ALGORITHM FOR DEVS STRUCTURE CHANGES

Lassaad BAATI, Claudia FRYDMAN, Norbert GIAMBIASI

Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20, France
lassaad.baati@lsis.org (*Lassaad Baati*)

Abstract

Several dynamic complex systems are too complex to be represented through only one modelling formalism (Petri nets, bond Graph, etc.) like military systems. Vangheluwe analyses some wide utilized formalisms and presents DEVS (a mathematically sound framework) as a common denominator for multi-formalism hybrid systems modelling. The main objection against the use of DEVS (and derived formalisms) in agent based simulation was their static nature. We proposed a dynamical hierarchical structure modelling approach. It preserves the DEVS formal model in order to take advantage of its experience and its demonstrated capabilities (closure under coupling, hierarchy, modularity, etc.), and propose a dichotomy between the structure and the behavioural model. This paper proposes a collapsed view of our approach focused on the structure changes boundaries described through algorithms.

Keywords: Discrete event modelling, DEVS, Variable/Dynamic structure, hierarchical structure.

Presenting Author's biography

LASSAAD BAATI is a PhD student in computer sciences at the Paul Cézanne University of Marseille. He obtained his MS in computer sciences, Modelling and Simulation in the same university. He worked for four years as a software engineer

1 Introduction

Modelling and simulation becomes broadly utilised to resolve dynamical and complex systems representation in order to analyse and diagnose their behaviour. Obviously, dynamical and complex systems have a wide range of specific features, and experts usually use adapted formalism to model an appropriate system. Thus, each of modelling formalism was used for a specific domain [1].

Several dynamic complex systems are too complex to be represented through only one modelling formalism (Petri nets, bond Graph, etc.) like military systems [2]. Efforts were made to propose multimodel utilization [3], which supports several formalisms utilization to model only one complex system. On the other hand, based on its hierarchical structure and closure under coupling features, Vangheluwe analyses some widely utilized formalisms and presents DEVS (a mathematically sound framework) as a common denominator for multi-formalism hybrid systems modelling [1].

The main objection against the use of DEVS (and derived formalisms) in agent based simulation was their static nature [4]. Dynamic structure becomes a challenging problem [5]; indeed dynamic system behaviour evolution goes beyond a limited behaviour changes to deeply changes related to its own structure. These structure changes become crucial in several dynamic and complex systems to be adapted in different situations and be optimal in simulation performance [5]. Several researches were done, to resolve dynamic structure modelling and simulation issues and inquiries. We noted that proposed solutions in the literature can be divided in two groups; the first one tries to resolve the structure variability by extending the DEVS formalism, in the way that they add parameters or functions to the predefined formal DEVS model. The second one preserves the DEVS formalism, and includes the structure behaviour or parameters into the behavioural DEVS model. We proposed in [6] a dynamic hierarchical structure modelling approach in the middle of these two groups. This approach preserves the DEVS formal model in order to take advantage of its experience and its demonstrated capabilities (closure under coupling, hierarchy, modularity, etc.), and proposes a dichotomy between the structure model and the behavioural model [7], which enhances modularity and reusability. This paper proposes a collapsed view of our approach focused on the structure changes boundaries described through algorithms.

This paper presents a DEVS review in section 2. Section 3 describes some related works. Section 4 exposes the dynamic hierarchical structure modelling approach based on DEVS. Section 5 explains boundaries of this approach and depicts some algorithms of changing structure functions. Finally, we conclude in section 6 and expose our future work.

2 DEVS review

DEVS [8] is a modular formalism for deterministic and causal systems' modelling. A DEVS atomic model has a continuous time base, inputs, states, outputs and functions (output, transition, lifetime of states). Larger models are built from atomic models connected together in a hierarchical fashion. Interactions are mediated through input and output ports. That allows for modularity. We propose below a related approach that supports variable structure model, and preserves the DEVS formalism.

