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 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, Belgium*)

([3]*School of Computer Science, McGill University, Montréal, Canada*)

[†]E-mail: nudtcb9372@gmail.com

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

Abstract: Improving simulation performance using activity tracking has attracted attention in the modeling field in recent years. The reference to activity had been successfully used to predict and promote the simulation performance. But tracking activity only makes use of the inherent performance information contained in the models. In order to extend the activity prediction in modeling, we propose the Activity Enhanced Modeling with 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 solve 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 DEVS.

Key words: Activity Tracking, Activity enhanced modeling, DEVS, Resource-aware Simulation Framework
doi:10.1631/jzus.C1300121 **Document code:** A **CLC number:**

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 roughly are divided into two types: Dynamic Load Balancing and Model-Based Performance Predicting (Balsamo et al., 2004). The former researches focus on the performance data at run-time, the data 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 researches build 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 by the training process. Lots of 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 method, 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 (Xiaolin Hu, 2008), Pipeline of Product Transport (Shibata et al., 2012) and Pedestrian Crowd simulation (Qiu and Hu, 2013). The spatial activity information from tracking is used to

‡ Corresponding author

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

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 the activity tracking and the quantization of activity discussed above are still limited in the **DEVS** (Zeigler et al., 2000) field. It does not apply the advantages of Domain Specific Modeling (DSM). The activity-based techniques still meet the problems list below:

- How to quantify activity of models?
- How to apply activity-based techniques in specific domains?
- 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**. **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**.

The rest of the paper is organized as follows: Section 2 introduces the definition of activity. Section 3 presents **Activity Enhanced Modeling**. The **Activity Meta-Model** in a specific domain and activity model transformation are detailed. Section 4 presents **Activity-Based Simulation** and the implementation of **Resource-Aware Simulation Framework**. Section 5 gives the case study to show the improvement using **Activity-Based Simulation**. Section 6 concludes the paper with the summary and discusses the future work.

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 dimension. 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, 2006) and Missile Launch are seen as typical examples to describe activity in both spatial and temporal dimensions. Figure 1 shows the activity intensity of Forest Fire. The black and red spots represent the passive and active activity regions respectively. Similarly, Figure 5 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 lowers the simulation performance. So, it is necessary to reallocate the resource under the instruction of activity.

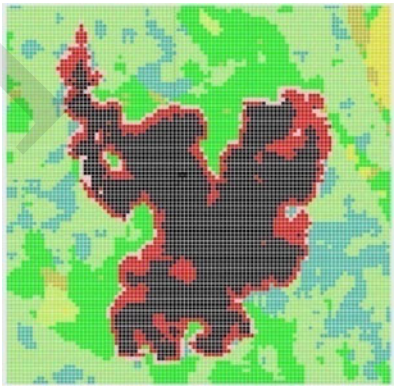

Fig. 1 Activity Region of Forest Fire (Hu, 2008)

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. (2010a; 2010b) is given below:

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

$\Phi(t)$ represents the mathematical abstraction of the real system that is either the continuous system or

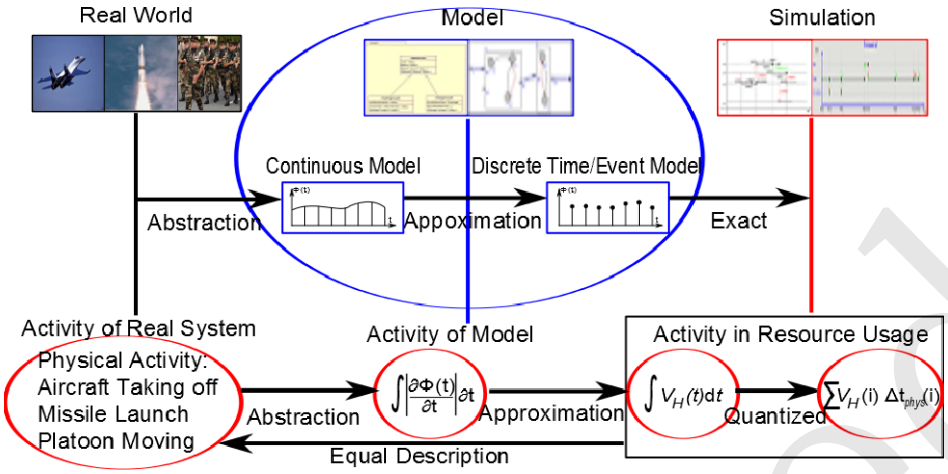

Fig. 2 Activity Formalization, from real world to simulation

discrete system (Discrete Time / Event). In the computer world, models are constructed on the basis of the mathematical description. All the continuous models are discretized to be discrete time or discrete event models. It is because models can only be simulated in a discrete manner in a computer. Therefore the mapping from mathematical model to simulation model is an approximation process. Likewise, the continuous activity model $A(t)$ is also 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:

$$A_H^R = \int_H^R v_H(t)dt \tag{2}$$

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. Lots of 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 Figure 3, 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 as

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

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

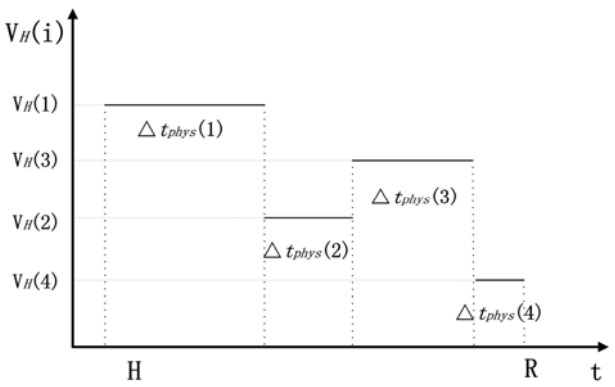

Fig. 3 Approximation of $v_H(i)$

Corresponding with Figure 3, Figure 4 gives a simple example of a car moving in a 5*3 map. The moving from start point to 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 number 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 four, two, three and one respectively, the corresponding $\Delta t_{phys}(i)$s are shown in Figure 3. Obviously, the larger the transition frequency, the longer the physical time cost will be. The physical time cost is decided by the computational intensity of transition. It is worth noting that the computational intensity of atomic transition **Move** is an important parameter in the quantification of the activity model in the moving car example.

