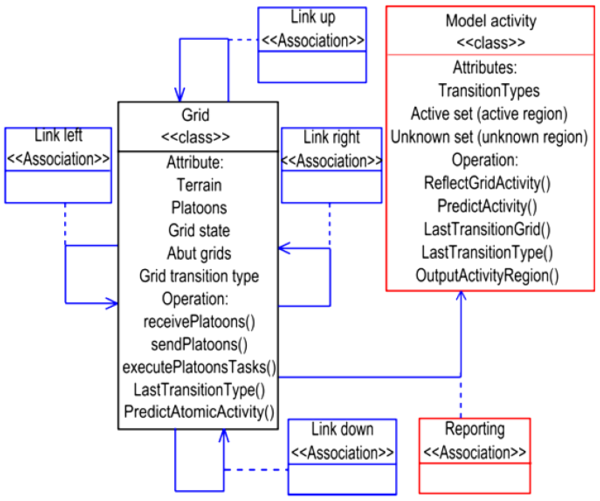

Fig. 11 Activity combined meta-model

Table 1 Grid attributes

Attribute name	Value set
Terrain	Road, field, river, mountain
Platoon	Null, red platoon, blue platoon, red and blue platoons
Platoon state	Moving, resting, fighting
Abut grids	Left, left-up, left-down, down, right-down, right, right-up, up
Grid transition type	Null, Move_Road, Move_Field, Rest_Road, Rest_Field, Fight_Road, Fight_Field

activity attributes and operations are merged inside the grid model. Grid is linked to model activity by the association reporting as discussed in Section 3.2. Platoon is modeled to be the events with attributes such as size, DamageState, task, direction, and CurState. The model of the march map is instantiated by the meta-model as shown in Fig. 12.

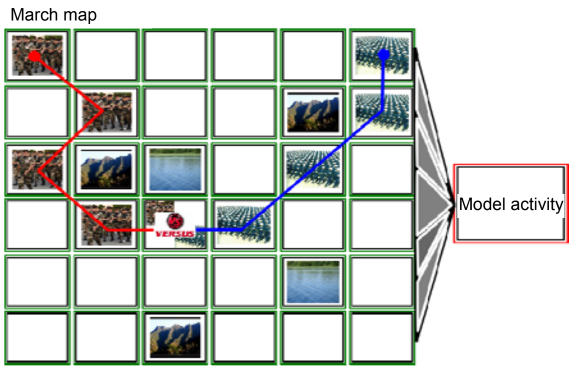

Fig. 12 Model of the march map
Two platoons move on the 6×6 march map; they happen to meet at some grid and the fighting is triggered

5.3 DEVS representation of the march map

Based on activity enhanced modeling, models with the activity model have to be transformed into the GPSF. Therefore, the models of the march map in a specific domain are transformed into DEVS before simulation. According to the rules of model transformation, the march map represented in DEVS is shown as

$$\delta_G=\{X_G, S_G, Y_G, \delta_{\mathrm{int}}, \delta_{\mathrm{ext}}, \lambda, t_a\},$$

where X_G={Platoons}, S_G={Terrain, Platoons, AbutGrids, ActiveState, GridTransitionTypes}, Y_G={Platoons, NextGrid, LastTransitionType}, δ_{int}={ExecutePltoonsTasks()} is the internal grid function, δ_{ext}={GetLeavingPlatoons(), LastTransitionType(), GetActiveState()} represents the external grid subfunctions including the leaving platoons and atomic activity information, λ={SendPlatoons(), SendLastTransitionType(), SendActiveState(), SendNextGrid()} sends platoons to the other grid, and outputs activity information to the activity predictor, and t_a returns the time stamp for the next prediction.