2.1 Formal Specification of an Atomic DEVS Model

$$AtomicDEVS = \langle X, Y, S, \delta_{\text{int}}, \delta_{ext}, \lambda, lt \rangle$$

- The time base is continuous and not explicitly mentioned: T= IR.
- X is the set of (external) inputs of the model. They interrupt its autonomous behaviour by the activation of the external transition function δ_{ext}.
- Y is the set of outputs.
- S represents the set of sequential states.
- δ_{int} is internal transition function, allowing the system to go from one state to another autonomously.
- λ is the output function.
- lt(s) is the lifetime function.

The system's reaction to an external event depends on its current state, the input value and the elapsed time. Fig. 1 explains internal and external transition processes.

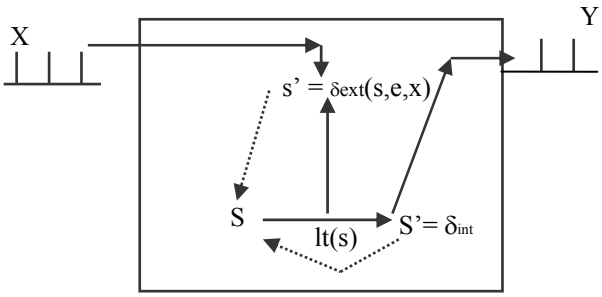

Fig. 1 Dynamics of a DEVS model

Fig. 1 represents detailed performed tasks during atomic model simulation, they are exposed as follow:
The model is initially in State S
If no external event occurs, it will stay in S for time period given by lt(S) which is the lifetime function, defining the lifetime of the states
After lt(S) time, ie elapsed time=lt(S), system outputs λ(S) and goes to the next state defined by the internal transition function δint.
If an external event X occurs, the new state is determined by input X, the current state S, and elapsed time e. (δext(s,e,x)).

2.2 Formal Specification of A Coupled DEVS Model

The coupled DEVS formalism describes a discrete events system in terms of a network of coupled components.

$$CoupledDEVS = \langle X_{self}, Y_{self}, D, \{M_d / d \in D\}, EIC, EOC, IC \rangle$$

Self stands for the model itself.
- X*self* is the set of possible inputs of the coupled model.
- Y*self* is the set of possible outputs of the coupled model.
- D is a set of names associated to the model's components, self is not in D.
- $\{M_d / d \in D\}$ is the set of the coupled model's components, with d being in D. These components are either atomic or coupled DEVS model.

EIC, EOC and IC define the coupling structure in the coupled system.
- EIC is the set of external input coupling, which connects the inputs of a coupled model to components inputs.
- EOC is the set of external output coupling, which connects the outputs of a coupled model to components outputs.
- IC defines the internal coupling, transforming a component's output into another component's input within the coupled model.

3 Related work

Basic DEVS formalism provides a static structure. To resolve dynamic structure modelling issues, many approaches were proposed. This section provides an overview of related work based or close to DEVS formalism.

Kyou H. Lee & al proposed Variable Structure System Specification (VSSS) formalism. It is composed of a VSSS coordinator, named compositeVSSS, and a mapped-model, named AtomicVSSS. The compositeVSSS employs inputs, outputs, submodels and state variables, state variable transition function, and a mapping function. The atomicVSSS model employs inputs, outputs, states, and output and transition functions. The mapping function supervises the compositeVSSS structure by activating dynamically submodels [9]. VSSS is closed on DEVS.

For ecological system, A. Uhrmacher & al proposed the dynamic DEVS formalism [10]. This formalism adds two functions to the classical DEVS formalism in order to allow structure changes. The first one is the "model transition function" which is included in the dynamic atomic model. The second one is the "network transition function", which is included in a dynamic network/coupled model. Their duty is changing autonomously their structure without any controllers. Recently, she proposed *p*-DEVS [4], which is the recent DEVS extension to resolve the structure variability issues. It allows input and output ports alteration dynamically during simulation.