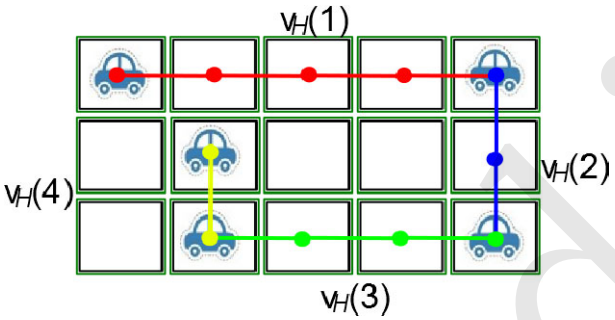

Fig. 4 A simple Moving Car example of $v_H(i)$

Figure 2 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 are also able to find physical activities. The behaviors like aircraft taking off, missile launch, and platoon moving are considered to be the physical activities of real systems. As discussed before, the physical activities are modeled in Equation 1 in the modeling view. Similar to the approximation from continuous model to discrete event or discrete time model, the activity model is also defined in Equation 2 by approximation. Because the performance is measured by the resource usage in the computer, the activity model is finally formalized in Equation 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 Compositional Model

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

$$AS_H^R = \{A_{c_j}^{AC} \mid (v_H^{c_j}(t)) > 0\},$$
$$(c_j \in C_s, s \in AC, H < t < R) \quad (4)$$

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 sub-components' states. It belongs to AC. C_s is the set of the 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 belongs to s. AS_H^R gives a generalized activity representation relying on the 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 the atomic components with similar activities. **Activity Region** defines how these regions evolve with time. The definition is given as follows:

$$AR_H^R = \{A_{c_{(x,y)}}^{AC} \mid (v_H^{c_{(x,y)}}(t)) > 0\},$$
$$(c_{(x,y)} \in C_s, s \in AC, H < t < R) \quad (5)$$

where $C_{(x,y)}$ is a sub-component at location (x, y). **Activity Region** can be refined with spatial information from the atomic activity model in the temporal dimension. Take the Missile Launch for example in Figure 5, the activity evolves with time elapsing. The activity stays 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 activity at different levels (intensities) is located at different locations respectively. So the **X** axis is a complement to the description of missile activity that evolves with distance to launch position. The bottom of the Figure 5 shows the **Refinement** by distance information. The Activity-Time-Distance axis is used to show the activity evolution along both the time and distance. Actually, the activity spots with time and distance are the **Activity Region**s. In the opposite way, the concise and portable activity expression is more appropriate in some specific cases. So the abstraction named **Collapse** from **Activity Region** to temporal atomic activity is necessary. **Collapse** can be seen as the projection from **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.

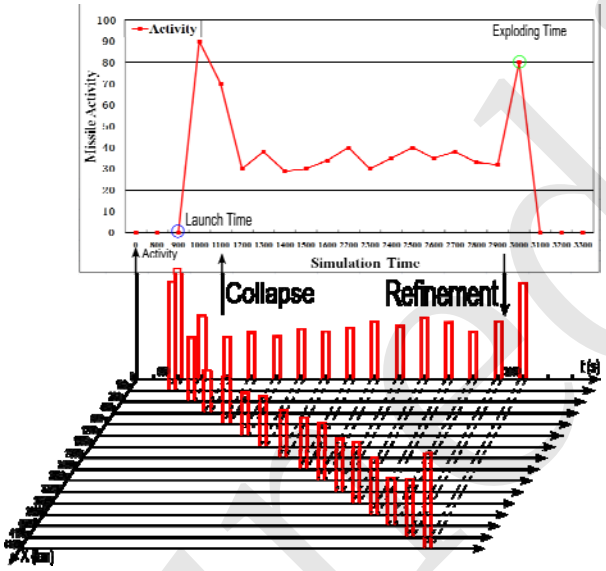

Fig. 5 Temporal Activity of Missile Launch

3 Activity Enhanced Modeling

On the basis of the activity formalization discussed before, we build the model to predict the activity in 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 it 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 too. 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** is capable of modeling activity for the models in the domain. Actually, the **Activity Meta-Model** provides the interfaces for modelers to implement the methods to do the 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 only supports the temporal activity prediction for the atomic component. On the other hand, **MA** predicts the activity of the whole model in an entire manner. The hierarchy, structure and spatial information are considered in **MA**. **MA** receives the sub-activities from AA models by association **Reporting**, 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 UML are shown in the left of Figure 6.

Atomic Activity (**AA**) is designed on the basis of Equation 3. The equation 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 the **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 the transition frequency and intensity. The results returned by the interface are the transitions which will be triggered in future. **LastTransitionType** is used to indicate the last transition type in order to match the physical time cost when the

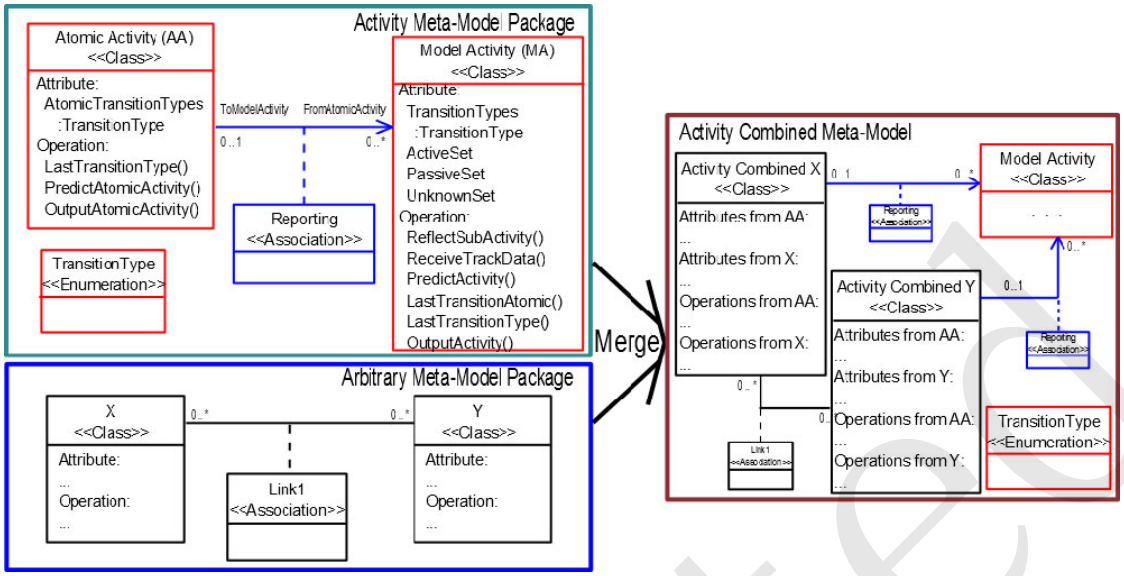

Fig. 6 Activity Meta-Model

simulation is running. **Model Activity** predicts activity in a global perspective. The **TransitionTypes** contain all the transition types occurring in this model. The transition types are marked with the sub-component. They represent the different activity frequency and intensity, compared to **AA**, **MA** emphasizes the **ActiveSet** that is the projection of **Active Region** in the spatial 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 the 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 **Activity Meta-Model** and the **Arbitrary Meta-Model** with the meta-model merge techniques (Emerson and Sztipanovits, 2006; Lagerström et al., 2008). The two packages are merged to be the **Activity Combined Meta-Model**, which is shown in the right of Figure 6. The merge operations are list below:

- **Atomic Activity** (**AA**) is merged into each atomic class in **Arbitrary Meta-Model**. The attributes and operations are directly copied to the newly atomic class. **AA** does not exist in the **Activity Combined Meta-Model** any more.
- **Model Activity** (**MA**) and **TransitionType** are kept in the **Activity Combined Meta-Model**.
- The **Reporting** associations between **AA** model and **AA** model are resumed. All the atomic classes within activity parts are linked to **MA** by **Reporting** associations.
- The associations in **Arbitrary Meta-Model** are still kept in **Activity Combined Meta-Model**.