The atomic activity model is imbedded in the grid model. ActiveState depends on the attributes terrain and platoon task. GridTransitionTypes is decided by the platoon task and the activity prediction is made by NextGrid calculated according to the moving directions of platoon. These value sets of the grid states are listed in Table 1. In other words, the platoon task and platoon direction decide the next active grid and its state. The next possible grid is one of the eight neighbors listed in the abut grids. So, the atomic activity prediction outputs both the current and next active states of the grid. Moreover, the activity predictor is obtained from the model transformation:

$$\delta_{\mathrm{P}}=\{X_{\mathrm{P}}, S_{\mathrm{P}}, Y_{\mathrm{P}}, \delta_{\mathrm{int}}, \delta_{\mathrm{ext}}, \lambda, t_{\mathrm{a}}\},$$

where X_{P}={GridTransitionTypes, Query, TrackData, NextGrids}, S_{P}=ActiveSet$\cup$PassiveSet$\cup$UnknownSet$\cup$AllT-ransitionsMappings, Y_{P}={LastTransitionActivity, ActivityPrediction}, δ_{int}={MakeTransitionMappings(), PredictActivity()} is the internal activity prediction calculation, δ_{ext}={ReflectGridTransitions(), RecieveNextGrids(), ReceiveGridActiveState(), ReceiveTrackData()} represents the eternal activity functions, including the sub-activities collection or activity prediction calculation using track data, λ={OutputActivity()} outputs results after activity prediction, and t_{a} returns the time stamp for the next prediction. The activity predictor collects the activity from all the grids via the reporting association. The activity region including ActiveGridSet, PassiveGridSet, and UnknownGridSet is generated by assembling the separated grid activity. ActiveGridSet is composed of the grids within platoon. PassiveGridSet consists of the grids where the terrain is mountain or river. The remaining grids compose the UnknownGridSet. The activity region is changing by the movements of platoons during the simulation. AllTransitionsMappings which are implemented by the mapping table are built by the GridTransitionTypes and TrackData from the tracker mentioned before. When the query from the load balance module is received, the activity predictor replies to the activity predictions.

5.4 Activity prediction in resource-aware simulation

The resource-aware simulation runs on a computer with a four-core Intel i5-2300 2.8 GHz CPU and 4 GB RAM. The models of the march map represented in DEVS are programmed in C++ in an object oriented fashion. The DEVS simulator of tool One-Model (Boukerche and Das, 1997; Guo, 2013) is used as the original parallel DEVS simulator. The communications between simulators in different local processes use the message passing interface (MPI) message passing library. The tracker, resource predictor, and load balance module are also implemented in C++. The local pool, track data pool, and global model state pool are all implemented by MySQL (Welling and Thomson, 2003).

As discussed in Section 4.4, the mapping table is built in simulation to store the track data for the activity predictor. The table is shown below within the track data of transitions in grids:

$$\begin{bmatrix} G_{(1,1)}\_\text{Move\_Road} & (0.014, 232\ \text{KB}) \\ G_{(1,1)}\_\text{Rest\_Road} & (0.009, 240\ \text{KB}) \\ G_{(1,1)}\_\text{Move\_Field} & (0.023, 241\ \text{KB}) \\ G_{(1,1)}\_\text{Rest\_Field} & (0.008, 220\ \text{KB}) \\ G_{(2,2)}\_\text{Move\_Road} & (0.001, 217\ \text{KB}) \\ G_{(2,2)}\_\text{Rest\_Road} & (0.003, 249\ \text{KB}) \\ \cdots & \cdots \end{bmatrix}. \tag{16}$$

The left column of the mapping table lists the transition IDs which are composed according to Eq. (10). The right column lists the two-tuples including computation and memory occupation. Memory occupations are directly assigned by track data, but the transition computations recorded directly by the operation system need to be corrected because of the measurement errors. The correction is made according to the average of CPU occupations. Based on Eq. (7), the computation of transitions is quantified as

$$C_{\delta}^{G_{(x,y)}}(i)=\frac{K_{\text{proc}}}{n_{\text{pas}}^{G_{(x,y)}}(i)}\sum_{r=1}^{n_{\text{pas}}^{G_{(x,y)}}(i)} p_{\text{occ}}^{G_{(x,y)}}(i_{(r)}). \tag{17}$$