T. Pawletta has proposed in [11] a DEVS based approach for modelling and simulation of hybrid variable model. He creates a coupled variable structure model N_{dyn}, according to coupled model in the classic DEVS. N_{dyn} is a coupled model with a specific composite state variable H_N. This variable includes state variables that would be able to change the model structure.

In the domain of adaptive computer architecture, F. Barros introduces a variable structure modelling formalism V-DEVS [12]. Then, the "Dynamic Structure DEVS" formalism named DSDEVS [13]. This formalism is an extension of DEVS; it provides an executive model within a dynamic structure coupled model. The executive is a modified DEVS model, it includes a structure transition γ and a set of structures Σ*. Thus he builds parallel Dynamic Structure DEVS formalism named DSDE [14] [15], which adds inputs X_N and outputs Y_N into the network model, and performs parallel simulation.

Xiaolin Hu introduced in [16] a variable structure modelling approach based on the DEVS formalism. He specified that a model can change the structure of another one through internal or external transition functions [17]. Otherwise, the structure behaviour is included in the behavioural model. Thus, the DEVS based model as a whole, can change autonomously its structure through transitions.

We note that these proposed approaches alter the DEVS formalism by adding functions or specific variables to manage structure variability. Xiolin Hu preserved the DEVS formalism but, includes structure variability into the behavioural models, thus decreasing reusability.

In [6] we presented the "Dynamic Hierarchical Structure DEVS modelling approach" based on the DEVS formalism and allowing dynamic structure modelling through DEVS components (structure models) as controllers. We precise that we preserved the DEVS formalism and its properties, like hierarchy, modularity, and closure under coupling. Then we proposed in [7] an algorithm implementing this approach based on the abstract simulator introduced by Zeigler [8].

4 Dynamic hierarchical structure DEVS modelling approach

In [6] we proposed the dynamic hierarchical structure modelling approach. It provides a model separation between dynamic behaviour and dynamic structure. And it preserves the DEVS formalism properties. Thus it can be adopted by a wide range of a DEVS tools easily. Below, we present this approach, its formal representation, and its corresponding algorithm based on the abstract simulator [8].

4.1 Dynamic Approach

This approach is based on the DEVS formalism to manage structure. It allows structure variability without altering the DEVS formalism properties. We propose a structure model which is an atomic DEVS model to manage the structure variability of another coupled DEVS model. Every change in the state of the structure model can imply a structure change of the managed coupled DEVS model. Being out of the managed coupled model, the structure model has larger capabilities of structure changes. [7]

4.2 Formal Representation

As described in [6] and [7], this approach preserves the DEVS formal representation for simulation proposed by Zeigler in [8]. We include structure variability information into the set of state variables of the structure model. We describe below the related formal representation [6].

The structure formal model: $Z = \langle X_Z, Y_Z, S_Z, \delta_{ext}, \delta_{\text{int}}, \lambda_Z, lt_Z \rangle$

and $Mc = \langle X_i, Y_i, D_i, M_i, EIC_i, EOC_i, IC_i, F_i \rangle$ is the managed behavioural coupled model, which evolves through several structures permitted by *Z*.

X_Z is the set of input values of the structure model.

Y_Z is the set of output values.

$S_Z = (\theta, \{st_i\})$ is the set of states.

$\delta_{ext} : Q \times X_Z \rightarrow S_Z$ is the external transition function.

$\delta_{\text{int}} : S_Z \rightarrow S_Z$ is the internal transition function.

$\lambda_Z : S_Z \rightarrow Y_Z$ is the output function.

$lt_Z : S_Z \rightarrow R_0^+ \cup \{\infty\}$ is the model state lifetime function.

$X_Z = \{(p,v) / p \in IPorts, v \in Vx_p\}$

Vx_p is a set of values of inputs in the port *p*.