We execute the operations as shown in Figure 6, **MA** and **TransitionType** are copied to **Activity Combined Meta-model**. The activity attributes from **AA** are embedded into the atomic class X and Y. The association **Reporting** is linked to each atomic class to connect **Model Activity**. The association Link1 between atomic class X and Y is retained.

3.3 Activity Combined Model in General Purpose Simulation Formalism (GPSF)

Activity Combined Meta-Model is still an application of Domain Specific Modeling (DSM). DSM

reduces the modeling complexity but bring out the difference in simulation (Kelly, 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 a lot of work has been done (D'Abreu and Wainer, 2005). Figure 7 shows the idea of how to do the modeling with activity from the domain to efficient simulation. The wide top of the sand glass indicates plenty of activity combined with modeling formalisms in the domains, the narrow middle of the glass means the multi-formalisms have to be transformed to unified **GPSF**, and the wide glass bottom represents the **Resource-Aware Simulator** that is implemented with many techniques to improve efficiency. 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 highly efficient simulator. Thanks to the precise definition, modularity and hierarchy, Discrete EVent Specification system (**DEVS**) is chosen to be the **General-Purpose Simulation Formalism**. The introduction of **DEVS** is detailed in Concepcion and Zeigler (1988) and Vangheluwe (2000). This portable formalism can be expanded more than a discrete event field. Lots of 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 **Activity Combined Meta-Model** to **DEVS** is composed of two parts: transformation from meta-model in a specific domain to

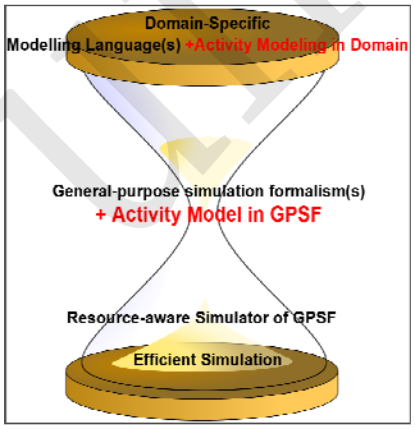

Fig. 7 Activity Modeling: From Modeling Domain to Efficient Simulation

DEVS and transformation from **Activity Meta-Model** in a specific domain to **DEVS**. The former part has been discussed in Sendall and Kozaczynski (2003), Syriani and Vangheluwe (2007), and K. and S. (2003). We discuss the activity model transformation in detail in the next section.

3.4 Activity Model Transformation

As discussed in section 3.2, the **Activity Meta-Model** consists of **Atomic Activity** (**AA**) and **Model Activity** (**MA**). Therefore, activity model transformation is also composed of **AA** transformation and **MA** transformation. It is worth noting 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 **Activity Predictor** transformed from the **MA** in a specific domain. According to the theory of model transformation (Czarnecki and Helsen, 2003) (Hans Vangheluwe, 2004), the transformation rules for **AA** and **MA** are listed respectively. **AA** transformation rules are:

- Find the attributes such as **AtomicTransitionTypes** belonging to **AA** in each class of **Activity Combined Meta-Model**.
- Find the corresponding target atomic **DEVS** models transformed from the original class in the meta-model in a specific domain.
- Add these attributes to the states of corresponding atomic **DEVS** models.
- Map the operation **PredictAtomicActvity**() of **AA** to a sub function in internal transition in the corresponding atomic **DEVS** model.
- Map the operations **OutputAtomicActivity**() and **LastTransitionType**() of **AA** to the sub functions in the λ function in corresponding atomic **DEVS** model.
- Transform the type **TransitionType** to the built-in type in **DEVS** model.

The rules of **MA** transformation are listed below:

- Build an atomic **DEVS** model **Activity Predictor**, the model is the target model of **MA**.
- Copy the attributes in **MA** to the model state in **Activity Predictor**.

- Transform the operation **PredictActivity**() to a sub function in the internal transition.
- Embed the operations **ReflectSubActivity**() and **ReceiveTrackData**() into the external transition.
- Integrate **LastTransitionType**() and **OutputActivity**() into the λ function.

The **Reporting** associations between activity model and sub-components are mapped to the activity connections. These connections between activity ports are used to report atomic activities to **Activity Predictor**.

These rules are generated on the basis of source target relationship. The rules application scope is limited in the models typed by **Activity Combined Meta-Model**. Additionally, activity model transformation also covers the model related intrinsic activity model. The intrinsic model does not evolve with the changes of states. Take Missile model for 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 in the process of transformation. These intrinsic activity prediction algorithms are finally merged together with the attributed related activity sub-functions in the internal transition of **Activity Predictor**.

After activity model transformation, the standard definition of **Activity Predictor** is listed below:

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

X_P={*Sub-Activity*, *Query*, *TrackData*}. S_P=*ActiveSet* ∪ *PassiveSet* ∪ *UnknownSet* ∪ *AllTransitionsMappings*. Y_P={*LastTransitionActivity*, *ActivityPrediction*}. δ_{int}={*PredictActivity*()} Internal activity prediction calculation. δ_{ext}={*ReflectSubActivity*(), *ReceiveTrackData*()} The sub-activities collection or activity prediction calculation use track data. λ={*LastTransitionAtomic*(), *LastTransitionType*(), *OutputActivity*()}. Output results are after activity prediction. t_a: Return 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 **Activity Predictor** represented in **DEVS**. **Activity-Based Simulation** using **DEVS** is proposed to simulate these models in order to improve the simulation performance. It will be discussed in detail in the next section.

4 Activity-based Simulation using DEVS

The principle of **Activity-Based Simulation** is described in the left part of Figure 8. **Models**, **Simulation Environment** and **Resource** are the basic elements in simulation. **Load Balance** module is added as the management part to find the most reasonable partition dynamically at runtime. **Simulation Environment** is used to simulate both the **Models** within **Atomic Activity** and **Activity Predictor**. Therefore **Simulation Environment** is composed of **Original Simulator**, **Resource Predictor** and **Tracker**. **Original Simulator** simulates the **Models** within **Atomic Activity** while **Resource Predictor** simulates the **Activity Predictor**. **Tracker** is a tracking system which tracks the resource usage by transitions of models at runtime.

It is worth noting that the implementation of **Original Simulator** supports parallel simulation so that **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 mainly measured by the indexes.

Based on the principle discussed before, it is necessary to answer four questions in order to implement the **Activity-Based Simulation**. The questions are listed below:

- How to track activity in **Activity-Based Simulation**?
- How to quantify resource usage in order to compute activity?
- How to predict the activity with the support of **Activity Predictor** at runtime?
- 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:

- Resource Usage. The resource that has been occupied by models evolves with physical time.
- Performance. The model activity with physical time, such as the frequency of model transitions.
- Resource Allocation. The current resource allocation status.
- Resource Available. The resource still available can be allocated.