$G_{(x,y)}$ is a grid in the march map, (x, y) is the location of the grid while m is the size of the grids, i is a transition of the grid such as Move_Road, $C_{\delta}^{G_{(x,y)}}(i)$ is the computation usage by transition i, and $p_{\text{occ}}^{G_{(x,y)}}(i_{(r)})$ is the rth ($1\leq r\leq n_{\text{pas}}^{G_{(x,y)}}(i)$) record of CPU occupation by transition i of $G_{(x,y)}$. These values of CPU occupation are recorded by the task manager of the operation system at runtime. $n_{\text{pas}}^{G_{(x,y)}}(i)$ is the amount of transition i which has been executed by $G_{(x,y)}$. $p_{\text{occ}}^{G_{(x,y)}}(i_{(r)})$ and $n_{\text{pas}}^{G_{(x,y)}}(i)$ are restored in the track data pool mentioned before; they are tracked to predict the computation for transition i in model $G_{(x,y)}$.

Figs. 13 and 14 present two snapshots of activity prediction and partition during simulation. In Fig. 13, both platoons in red and blue (references to color refer to the online version of this figure) just finished the first step in their initial grids. They will be sent to the next grids in the next step. Based on the current states of grids and the activity model, the ActiveSet, PassiveSet, and UnknownSet are generated by activity prediction. As shown in Fig. 13, $G(2, 2)$ and $G(6, 2)$ in red compose the ActiveSet. Meanwhile, the possible grids for platoons filled by green belong to the UnknownSet. The remaining grids filled by grey compose the PassiveSet. The partition is still in its initial state in which the 36 grids are distributed evenly to four local processes. According to Eqs. (11)–(13), the activity of grids can be predicted as

$$C_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)=\frac{1}{\Delta t}\sum_{G_{(x,y)}\in P(l)}\sum_{i=1}^{n}F_{\delta}^{G_{(x,y)}}(i)C_{\delta}^{G_{(x,y)}}(i)\Delta t, \quad (18)$$

$$M_{\mathrm{LP}_l}^{\mathrm{Ave}}(H,R)=\frac{1}{\Delta t}\sum_{G_{(x,y)}\in P(l)}\sum_{i=1}^{n}m_{\mathrm{occ}}^{G_{(x,y)}}(i)\Delta t. \quad (19)$$

$C_{\delta}^{G_{(x,y)}}(i)$ is the computation by transition i of $G_{(x,y)}$, computed using Eq. (17), and $F_{\delta}^{G_{(x,y)}}(i)$ is the transition frequency predicted for transition i of $G_{(x,y)}$. The frequency is predicted using the internal activity prediction function in the activity predictor. In our case, this information is acquired with the help of the current locations and the simulation scenario. The platoons follow the assigned tasks in the scenario, so the next actions can be predicted according to the tasks and current states. Δt is the physical time cost for each step. Because the march map is a typical chess-like model, the models are computed step by step. So, the time cost of computation is recorded according to the time step in simulation. In other words, the time factor in the computation of the local process is seen as an average value for transitions. As a result, the computation and memory occupation of local processes are

$$C_{\mathrm{LP}_l}^{\mathrm{Ave}}(\Delta t)=\sum_{G_{(x,y)}\in P(l)}\sum_{i=1}^{n}F_{\delta}^{G_{(x,y)}}(i)\cdot C_{\delta}^{G_{(x,y)}}(i),$$

$$M_{\mathrm{LP}_l}^{\mathrm{Ave}}(\Delta t)=\sum_{G_{(x,y)}\in P(l)}\sum_{i=1}^{n}m_{\mathrm{occ}}^{G_{(x,y)}}(i).$$