X_Z is the set of couples, input ports and values. These inputs are restricted to events that allow structure changes of Mc (Structure model).

$Y_Z = \{(p,v) / p \in OPorts, v \in Vy_p\}$

Vy_p is a set of values of outputs in the port *p*.

Y_Z is the set of couples, output ports and values that returns the evolution steps of the structure changes.

$S_Z = (\theta, \{st_i\})$ is the set of state variables.

θ is the set of variable states that belong to the structure model.

st_i is the set of parameters that describe the structure of the structure model coupled model Mc. i refers to the structure related to the structure model state S_Z. Each structure is considered as a new model.

X_i is the set of inputs of the structure i.

Y_i is the set of output values of the structure i.

D_i is the set of names of submodels of the structure i.

$EIC_i \in (st_i.IN \times M_d.IN) \quad / d \in D_i$ is the set of external input coupling. Each one connects input port ($st_i.IN$) of the managed coupled model Mc in its sti structure to component input ports ($M_d.IN$). With d the name of a component in the Mc model in the st_i structure.

$EIC_i = \{((st_i, a),(d,b)) / d \in D, a \in IPorts_i, b \in IPorts_d\}$

$EOC_i \in (st_i.OUT \times M_d.OUT) \quad / d \in D_i$ is the set of external output coupling. Each one connects output port ($st_i.OUT$) of the managed coupled model Mc in its st_i structure to component output ports ($M_d.OUT$). With d the name of a component in the Mc model in the st_i structure.

$EOC_i = \{((st_i, a),(d,b)) / d \in D_i, a \in OPorts_i, \ b \in OPorts_d\}$

$IC_i \in (d.OUT \times d'.IN) \quad with\ d \neq d'$ is the set of internal coupling. These are the connections between submodels ports within the managed behavioural coupled model Mc.

$IC_i = \{((d,a),(d',b)) / d \neq d', \ a \in OPorts_d, b \in IPorts_{d'}\}$

$F_i = \sum f_c$ is a set of coupling functions of the structure *i*. f_c is a coupling function related to c (coupling relation), with c in {*EIC* U *EOC* U *IC*}. Each coupling relation has its own coupling function.

$\delta_{ext} : Q \times X_Z \rightarrow S_Z$ is the external transition function. It is triggered by an external event and it allows structure changes of the managed coupled model by providing new structure model state.

Where $Q = \{(s,e) / s \in S_Z \ and \ 0 \leq e \leq lt_Z(s)\}$ with e is the elapsed time in the state s: Q is the set of totally state.

$\delta_{ext}(s_m, e, X_m) = s_n \ / \ s_m, s_n \in S_Z \quad et \quad m \neq n$

$\Rightarrow \quad s_m = (\theta_m, st_m)$

$\delta_{ext}((\theta_m, st_m), e, X_m) = (\theta_n, st_n)$

S_m is the current state. It includes the set of state variables directly related to atomic structure model, and those related to structure behaviour coupled model (st_m).

θ_m, θ_n are sets of state variables in structure model other than st_m, st_n

e is the elapsed time in sm state.

$st_m = \{X_m, Y_m, D_m, \{M_d / d \in D_m\}, EIC_m, EOC_m, IC_m, F_m\}$

X_m is an input value introduced with an external event.

$\delta_{\text{int}} : S_Z \rightarrow S_Z$ is the internal transition function. It is executed after spending lt(s) in the same state without receiving any external event. In some cases, spending a time in the same state can trigger a structure change of the system.

$\delta_{\text{int}}(s_m) = s_n \ / \ s_m, s_n \in S_Z$

$\delta_{\text{int}}((\theta_m, st_m), lt_Z(s_m)) = (\theta_n, st_n)$

S_m is the current state. It includes the set of state variables (θm) related to the atomic structure model, and those related to structure behavioural coupled model (stm).