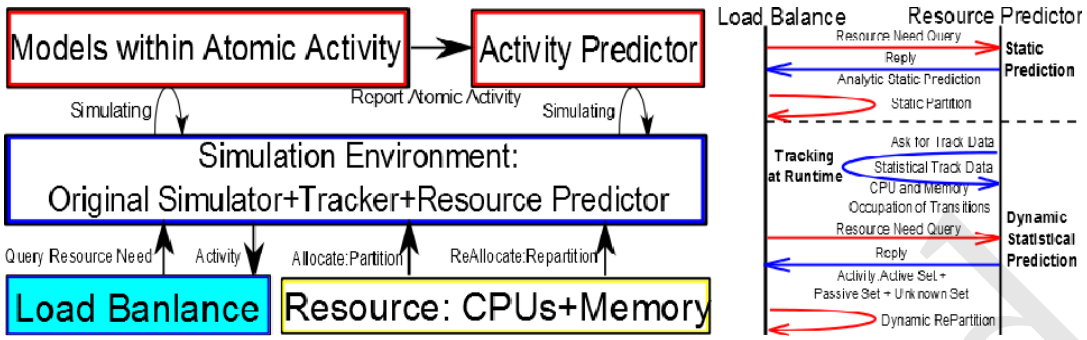

Fig. 8 Activity Tracking

The query and answer mechanism of Activity Tracking is illustrated in the right part of Figure 8. The interactions between **Load Balance** module and **Resource Predictor** are used to obtain the most optimal resource allocation. As mentioned before, **Resource Predictor** is a specific simulator to simulate **Activity Predictor** at runtime. **Resource Predictor** provides two types of predictions: static analytical prediction and statistical prediction. Static analytical prediction comes from the modelers in activity modeling. Take Missile Launch for example; the activity before launch and after exploding stays at zero. Statistical prediction is computed with the help of track data from **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, **Load Balance** module queries the static analytical prediction from **Resource Predictor**. The predictor calculates the whole activity trace in the preparation phase. Then the static activity distribution along the simulation is obtained. **Load Balance** module allocates the resource based on the static analytical prediction replied to by **Resource Predictor**.

At runtime, **Resource Predictor** queries the track data from **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. The static analytical prediction is corrected by the statistical track data during the simulation. When the query from **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 the CPU and the memory, especially the CPU. Therefore, the quantification of resource usage is realized by the tracking of the CPU and memory occupations.

Because the DEVS is chosen to be the target formalism in implementation, the activity model is also realized in a compositional manner. Thus the definition of Equation 4 can be rewritten below:

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

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 atomic **DEVS** models in **Models**. s is a sub state of A_j while S_j is the state space of A_j. $F_\delta^{A_j(s)}(i)$ is the frequency of transition i in A_j in state s. It 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. Firstly, the definition of the transition computation is listed as

$$C_\delta^{A_j(s)}(i) = K_{proc} \int_H^R p_{occ}^{A_j(s)}(i)dt \quad (0 < i < n, 0 < j < m, s \in S_j) \tag{7}$$

i: A transition belongs to A_j. There are n transitions in A_j in total. $A_j(s)$: An atomic **DEVS** model with state s,

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

The precise values of H, R and $p_{occ}^{A_j(s)}(i)$ can be measured at run time. But K_{proc} is not a measurable parameter that can be represented by the performance data such as a million instructions per second (MIPS) (MacNeil, 2004) and the 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 the Equation 7, K_{proc} is calculated as

$$K_{proc} = \frac{C_{\delta}^{A_j(s)}(i)}{\int_H^R p_{occ}^{A_j(s)}(i)dt} \tag{8}$$

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 get the performance coefficient. Moreover, the standardized test models are customized to satisfy the requirements from the models in different categories. The customized models are able to obtain accurate K_{proc} with the consideration of the computation types such as float point operation and integer operation.

In another respect, the 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 defined below:

$$M^{A_j(s)}(i) = \int_H^R m_{occ}^{A_j(s)}(i)dt \tag{9}$$

$m_{occ}^{A_j(s)}$: 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. **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 run time. **Mapping Table** holds and updates the mapping relationships when the resource is allocated or reallocated. It is worth noting that the mapping relationship is organized by the unique activity id $ID_{A_j(i)}^C$. The definition of **Mapping Table** (MT) and id are listed as

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

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

Figure 9 proposes 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 are built for the **Activity-Based Simulation**. Instantiated by the meta-model, the generated model in the domain is composed of the original model and activity model. With the help of model transformation, the model is transformed to **GPSF** (**DEVS**) domain. The model in **GPSF** consists of the original model within atomic activity models represented in **DEVS** and the activity model called **Activity Predictor** with connections with atomic activity models inside the original model. **Activity Predictor** includes the model attributed

related activity model and model related intrinsic activity. As discussed in Section 3.4, modelers don't have to describe the intrinsic activity features in the **Activity Meta-Model** in the specific domain, the transformation collects the instinct information and adds them to the activity model in **GPSF** (**DEVS**).

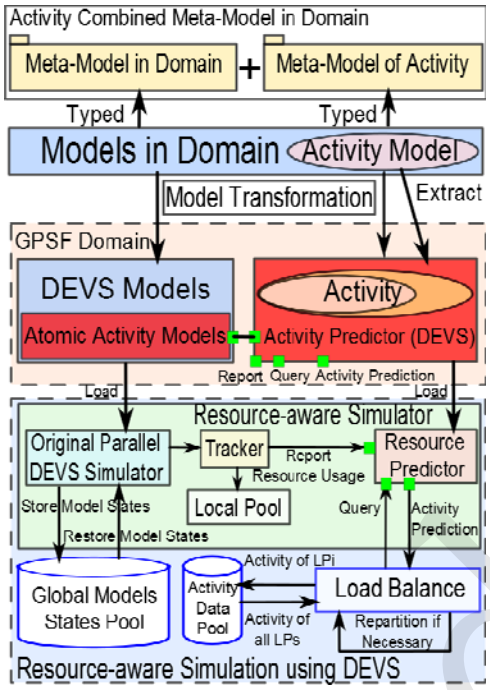

Fig. 9 The Framework of Resource-Aware Simulation

In the **Activity-based Simulation** step, the original parallel **DEVS** simulator is extended to construct the **Resource-Aware Simulator**. **Tracker** module, **Resource Predictor** module and **Local Pool** are added to the new simulator. Original models in **DEVS** are still simulated by the original parallel **DEVS** simulator. Tracker module tracks the CPU and memory occupations (resource usage) by models' computation. The track data are stored in **Local Pool** and reported to **Resource Predictor**. As a specific simulator for **Activity Predictor**, **Resource Predictor** receives the track data, makes the activity prediction and answers the activity query from **Load Balance** module. It is worth indicating that the activity discussed here is the activity of an LP (Local Process) in parallel simulation. **Load Balance** module collects the activity information from all LPs in parallel simulation. Meanwhile, the activity information is stored in **Activity Data Pool** in case of global activity computation for **Load Balance** module. Repartition is made by **Load Balance** module when the conditions of overload are met at runtime. Additionally, **Global Models States Pool** is implemented to store the states for models located at all LPs. 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.