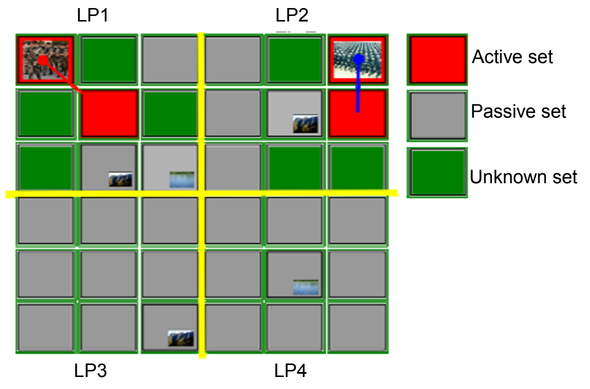

Fig. 13 Activity prediction and partition of the first step

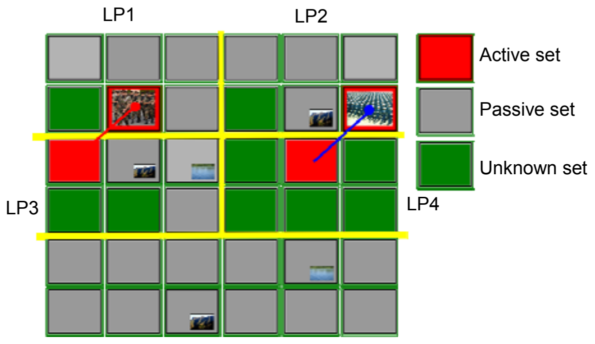

Fig. 14 Activity prediction and partition of the second step

Obviously, the models located in LP3 and LP4 all belong to PassiveSet. Thus, the $C_{\delta}^{G_{(x,y)}}(i)$ and transition frequencies $F_{\delta}^{G_{(x,y)}}(i)$ predicted by the activity predictor in LP3 and LP4 are all zero.

So, the predictions of the computation of local processes are

$$C_{\mathrm{LP}_1}^{\mathrm{Ave}}(\Delta t)>>C_{\mathrm{LP}_3}^{\mathrm{Ave}}(\Delta t)=0,$$
$$C_{\mathrm{LP}_2}^{\mathrm{Ave}}(\Delta t)>>C_{\mathrm{LP}_4}^{\mathrm{Ave}}(\Delta t)=0.$$

As a result, the resource allocated to LP3 and LP4 is wasted in the current partition in this step; it is necessary to repartition the models. Fig. 14 gives the activity prediction and new partition of the next step. With the help of move directions of platoons, ActiveSet, PassiveSet, and UnknownSet are updated using the activity model. According to the activity information, the load balance module reallocates the resource, and the new partition ignores the grids in the bottom, which will not be activated in the near future. Now the number of grids located in each local process is six. Twenty-four grids are maintained in the simulation with four local processes. According to Eq. (13), the new partition is

$$\begin{aligned}&\text{Partition}=\{P(1), P(2), P(3), P(4)\},\\ &P(1)=\{G_{(x,y)}|(1\le x\le 3,\ 1\le y\le 2)\},\\ &P(2)=\{G_{(x,y)}|(4\le x\le 6,\ 1\le y\le 2)\},\\ &P(3)=\{G_{(x,y)}|(1\le x\le 3,\ 3\le y\le 4)\},\\ &P(4)=\{G_{(x,y)}|(4\le x\le 6,\ 3\le y\le 4)\}.\end{aligned}$$

The new partition distributes ActiveSet and UnknownSet evenly into four local processes. Theoretically, without consideration of cost by activity prediction and load balance, the load from models is two-thirds of that in the static partition case.

In addition, Fig. 15 presents the activity prediction supported by analytic prediction. Usually, the dynamic activity predictions are limited inside the moving windows as the models are non-deterministic and random when the simulation is being made. In our case, however, the platoon routes are assigned in initialization at the beginning of the simulation. The chess-like models that follow the certain rules do not bring randomness into the simulation. The moving windows are extended to the whole simulation process without bringing errors. So, the activity prediction for the whole simulation can be made for the march map. In consequence, ActiveSet and PassiveSet are reported to the load balance module to find the optimal partition shown in Fig. 15.