$st_m = \{X_m, Y_m, D_m, \{M_d / d \in D_m\}, EIC_m, EOC_m, IC_m, F_m\}$

θ_m, θ_n are sets of state variables in the atomic structure model other than st_m, st_n

lt_Z is the lifetime function of the model states.

$\lambda_Z : S_Z \rightarrow Y_Z$ is the output function.

$\lambda(s_m) = Y_m$

Y_n allows coupling relation with other models.

$lt_Z : S_Z \to R_0^+ \cup \{\infty\}$ is the lifetime function of the model states.

F_m is a set of coupling functions. There is a specific coupling function related to each coupling relation.

4.3 Dynamic Hierarchical Structure Approach algorithm

Zeigler introduced in [8] the abstract simulator to define the simulation semantics of DEVS models. The benefit of this concept is the dichotomy between the models and the simulator. In the abstract simulator, each component (processor) corresponds to a model component. The related algorithms were described in [8]. The abstract simulator includes a root coordinator which corresponds to the top level and manages the clock of the whole simulation, a coordinator which coordinates the interaction between models in the same level, child and parent, finally a simulator which performs a simulation of an atomic DEVS model.

Based on the abstract simulator concept, we propose in [7] a mapping of a dynamic hierarchical structure model. We only add a "structure coordinator" to the classic DEVS abstract simulator. All remaining components are the same. This new component extends the coordinator activities to manage structure changes in the child coordinator. The simulator corresponding to the structure atomic model provides an output message "`Ymessage(y,t)`" that includes the structure changes to perform, to the "structure coordinator". The "structure coordinator" applies these changes on the subordinate coordinator. The changes can be adding/deleting models, adding/deleting coupling relations, adding/deleting input or output ports. We note that an atomic model that manages structure is allowed for each variable structure coupled model. Thus, the complexity decreases in the low level models and the reusability increases.

4.3.1 Structure Model Algorithm

Fig. 2 presents the algorithm of the structure coordinator when it receives a *`Ymessage`(y,t)*.

The structure coordinator algorithm adds a variable named "`actions_list`", which includes the set of future updates that must be applied on the managed coupled model to reach the aimed structure.

When the structure coordinator receives `Ymessage`, it verifies at the first time if the message source is the child simulator or the child coordinator. If the source is the child coordinator, then, the EIC and IC are checked in order to determine respectively if the `Ymessage` will be forwarded to parent coordinator, or if the `Ymessage` will be forwarded to the child coordinator after being transformed into `Xmessage` by the coupling function F (x=F(y)). Otherwise, if the message source is the simulator, hence, the `Ymessage` is consulted to get the actions set that will be performed. Eventually, the EIC set is checked to determine if the `Ymessage` will be forwarded to the parent coordinator. The "structure coordinator" scrolls through the set of actions in order to perform each of them through "`execute_action`" function. Indeed, this function calls all structure changes actions related functions. We note that some structure changes imply an embedded initialization.

```
variables
   parent      //parent coordinator
   tl          //time of last event
   tn          //time of next event
   //subordinated coupled model
   DEVN{X,Y,{Md/d in D},D,EIC,EOC,IC,F}
   //list of elements (d,tnd) sorted by
   //tnd ordered list
   event-list
   d*          //selected imminent child
   //list of actions changing structure
   Actions_list

when receive Ymessage (y,t) with output y
   if d* is simulator
   send Ymessage with value YN=F(Yd*) to parent
   c*= yS.related_coordinator
            //loop to perform a set of actions
   while yS.actions_list not empty set
      do
        execute_action(yS.action(i),
                       yS.related_coordinator)
      end do
   else
      //check EOC to get external output event
   receivers={r|r in D-d*, r in {DEVN.IC.source}}
   if d* in {DEVN.EOC.source}
      send Ymessage with value YN=F(Yd*) to parent
   if d* in {DEVN.IC.source}
      send Ymessage with input value x=F(Yd*)to r
……
end DEVS-Structure-Coordinator
```

Fig. 2 Structure coordinator Algorithm (*`Ymessage`*)

4.3.2 "execute_action" Algorithm

Structure changes are included in the "`actions_list`" variable. Then the structure coordinator calls the execute action function to apply related changes. A part of the algorithm corresponding to this function is represented in Fig. 3. This function analyses the action sent by the structure coordinator. A part of these performed actions were described below.