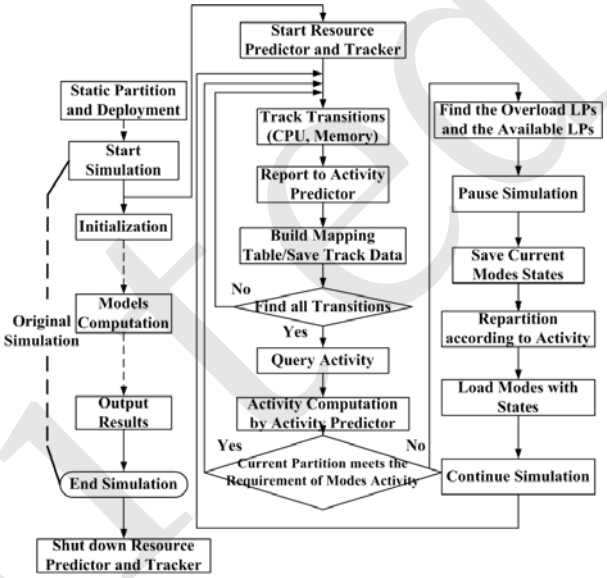

Fig. 10 The Work Process of Activity Predictor

4.4 Resource Reallocation with Activity Prediction

With the help of activity prediction in **Resource Aware Simulation Framework**, **Load Balance** module is able to make resource reallocation(Repartition) during simulation in order to avoid LP overload and improve the performance. **Load Balance** module queries activity predictions from **Activity Predictor**. Activity predictions provide the resource usage of each LP in future. So the resource usage of all LPs can be predicted by **Load Balance** module. Then the module adjusts the resource allocation to find the most optimal partition. The work process is illustrated in detail in Figure 10.

Different from the original simulation shown in the left part of the figure, activity prediction and resource reallocation are coupled into the resource-aware simulation. Just after the starting of simulation, **Resource Predictor** and **Tracker** are activated.

Tracker tracks the resource usage of transitions of models simulated in the original simulation. The track data are reported to **Activity Predictor** simulated by **Resource Predictor**. **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 **Load Balance** module, **Activity Predictor** makes the prediction of the current LP and sends it back. If the current partition cannot meet the requirements of activities received from all the LPs, **Load Balance** module finds the overload and available LPs. Then the resource allocations are triggered. The process of reallocation falls into five steps: Pause the simulation, Save the states of models in **Global Models States Pool**, Repartition the models according to the activity, Restore the state from **Global State Pool**, and Continue the simulation. However, there are two problems still not interpreted by Figure 10: How to do the **Load Balance** and how to find the overload LP in resource-aware simulation. For the former problem, we reuse the **Load Balance** algorithms in Deelman and Szymanski (1998) in the implementation of **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 LP with support of quantification of resource usage.

$$C_{LP_l}^{Ave}(H,R)=\frac{\int_H^R\sum_{i=1}^{n}\sum_{j=1}^{m}F_{\delta}^{A_j(s)}(i)*C_{\delta}^{A_j(s)}(i)dt}{R-H} \quad (11)$$

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

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

LP_l: A LP in parallel simulation; there are k LPs in total. $P(l)$: The partition located in LP_l. $A_j(s)$: A atomic **DEVS** model belongs to $P(l)$ with state s. i: A transition of A_j. H: The input start time of LP_l for computation. R: The input end time of LP_l for computation. Overload means the resource need exceeds the LP limitations. With the help of $C_{LP_l}^{Ave}$ and $M_{LP_l}^{Ave}$, the conditions of LP overload from time H to R are defined below:

$$LP_l^O(H,R)=\{C_{LP_l}^{Ave}(H,R)>>1,\ M_{LP_l}^{Ave}(H,R)>>Mem(LP_l)\} \quad (14)$$

$Mem(LP_l)$: The memory allocated for LP_l.

It means the LP is overloaded when the average CPU occupation is much larger than one hundred percent or the average memory occupation is much larger than the allocated memory. The reallocation is triggered when the LPs are in the conditions mentioned before. It is worth indicating that the reallocation is sometimes time consuming, so 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 decreased.

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**. **March Map** constituted by **Grid**s is usually a typical case to simulate the combat simulation. The composition of **Grid**s 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 **Platoon**s in different colors, the fight happens. In the modeling view, **March Map** is modeled in the spatial-oriented representation. **Grid** is an atomic model that cannot be decomposed again. The **Grid**s are spatially linked together to construct the **March Map**. **Platoon**s are modeled as the events sent and received by **Grid**s.

5.2 Activity Model of March Map

Obviously, **March Map** is a typical spatial compositional model. So the quantified definition of activity model in Equation 6 can be improved with the help of **Activity Region** definition in Equation 5. The definition for the activity model of **March Map** is listed as

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

where

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

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

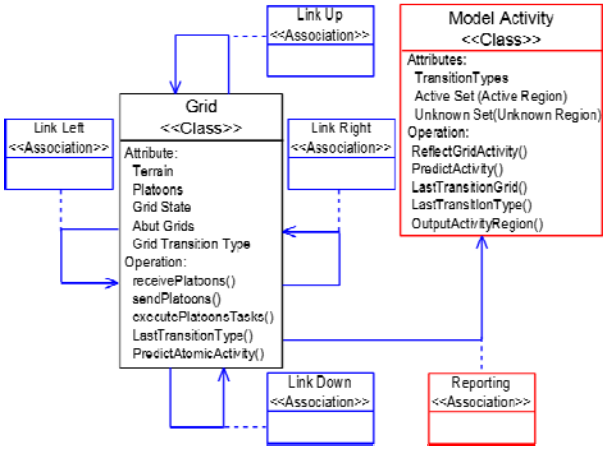

Fig. 11 the Activity Combined Meta-Model

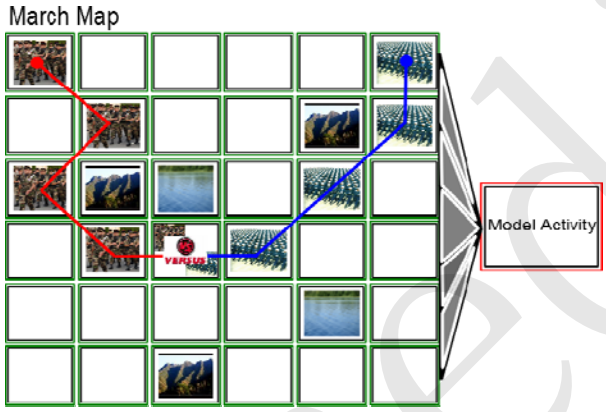

Fig. 12 the Model of March Map

According to the description of **March Map**, a **Grid** can be linked with other **Grid**s in four directions. The states of **Grid** are composed of Terrain, Platoons, Platoon State and Abut Grids. It is easy to build the meta-model of **March Map** with these attributes. In the view of **Activity Enhanced Modeling**, the **Grid**s with **Platoon**s inside are active because of the computation for Platoon tasks. So the transitions of **Grid** are the basic elements in **Atomic Activity** for the activity model. Based on the merge operations discussed in section 3.2, activity combined meta-model of **March Map** is devised as Figure 11. The attributes are listed in Table 1. The complementary 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, CurState. The model of **March Map** is instantiated by the meta-model as shown in Figure 12. The red and blue **Platoon**s move on the 6*6 **March Map**, they happen to meet at some **Grid** and the fighting is triggered.