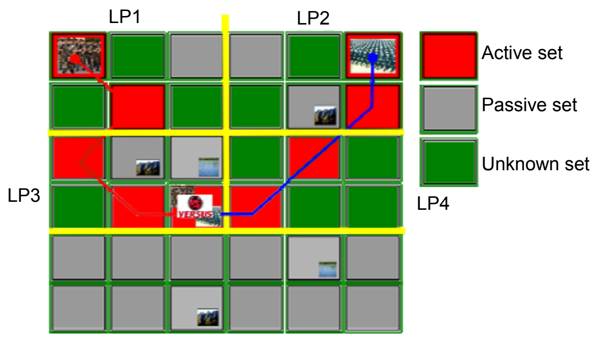

Fig. 15 Partition with analytic activity prediction

5.5 Results analysis

Fig. 16 shows the experimental results of using the march map. Activity combined simulation is more efficient in comparison with the original simulation. With increase in grid size, the physical time cost by activity combined simulation increases linearly, while the original simulation reflects a quadratic increase of physical time needed. The great performance improvement comes from the activity prediction which indicates the active grids. Though the grid size increases in a quadratic manner, the platoons are still limited in some fixed area according to the assigned tasks. The range of the active area relies on the length of the march map. In consequence, the grids that platoons will not enter are seen as passive so that the simulator can ignore them. The overhead of these grids is saved. As a result, the performance is improved a lot.

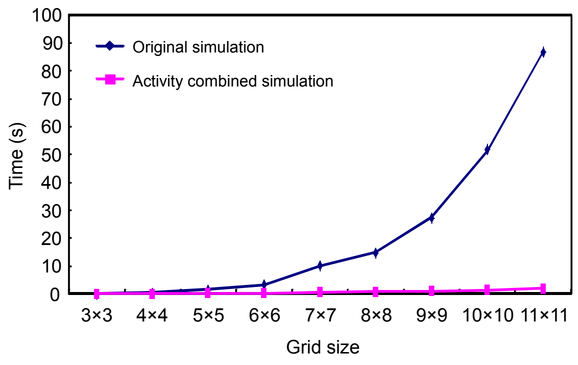

Fig. 16 Performance of activity combined simulation

From another aspect, Fig. 17 illustrates the accuracy of the transition computation. The grid transitions are categorized into several levels such as Move_Road and Fight_Field listed in Table 1. Because the models of grids are all isomorphic, the transitions are summarized together to show the obvious change in experiment results. The computation of transition Move_Road of grids is predicted in the following equation:

$$C_{\delta}^{\text{Grids}}(i_{\text{M}},T)=\sum_{x=1}^{m}\sum_{y=1}^{m}F_{\delta}^{G_{(x,y)}}(i_{\text{M}},T)\cdot C_{\delta}^{G_{(x,y)}}(i_{\text{M}},T),\quad(20)$$

where T is the simulation time, i_{M} is the transition Move_Road, $C_{\delta}^{\text{Grids}}(i_{\text{M}},T)$ is the prediction of integral computation by transition Move_Road of the whole grid models at simulation time T, $C_{\delta}^{G_{(x,y)}}(i_{\text{M}},T)$ is the computation by transition Move_Road of $G_{(x,y)}$ computed using Eq. (17) at simulation time T, and $F_{\delta}^{G_{(x,y)}}(i_{\text{M}},T)$ is the frequency of transition Move_

Road in $G_{(x,y)}$. The frequency is predicted using the internal activity prediction function in the activity predictor at simulation time T.

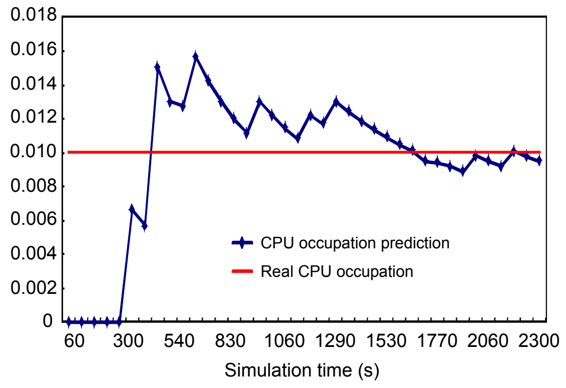

Fig. 17 Prediction of computation for Move_Road transitions

Based on Eq. (20), the experiments show how to obtain the accurate transition computation. The horizontal line is the real CPU occupation for the Move_Road transition. The real value collected at the end of the simulation is used as an accurate datum line for the prediction.