"`delete_atomic(c*, c*.DEVS)`" is a function with two parameters; the concerned child coordinator "`c*`", and the model to delete "`c*.DEVS`". We note that deleting models implies deleting all related coupling relations.

"`add_atomic(c*, c*.DEVS, EIC_list, EOC_list, IC_list)`" adds to a the child coordinator "`c*`" a DEVS model "`c*.DEVS`", with related External Input Coupling "`EIC_list`", External Output Coupling "`EOC_list`", and Internal coupling "`IC_list`".

"`delete_EIC ((c*.N, c*.inport), (c*.d.target, c*.d.inport))`" deletes an external input coupling from the input port "`c*.inport`" in the model "`c*.N`", to the input port "`c*.target`" of the model "`c*.d.inport`".

"`add_EOC((c*.d.source, c*.d.outport), (c*.N, c*.outport))`" adds an external output coupling in the child coordinator. A coupling relation is from the output port "`c*.d.outport`" of the source model "`c*.d.source`", to the output port "`c*.outport`" of the coupled model "`c*.N`".
"`add_IC((c*.d.source, c*.d.outport), (c*.d.target, c*.d.inport))`" adds an internal coupling in the child coordinator. A coupling relation is from the output port "`c*.d.outport`" of the source model "`c*.d.source`", to the output port "`c*.d.inport`" of the coupled model "`c*.d.target`".
"delete_Inport(c*.d,c*.d.inport)" deletes an input port "c*.d.inport" in the model "c*.d".

```
execute_action(action,c*)
Variables d,DEVN,DEVS,c*
//c* is the coordinator that structure
//will change
Switch (action)
   Case "delete_atomic_model"
      delete_atomic(c*,c*.DEVS)
   Case "add_atomic_model"
      add_atomic(c*,c*.DEVS,EIC_list,
                 EOC_list,IC_list)
   Case "delete_EIC_model"
      delete_EIC((c*.N,c*.inport),
                 (c*.d.target,c*.d.inport))
   Case "add_EOC_model"
      add_EOC((c*.d.source,c*.d.outport),
              (c*.N,c*.outport))
   Case "add_IC_model"
      add_IC ((c*.d.source,c*.d.outport),
              (c*.d.target,c*.d.inport))
   Case "delete_Input_port"
      delete_Inport(c*.d,c*.d.inport)
   ..................
```

Fig. 3 Algorithm of "`execute_action`" function

We note that deleting input/output ports implies deleting all related coupling relations (EIC, EOC, and IC). Also, deleting an atomic model implies deleting the related coupling relations and the corresponding events. Then, deleting a coupled model implies deleting all submodels, and their related coupling relations. On the other hand, adding coupling relations implies adding depending events. In the same way, adding models implies adding dependent coupling relations and events. Each added model must be initialized to respect coherence of the whole model simulation.
Below we present some of these functions in order to explain the whole model structure changes impact, and boundaries of some structure changes.

5 Structure changing Algorithm

Obviously, altering structure becomes primary in dynamic and complex systems modelling and simulation. However, structure variability needs more details about limits and boundaries of this variability. This section describes summary some changing structure functions with limits of their applications.