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

5.3 DEVS Representation of March Map

Based on the **Activity Enhanced Modeling**, models with activity model have to be transformed into the **GPSF**. Therefore the models of **March Map** in a specific domain are transformed into **DEVS** before simulation. According to the rules of **Model Transformation**, **March Map** represented in **DEVS** is shown as

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

X_G={*Platoons*}. S_G={*Terrain*, *Platoons*, *AbutGrids*, *ActiveState*, *GridTransitionTypes*}. Y_G={*Platoons*, *NextGrid*, *LastTransitionType*}.

δ_{int}={*ExecutePltoonsTasks*()} Internal **Grid** function.

δ_{ext}={*GetLeavingPlatoons*(), *LastTransitionType*(), *GetActiveState*()} External **Grid** sub functions including the leaving platoons and atomic activity information. λ={*SendPlatoons*(), *SendLastTransitionType*(), *SendActiveState*(), *SendNextGrid()*}

Send **Platoon**s to other **Grid**, Output activity information to **Activity Predictor**. t_a: Return the time stamp for the next prediction. The atomic activity model is imbedded in **Grid** model. **ActiveState** depends on the attributes **Terrain** and **Platoon Task**. **GridTransitionTypes** are decided by the **Platoon Task** and the activity prediction is made by **NextGrid** calculated by the moving directions of **Platoon**. These value sets of the **Grid** states are list 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 **Grid**s. So the atomic activity prediction outputs both the current and next active state of **Grid**. Moreover, **Activity Predictor** is also obtained from the **Model Transformation**, the model is listed below:

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

X_P={*GridTransitionTypes*, *Query*, *TrackData*, *NextGrids*}. S_P=*ActiveSet* ∪ *PassiveSet* ∪ *UnknownSet* ∪ *AllTransitionsMappings*.

Y_P={*LastTransitionActivity*, *ActivityPrediction*}.

δ_{int}={*MakeTransitionMappings*(), *PredictActivity*()} Internal activity prediction calculation.

δ_{ext}={*ReflectGridTransitions*(), *RecieveNextGrids*(), *ReceiveGridActiveState*(), *ReceiveTrackData*()}.

Eternal activity functions include the sub-activities collection or activity prediction calculation using track data. λ={*OutputActivity*()} Output results after activity prediction. t_a: Return the time stamp for the next prediction. **Activity Predictor** collects the activity from all the **Grid**s via the **Reporting** association. **Activity Region** including **ActiveGridSet**, **PassiveGridSet** and **UnknownGridSet** are generated by assembling the separated **Grid** activity. **ActiveGridSet** is composed by the **Grid**s within **Platoon**. **PassiveGridSet** consists of the **Grid**s where the terrain is mountain or river. The remaining **Grid**s compose the **UnknownGridSet**. The **Activity Region** is changing by the movements of **Platoon**s during the simulation. **AllTransitionsMappings** which are implemented by **Mapping Table are** built by the **GridTransitionTypes** and **TrackData** from **Tracker** mentioned before. When the **Query** from **Load Balance** module is received, **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 cores Intel i5-2300 CPU @2.8 GHz and 4GB RAM. The models of **March Map** represented in **DEVS** are programmed in C++ in an object oriented fashion. The **DEVS** simulator of tool OneModel (Guo, 2013) and (Boukerche and Das, 1997) is used as the **Original Parallel DEVS Simulator**. The communications between simulators in different LPs use the MPI message passing library. Tracker, **Resource Predictor** and **Load Balance** module are also implemented in C++. **Local Pool**, **Track Data Pool** and **Global Models States Pool** are all implemented by MySQL (WELLING and THOMSON, 2005).

As discussed in Section 4.4, **Mapping Table** is built in simulation to store the track data for **Activity Predictor**. The table is shown below within the track data of transitions in **Grid**s.

$$\begin{bmatrix} G_{(1,1)}\_Move\_Road & (0.014,232K) \\ G_{(1,1)}\_Rest\_Road & (0.009,240K) \\ G_{(1,1)}\_Move\_Field & (0.023,241K) \\ G_{(1,1)}\_Rest\_Field & (0.008,220K) \\ G_{(2,2)}\_Move\_Road & (0.001,217K) \\ G_{(2,2)}\_Rest\_Road & (0.003,249K) \\ \cdots & \cdots \end{bmatrix} \quad (16)$$

The left column of **Mapping Table** lists transition IDs which are composed according to Equation 10. The right column is 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 by the average of CPU occupations. Base on Equation 7, the computation of transitions are quantified below:

$$C_{\delta}^{G_{(x,y)}}(i) = K_{proc} * \frac{1}{n_{pas}^{G_{(x,y)}}(i)} \sum_{r=1}^{n_{pas}^{G_{(x,y)}}(i)} p_{OCC}^{G_{(x,y)}}(i_{(r)}) \quad (17)$$

$G_{(x,y)}$ is a **Grid** in **March Map**, (x,y) is the location of the **Grid** while m is the size of the **Grid**s. i is a transition of **Grid** such as *Move_Road*. $C_{\delta}^{G_{(x,y)}}(i)$ is the computation usage by transition i. $p_{OCC}^{G_{(x,y)}}(i_{(r)})$ is the rth ($1 \le r \le n_{pas}^{G_{(x,y)}}(i)$) 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. $n_{pas}^{G_{(x,y)}}(i)$ is the number of transition i which has been executed by $G_{(x,y)}$. $p_{OCC}^{G_{(x,y)}}(i_{(r)})$ and $n_{pas}^{G_{(x,y)}}(i)$ are restored in the **Track Data Pool** mentioned before. They are tracked to predict the computation for the transition i in model $G_{(x,y)}$.

Figure 13 and Figure 14 present two snapshots of activity prediction and partition during simulation. In

Figure 13, both **Platoon**s in red and blue just finished the first step in their initial **Grid**s. They will be sent to the next **Grid**s in the next step. Based on the current states of **Grid**s and activity model, the **ActiveSet**, **PassiveSet** and **UnknownSet** are generated by activity prediction. As shown in the Figure, $G(2,2)$ and $G(6,2)$in red compose the **ActiveSet**. Meanwhile, the possible **Grid**s for **Platoon**s are filled by green belonging to the **UnknownSet**. The rest **Grid**s filled by grey compose the **PassiveSet**. The partition is still in its initial state in which the 36 **Grid**s are distributed evenly to four LPs. With the help of Equations 11, 12 and 13, the activity of **Grid**s can be predicted as

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

$$M_{LP_l}^{Ave}(H,R)=\frac{1}{\Delta t}\sum_{G_{(x,y)}\in P(l)}\sum_{i=1}^{n}m_{occ}^{G_{(x,y)}}(i)*\Delta t \tag{19}$$

$C_{\delta}^{G_{(x,y)}}(i)$ is the computation by transition i of $G_{(x,y)}$, the value is computed by Equation 17. $F_{\delta}^{G_{(x,y)}}(i)$ is the transition frequency predicted for i of $G_{(x,y)}$. The frequency is predicted by internal activity prediction function in **Activity Predictor**. In our case, this information is acquired with the help of current locations and simulation scenario. The **Platoon**s follow the assigned tasks in 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 **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 another words, the time factor in the computation of LP is seen as an average value for transitions. As a result, the computation and memory occupation of LPs are listed below:

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

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

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

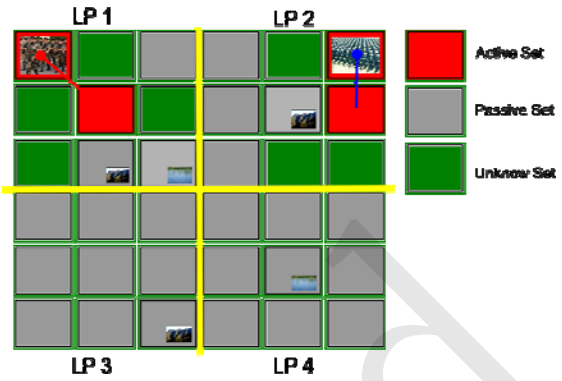

Fig. 13 The Activity Prediction and Partition of First Step

Fig. 14 The Activity Prediction and Partition of Second Step

So the predictions of the computation of LPs are listed below:

$$C_{LP_1}^{Ave}(\Delta t)>>C_{LP_3}^{Ave}(\Delta t)=0$$

$$C_{LP_2}^{Ave}(\Delta t)>>C_{LP_4}^{Ave}(\Delta t)=0$$

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. Figure 14 gives the activity prediction and new partition of the next step. With the help of move directions of **Platoon**s, **ActiveSet**, **PassiveSet** and **UnknownSet** are updated by the activity model. According to the activity information, **Load Balance** module reallocates the resource, the new partition ignores the **Grid**s in the bottom which will not be activated in the near future. Now the number of **Grid**s located in each LP is six. 24 **Grid**s are maintained in the simulation with four LPs. According to Equation 13, the new partition is listed below:

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

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

In addition, Figure 15 presents the activity prediction supported by analytic prediction. Usually, the dynamic activity predictions are limited inside the moving windows because of the non-deterministic and randomness of models when the simulation is being made. But in our case, 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 **Load Balance** module to find the optimal partition shown in Figure 15.

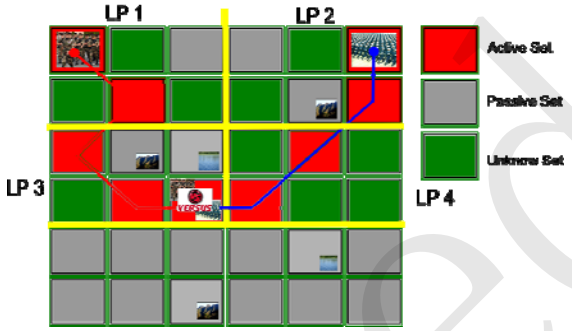

Fig. 15 the Partition with Analytic Activity Prediction

5.5 Results Analysis

Figure 16 shows the experimental results of the March Map. As shown by the curve, activity combined simulation is more efficient in the comparison with the original simulation. With the increase in Grids 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 Grids 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.

From another aspect, the curve in Figure 17 illustrates the accuracy of the transition computation. The **Grid** transitions are categorized into several levels such as *Move_Road* and *Fight_Field* list in Table 1. Because the models of **Grid**s are all isomorphic, the transitions are summarized together in order to show the obvious change in experiment results. The computation of transition *Move_Road* of **Grid**s is predicted in the equation below:

$$C_{\delta}^{Grids}(i_m,T)=\sum_{x=1}^{m}\sum_{y=1}^{m}F_{\delta}^{G_{(x,y)}}(i_m,T)*C_{\delta}^{G_{(x,y)}}(i_m,T) \quad (20)$$

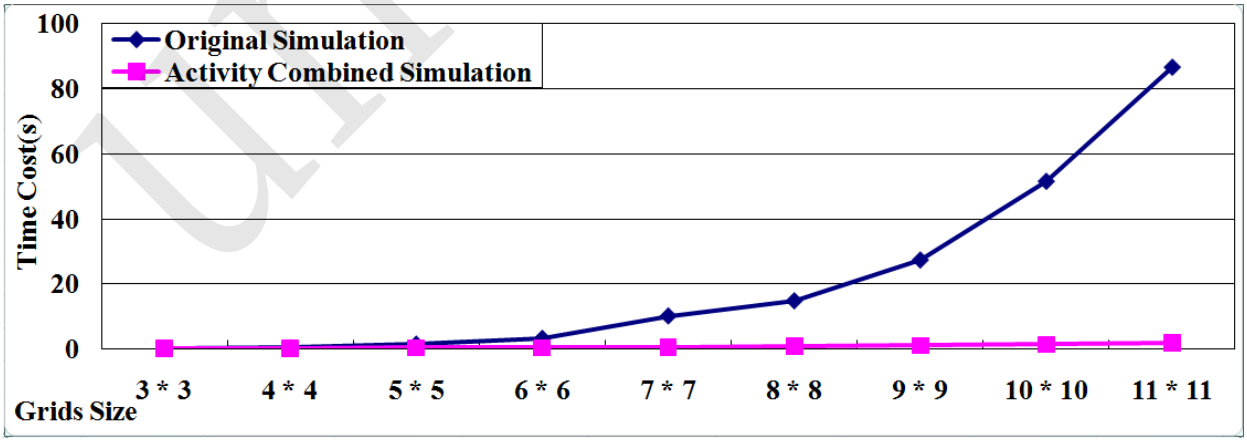

Fig. 16 the Performance of the Activity Combined Simulation

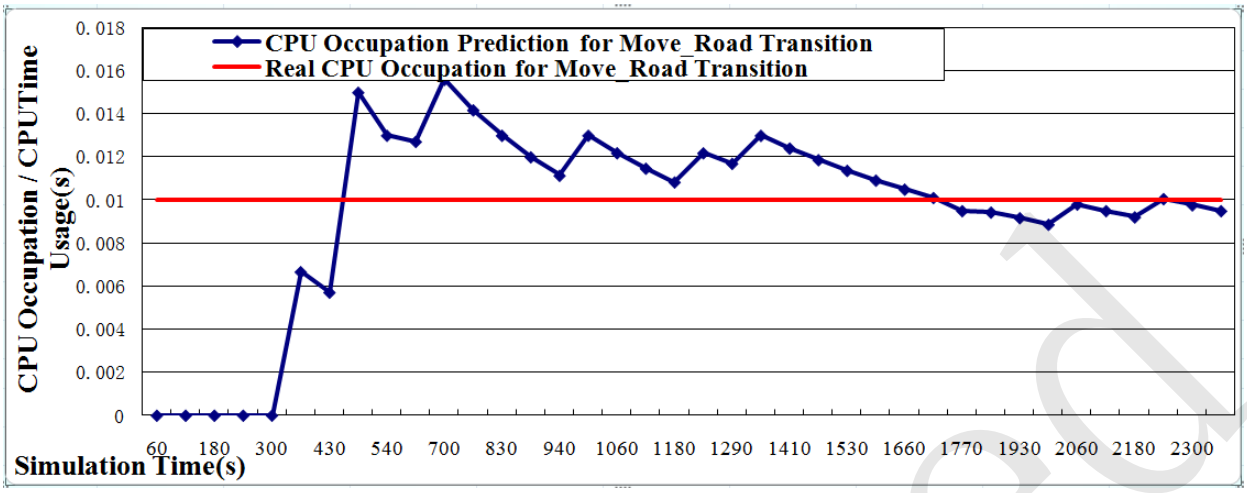

Fig. 17 the prediction of computation of *Move_Road* Transitions

T is the simulation time. i_m is the transition *Move_Road*. $C_{\delta}^{Grids}(i_m,T)$ is the prediction of integral computation by transition *Move_Road* of the whole **Grid** models at simulation time T. $C_{\delta}^{G_{(x,y)}}(i_m,T)$ is the computatio n by transition *Move_Road* of $G_{(x,y)}$ computed by Equation 17 at simulation time T. $F_{\delta}^{G_{(x,y)}}(i_m,T)$ is the frequency of transition *Move_Road* in $G_{(x,y)}$. The frequency is predicted by internal activity prediction function in **Activity Predictor** at simulation time T.