Fig. 17 indicates that the prediction at the beginning stays at zero because of the inaccuracy of measurement (the tracked CPU occupation of Move_Road approximates to zero). Thanks to the accumulation of the track data, the prediction approaches the accurate value along the simulation. According to Fig. 17, the predicted value oscillates closely to the datum line after 1400 s. The prediction is favored by the correct transition frequency prediction and the growing accuracy of the average CPU occupation by the transition.

In summary, the march map case shows how to implement the activity combined model in our resource-aware simulation framework. The information from separated atomic activity is integrated by model activity to obtain the activity region. The region focuses on the locations of the active grid and finds the active area for the simulation. With the feedback of activity, resources can be allocated more reasonably and the performance is improved. It can be concluded that the larger the march map, the better the performance that will be acquired from activity prediction.

6 Conclusions

Modeling activity has been shown to be a good method for modelers to make a great contribution in the improvement of simulation efficiency. The activity definition is given in a continuous system first and extended to discrete expression later. Considering the model structure, the activity in the compositional model is discussed. Furthermore, the spatial activity model is defined to add spatial information to the formalization. Activity enhanced modeling starts with the activity meta-model, and the meta-model in a specific domain will be transformed to GPSF representation. The transformation not only transforms the attribute related activity model, but also generates the model related intrinsic activity model. We present the resource-aware simulation framework to drive the activity model in simulation. Activity tracking and quantization are implemented in the framework. The case study gives an example to apply the activity model in simulation. It testifies that the modeling considered activity improves the simulation performance.

In summary, the features of activity-based simulation are:

1. Activity enhanced modeling gives a method to define the activity model in a specific domain.

2. Domain specific activity modeling maintains the advantages in minimizing modeling accidence.

3. The quantization of activity and resources makes possible the application of the activity model in the simulation.

4. The resource-aware simulation framework implements the activity-based simulation, integrating the activity in both modeling and simulation aspects.

In the future, more cases are needed to extract the more refined activity meta-model. The meta-modeling of intrinsic activity information should be considered in activity enhanced modeling. We also have to do more work on resource reallocation in the resource-aware simulation framework. The observation and measurement of the reallocation costs are needed so that we can decide whether the performance is indeed improved.

References

Balsamo, S., di Marco, A., Inverardi, P., *et al.*, 2004. Model-based performance prediction in software development: a survey. *IEEE Trans. Softw. Eng.*, **30**(5):295-310. [doi:10.1109/TSE.2004.9]

Boukerche, A., Das, S.K., 1997. Dynamic load balancing strategies for conservative parallel simulations. Proc. 11th Workshop on Parallel and Distributed Simulation, p.20-28. [doi:10.1109/PADS.1997.594582]

Concepcion, A.I., Zeigler, B.F., 1988. DEVS formalism: a framework for hierarchical model development. *IEEE Trans. Softw. Eng.*, **14**(2):228-241. [doi:10.1109/32.4640]

Czarnecki, K., Helsen, S., 2003. Classification of model transformation approaches. OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture, p.1-17.

D'Abreu, M.C., Wainer, G.A., 2005. M/CD++: modeling continuous systems using Modelica and DEVS. 13th IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, p.229-236. [doi:10.1109/MASCOTS.2005.36]

Deelman, E., Szymanski, B.K., 1998. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. Proc. 12th Workshop on Parallel and Distributed Simulation, p.46-53. [doi:10.1109/PADS.1998.685269]

Emerson, M., Sztipanovits, J., 2006. Techniques for metamodel composition. 6th Workshop on Domain Specific Modeling, p.123-139.

Guo, G., 2013. User Manual of OneModel. National University of Defense Technology, Changsha, China (in Chinese).

Hu, X.L., Ntaimo, L., 2006. Dynamic multi-resolution cellular space modeling for forest fire simulation. Proc. Spring Simulation Multi-conf., p.95-102.