5.1 Adding atomic model function

Fig. 4 describes the algorithm of the adding atomic model function. We review that all utilized models during simulation must be predefined in the used library.

```
Add_atomic( Coordinator c,
            AomicModel devsAtomic,
            Linkage EIC_list,
            Linkage EOC_list,
            Linkage IC_list)
{
Load devsAtomic;
int i=0;
While (EIC_list is not empty)
  do
     add_EIC(c, EIC_list[i]);
     i++;
  end do
i=0;
While (EOC_list is not empty)
  do
     add_EOC(c, EOC_list[i]);
     i++;
  end do
i=0;
While (IC_list is not empty)
  do
     add_IC(c, IC_list[i]);
     i++;
  end do
initializeAtomic(devsAtomic);
}
```

Fig. 4 Adding atomic model function

Models embedded in the initial structure were initialized with the classic simulation process introduced by Zeigler [8]. Hence, the first instruction in this function is to load the atomic model in the corresponding coupled model. Then related coupling relations were added to the coupled model, and finally the added model will be included. All new coupling relations are added to coupling relation lists (EIC, IC EOC). Before resume the whole simulation, the added atomic DEVS model must be initialized to preserve coherence and integrity of the global model.

5.2 Deleting atomic model function

```
delete_atomic(Coordinator c,
              AomicModel devsAtomic)
{
int i=0;
While (linkage is not empty)
  do
     if (devsAtomic in {linkage[i].source,
           linkage[i].target )
        then
        linkage[i].delete();
     i++;
  end do
devsAtomic.delete();
}
```

Fig. 5 Deleting atomic model function

As shown in fig. 5, the function of deleting DEVS atomic model is related to the corresponding coordinator. It deletes the atomic model, and scrolls coupling relation sets in order to delete all deleted model coupling relation links. We note that "linkage" describes all coupling relations including EIC, EOC, and IC sets. To perform this action we have to verify that this atomic model is not the only embedded influenced/influencer submodel.

5.3 Deleting input port function

Altering ports in DEVS models during simulation is critical. Only recently input and output ports dynamic changing has been discussed in the context of system theoretical approaches toward modelling and simulation [4].

```
delete_InPort(AomicModel devsAtomic,
              Port inputPort)
{
int i=0;
While (linkage is not empty)
  do
     if (inputPort in linkage[i])
        then
        linkage[i].delete();
     i++;
  end do
i=0;
While (event-list is not empty)
  do
     if (inputPort in event-list[i])
        then
        event-list[i].delete();
        i++;
  end do

delete inputPort;
}
```

Fig. 6 Deleting input port function

As shown in Fig. 6, the function scrolls all events and links related to the port and delete them. We note that deleting each coupling relation implies deleting of all related events. To perform this function the model must respect the condition that the port is not involved in the last coupling relation, and that deleting this port will not isolate a component model.

6 Conclusion

This paper presents our dynamic structure hierarchical modelling approach based on the DEVS formalism. We review above the formal model of the dynamic hierarchical structure approach, and we describe the corresponding architecture. We note that our approach preserves the DEVS formalism properties and allows us to profit of all DEVS proved capabilities. It separates the dynamic structure and dynamic behaviour representation, which enhances modularity and reusability of the generated models. We focus here on the approach limits and boundaries through brief presentation of some change structure functions.

Presently, we're working on an extension of an existent DEVS M&S environment, "LSIS_DME" which was developed with JAVA language by LSIS laboratory team. Then we will implement designed models of Low intensity conflicts presented in [18], in order to determine the real capabilities and limits of this approach.

7 References

[1] Vangheluwe, H.L.M., "DEVS as a common denominator for multi-formalism hybrid systems modelling", Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on. Anchorage, Alaska, USA, September 25-27, 2000

[2] Wainer, G., Madhoun, R., "Creating Spatially-Shaped Defense Models Using DEVS and Cell-DEVS", The Society for Modeling and Simulation International, ***JDMS,*** **Volume 2, Issue 3, July 2005 Pages 121–143**

[3] Yilmaz, L. and T.I. Ören. 2005. "Discrete-Event Multimodels and their Agent-Supported Activation and Update." In Proceedings of the Agent-Directed Simulation Symposium of the Spring Simulation Multiconference. SMC'05, (April 2005), San Diego, CA, 63-72.