Based on Equation 20, the experiments show how to obtain the accurate transition computation. The straight red line is the real computation 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. The blue curve shows the predicted computation by *Move_Road* during the simulation.

The curve indicates that the prediction at the beginning stays at the zero because of the inaccuracy of measurement (The tracked CPU occupation of *Move_Road* approximates to the zero). Thanks to the accumulation of the track data, the prediction approaches the accurate value along the simulation. According to the figure, the predicted value oscillates closely to the datum line after the 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 Activity Region. The region focuses on the locations of 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 the 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**, the meta-model in a specific domain will be transformed to the **GPSF** representation. The transformation not only transforms the attributed 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 listed below:

- **Activity Enhanced Modeling** gives a method to define the activity model in a specific domain.
- Domain specific activity modeling maintains the advantages in minimizing the modeling accidence.
- The quantization of activity and resources make possible the application of the activity model in the simulation.
- **Resource-aware Simulation Framework** implements the activity-based simulation, the framework integrates the activity in both modeling and simulation aspects.

In 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 **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, Simonetta, Di Marco, Antinisca, Inverardi, Paola, Simeoni, Marta, 2004. Model-based performance prediction in software development: a survey. IEEE Trans. Softw. Eng., 30(5):295-310.

Boukerche, A., Das, S.K., 1997. Dynamic load balancing strategies for conservative parallel simulations. Proceedings 11th Workshop on Parallel and Distributed Simulation, p.20-28.

Concepcion, A.I., Zeigler, B.P., 1988. DEVS formalism: a framework for hierarchical model development. IEEE Transactions on Software Engineering, 14(2):228-241.

Czarnecki, Krzysztof, Helsen, Simon, 2003. Classification of model transformation approaches. OOPSLA03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.

D'Abreu, M.C., Wainer, G.A., 2005. M/cd++: modeling continuous systems using Modelica and DEVS. 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, p.229-236.

Deelman, E., Szymanski, B.K., 1998. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on, p.46-53.

Emerson, Matthew, Sztipanovits, Janos, 2006. Techniques for MetaModel composition. 6th OOPSLA Workshop on Domain-Specific Modeling, OOPSLA, p.123-139.

National University of Defense Technology, 2013. User manual of onemodel.

Hans Vangheluwe, Juan de Lara, 2004. Computer automated multi-paradigm modelling for analysis and design of traffic networks. Proceedings Simulation Conference, 1:249-258.

Hu, Xiaolin, Muzy, Alexandre, Ntaimo, Lewis, 2005. A hybrid agent-cellular space modeling approach for fire spread and suppression simulation. WSC '05: Proceedings of the 37th conference on Winter simulation, p.248-255.

Hu, Xiaolin, Zeigler, Bernard P, 2013. Linking information and energy-activity-based energy-aware information processing. Simulation, 89(4):435-450.

Jammalamadaka, Rajanikanth, 2003. Activity characterization of spatial models: Application to the discrete event solution of partial differential equations. University of Arizona, Tucson, Ariz.

Lagerström, Robert, Chenine, Moustafa, Johnson, Pontus, Franke, Ulrik, 2008. Probabilistic MetaModel merging. CAiSE Forum, p.25-28.

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

Muzy, A., Nutaro, J.J., Zeigler, B.P., Coquillard, P., 2008. Modeling and simulation of fire spreading through the activity tracking paradigm. Ecological Modelling, 219(1-2):212 - 225.

Muzy, A., Zeigler, B.P., 2008. Introduction to the activity tracking paradigm in component-based simulation. The Open Cybernetics and Systemics Journal, 2:48-56.

Muzy, Alexandre, Jammalamadaka, Rajanikanth, Ziegler, Bernard P, Nutaro, James J, 2010a. The activity-tracking paradigm in discrete-event modeling and simulation: the case of spatially continuous distributed systems. Simulation, 87(5):449-464.

Muzy, Alexandre, Touraille, Luc, Vangheluwe, Hans, Michel, Olivier, Traoré, Mamadou Kaba, Hill, David R. C., 2010b. Activity regions for the specification of discrete event systems. Proceedings of the 2010 Spring Simulation Multiconference, p.136:1-136:7.

Petriu, Dorina C., Shen, Hui, 2002. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications, p.159-177.

Qiu, Fasheng, Hu, Xiaolin, 2013. Spatial activity-based modeling for pedestrian crowd simulation. Simulation, 89(4):451-465.

Sendall, S., Kozaczynski, W., 2003. Model transformation: the heart and soul of model-driven software development. IEEE Software, 20(5):42-45.

Shibata, Danilo, Alfenas, Daniel, Guiraldelli, Ricardo, Pereira-Barretto, Marcos R., Marcellino, Fernando, 2012. Activity based scheduling simulator for product transport using pipeline networks. Proceedings Simulation Conference, p.1-12.

Steven Kelly, Juha-Pekka Tolvanen, 2008. Domain-Specific

Modeling: enabling full code generation. Wiley-IEEE Computer Society Press, p.448.

Syriani, Eugene, Vangheluwe, Hans, 2007. Programmed graph rewriting with DEVS. Applications of Graph Transformations with Industrial Relevance, p.136-151.

Vangheluwe, H.L.M., 2000. DEVS as a common denominator for multi-formalism hybrid systems modelling. IEEE International Symposium on Computer-Aided Control System Design, p.129-134.

WELLING, L.A., THOMSON, L.A., 2005. Php and mysql web development. Sams Publishing.

Xiaolin Hu, Lewis Ntaimo, 2006. Dynamic multi-resolution cellular space modeling for forest fire simulation. Proceedings of the DEVS Integrative M&S Symposium (DEVS'06), Spring Simulation Multiconference, p.95-102.

Xiaolin Hu, Lewis Ntaimo, 2008. Devs-fire: towards an integrated simulation environment for surface wildfire spread and containment. Simulation, 84(4):137-155.

Zeigler, Bernard P., Jammalamadaka, Rajanikanth, Akerkar, Salil R., 2004. Continuity and change (activity) are fundamentally related in DEVS simulation of continuous systems. AIS, p.1-13.

Zeigler, P Bernard, Herbert, Praehofer, Gon, Kim Tag, 2000. Theory of Modeling and Simulation, 2nd Ed. Academic Press.