Hu, X.L., Ntaimo, L., 2008. DEVS-FIRE: towards an integrated simulation environment for surface wildfire spread and containment. *Simulation*, **84**(4):137-155. [doi:10.1177/0037549708094047]

Hu, X.L., Zeigler, B.P., 2013. Linking information and energy—activity-based energy-aware information processing. *Simulation*, **89**(4):435-450. [doi:10.1177/0037549711400778]

Hu, X.L., Muzy, A., Ntaimo, L., 2005. A hybrid agent-cellular space modeling approach for fire spread and suppression simulation. Proc. Winter Simulation Conf., p.248-255. [doi:10.1109/WSC.2005.1574258]

Jammalamadaka, R., 2003. Activity Characterization of Spatial Models: Application to Discrete Event Solution of Partial Differential Equations. PhD Thesis, the University of Arizona, Tucson, USA.

Kelly, S., Tolvanen, J.P., 2008. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press, USA.

Lagerström, R., Chenine, M., Johnson, P., *et al.*, 2008. Probabilistic metamodel merging. CAiSE Forum, p.25-28.

MacNeil, T., 2004. Don't be misled by MIPS. IBM Systems Magazine Webinars.

Muzy, A., Zeigler, B.P., 2008. Introduction to the activity tracking paradigm in component-based simulation. *The Open Cybern. Syst. J.*, **2**:30-38.

Muzy, A., Nutaro, J.J., Zeigler, B.P., *et al.*, 2008. Modeling and simulation of fire spreading through the activity tracking paradigm. *Ecol. Model.*, **219**(1-2):212-225. [doi:10.1016/j.ecolmodel.2008.08.017]

Muzy, A., Touraille, L., Vangheluwe, H., *et al.*, 2010. Activity regions for the specification of discrete event systems. Proc. Spring Simulation Multi-conf., p.1-7. [doi:10.1145/1878537.1878679]

Muzy, A., Jammalamadaka, R., Zeigler, B.P., *et al.*, 2011. The activity-tracking paradigm in discrete-event modeling and simulation: the case of spatially continuous distributed systems. *Simulation*, **87**(5):449-464. [doi:10.1177/0037549710365155]

Petriu, D.C., Shen, H., 2002. Applying the UML performance profile: graph grammar-based derivation of LQN models from UML specifications. Proc. 12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation, p.159-177.

Qiu, F.S., Hu, X.L., 2013. Spatial activity-based modeling for pedestrian crowd simulation. *Simulation*, **89**(4):451-465. [doi:10.1177/0037549711435950]

Sendall, S., Kozaczynski, W., 2003. Model transformation: the heart and soul of model-driven software development. *IEEE Softw.*, **20**(5):42-45. [doi:10.1109/MS.2003.1231150]

Shibata, D., Alfenas, D., Guiraldelli, R., *et al.*, 2012. Activity based scheduling simulator for product transport using pipeline networks. Proc. Winter Simulation Conf., p.1-12.

Syriani, E., Vangheluwe, H., 2007. Programmed graph rewriting with DEVS. 3rd Int. Symp. on Applications of Graph Transformations with Industrial Relevance, p.136-151. [doi:10.1007/978-3-540-89020-1_11]

Vangheluwe, H., 2000. DEVS as a common denominator for multi-formalism hybrid systems modelling. IEEE Int. Symp. on Computer-Aided Control System Design, p.129-134. [doi:10.1109/CACSD.2000.900199]

Vangheluwe, H., de Lara, J., 2004. Computer automated multi-paradigm modelling for analysis and design of traffic networks. Proc. Winter Simulation Conf., p.249-258.

Welling, L., Thomson, L., 2003. PHP and MySQL Web Development. Sams Publishing, USA.

Zeigler, B.P., Praehofer, H., Kim, T.G., 2000. Theory of Modeling and Simulation (2nd Ed.). Academic Press, USA.

Zeigler, B.P., Jammalamadaka, R., Akerkar, S.R., 2004. Continuity and change (activity) are fundamentally related in DEVS simulation of continuous systems. 13th Int. Conf. on AI, Simulation, Planning in High Autonomy Systems, p.1-13.