[4] A. M. Uhrmacher, J. Himmelspach, M. Röhl, and R. Ewald, 2006. "Introducing variable ports and multi-couplings for Cell biological modeling in DEVS", in Proceedings of the 2006 Winter Simulation Conference, IEEE

[5] F. Barros. 1998. "Multimodels and Dynamic Structure Models: An Integration of DSDE/DEVS and OOPM", In Proceedings of the 1998 Winter Simulation Conference, pp. 413-419, Winter Simulation Conference, December-1998

[6] L. Baâti, C. Frydman, N. Giambiasi, "Simulation Semantics for Dynamic Hierarchical Structure DEVS Model" DEVS06. In Proceedings of DEVS'06-SpringSim'06, Huntsville Alabama, April 2006.

[7] L. Baâti. 2007 "Simulators for DEVS models with Dynamic Structure". AIS-CMS International modeling and simulation multiconference. Buenos Aires - Argentina. February 8th to 10th 2007. Submitted & accepted.

[8] Zeigler, B. P.; Praehofer, H.; Kim, T. G. 2000. *Theory of modeling and simulation*, Second edition, integrating discrete event and continuous complex dynamic systems, Academic Press, 2000.

[9] LEE, K., CHOI, K., KIM, J., AND VANSTEENKISTE, G., 1997, "A methodology for variable structure system specification: Formalism, framework, and its application to ATM-based network systems", ETRI Journal 18, 4, 245–254.

[10] Uhrmacher, A.M., 2001, "Dynamic Structures in Modeling and Simulation - A Reflective Approach", ACM Transactions on Modeling and Simulation, Vol.11. No.2 , (April): 206-232.

[11] T. Pawletta, S. Pawletta. 2004. "A DEVS-based simulation approach for structure variable hybrid systems using high accuracy integration methods". In Proceedings of the Conference on Conceptual Modeling and Simulation, Part of the Mediterranean Modelling Multiconference CSM2004-I3M, (October 28-31), Genova, Italy, Vol.1, 368-373.

[12] Barros, F.J.; Mendes, M.T.; Zeigler, B.P. "Variable DEVS-variable structure modeling formalism: an adaptive computer architecture application". 'Distributed Interactive Simulation Environments'., In Proceedings of the Fifth Annual Conference on , 7-9 Dec 1994 Page(s): 185 -191.

[13] Fernando Barros. 1998. "Abstract Simulators for the DSDE Formalism." In Proceedings of the 1998 Winter Simulation Conference, pp. 407-412, 1998 Winter Simulation Conference, December-1998.

[14] F.J. Barros. "Modeling Formalism for Dynamic Structure Systems". ACM Transactions on Modeling and Computer Simulation, 7(4):501--514, 1997.

[15] F. Barros. 1998. "Multimodels and Dynamic Structure Models: An Integration of DSDE/DEVS and OOPM", In Proceedings of the 1998 Winter Simulation Conference, pp. 413-419, Winter Simulation Conference, December-1998.

[16] Xiaolin. Hu, 2004, "A Simulation-based Software Development Methodology for Distributed Real-time Systems", Ph. D. Dissertation, Electrical and Computer Engineering Dept., University of Arizona, May 2004.

[17] Xiaolin Hu, B. P. Zeigler, and S. Mittal, "Variable Structure in DEVS Component-Based Modeling and Simulation", SIMULATION: Transactions of The Society for Modeling and Simulation International, Vol. 81, No. 2, pp. 91-102, 2005.

[18] L. BAATI, C. FRYDMAN and N. GIAMBIASI, 2007. "LSIS_DME M&S Environment Extended by Dynamic Hierarchical Structure DEVS Modeling Approach", DEVS'07, in Proceedings of SpringSim'07, Norfolk VA, USA

