Specification of dynamic structure discrete event systems using single point encapsulated control functions

Alexandre Muzy, Bernard P. Zeigler

Specification of dynamic structure discrete event systems using single point encapsulated control functions

Alexandre Muzy*, Bernard P. Zeigler**

* I3S UMR CNRS 7271, Bio-info team, CS 40121 - 06903 Sophia-Antipolis Cedex, France, Email: alexandre.muzy@cnrs.fr.
** Chief Scientist, RTSync Corp, 530 Bartow Drive Suite A Sierra Vista, AZ 85635, United-States of America, Email: zeigler@rtsync.com.

Abstract

In Discrete Event System Specification (*DEVS*), the dynamics of a network is constituted only by the dynamics of its basic components. The state of each component is fully encapsulated. Control in the network is fully decentralized to each component. At dynamic structure level, *DEVS* should permit the same level of decentralization. However, it is hard to ensure structure consistency while letting all components achieve structure changes. Besides, this solution can be complex to implement. To avoid these difficulties, usual dynamic structure approaches ensure structure consistency allowing structure changes to be done only by the network having new added dynamics change capabilities. This is a safe and simple way to achieve dynamic structure. However, it should be possible to simply allow components of a network to modify the structure of their network, other components and/or their own structure - without having to modify the usual definition a *DEVS* network. In this manuscript it is shown that a simple fully decentralized approach is possible while ensuring full modularity and structure consistency.

1 Introduction

In systems theory tradition, the discrete event specification has sought for many years to specify dynamic structure sytems:

- Dynamic Structure Discrete Event System Specification (*DSDEVS*)[1]: Where a single central controller is in charge of executing structure changes. Having a single locus of control for structure changes constitutes a relatively simple way of ensuring both behavior and structure consistencies.

- *DynamicDEVS*[2]: Where a sequential implementation allows local and decentralized *internal structure* changes. Interface structure changes as well as the addition/deletion of components are subsequently integrated at network level.
- Variable structures[3]: Contrary to the two previous works this is not a formal approach. However, it is an attempt to have many decentralized loci of control for achieving structure changes. Local components are able to *sequentially*[1] modify the *whole structure* of other components, in the same network.
- Continuous Flow System Specification (*CFSS*)[4]: Where the implementation of multirate integration methods and dynamic stucture models can be achieved. *CFSS* components *sample* directly their influencers' states.

To deal with the autonomy of structure changes, the notion of *single point of control* is introduced here. In a *single point of control*, at each time, only one component is responsible of structure changes. This component can always be the same for the whole simulation (*static single point*) or can change (*dynamic single point)*.

The scope of the present contribution is twofold:

1. To constitute a coherent framework for usual dynamic structure formalisms. This framework would allow the different formalisms to be represented with the same elements and mechanisms. This is of interest for the community, *e.g.,* to debate differences between formalisms,
2. To propose a fully decentralized modular approach closer to reality and *DEVS* thus opening new exciting research perspectives.

The manuscript is organized as follows. In Section 2, both static and dynamic structure specifications of dynamic systems are defined. In Section 3, both fixed and dynamic single points of control of structure changes are used to represent usual formalisms. In Section 4 a fully modular and autonomous approach is proposed. Finally, in Section 5, conclusion and perspectives close the manuscript.

2 Discrete event and dynamic structure specifications of dynamic systems

Structure changes are defined as based on discrete event transitions.

2.1 Usual static structure formalism

The structure of both network and basic discrete event systems is presented here.

[1] In the Discrete Event System Specification, concurrent events (changes of states), occurring at the same time, are executed one after the other. Each change of state influencing other conccurrent state changes, at current time.

2.1.1 Basic (Atomic) Discrete Event Specification

Definition 2.1. A basic Discrete Event System Specification ($DEVS$) is a structure:

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

Where, X is the *set of input events*, Y is the *set of output events*, S is the *set of partial states*, $\delta_{ext} : Q \times X \to S$ is the *external transition function* with $Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ the set of total states, $\delta_{int} : S \to S$ is the *internal transition function*, $\lambda : S \to Y$ is the *output function*, and $ta : S \to \mathbb{R}^{0,+}_{\infty}$ is the *time advance function.*

2.1.2 Network structure

Definition 2.2. A $DEVS$ network is a structure:

$$N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$$

Where X is the set of input events, Y is the set of output events, D is the set of component names, for each $d \in D$, M_d is a basic model (whose structure differs from one $DEVS$-based formalism to another), for each $d \in D \cup \{N\}$, I_d is the set of *influencers* of d such that $I_d \subseteq D \cup \{N\}, d \notin I_d$ and, for each $i \in I_d$, $Z_{i,d}$ is a *coupling function*, the $i-$ to $-d$ output translation, defined for: (i) *external input couplings*: $Z_{self,d} : X_{self} \to X_d$, with $self$ the network name, (ii) *internal couplings*: $Z_{i,j} : Y_i \to X_j$, and (iii) *external output couplings*: $Z_{d,self} : Y_d \to Y_{self}$, and $Select : 2^D - \{\emptyset\} \to D \cup \{\emptyset\}$ is the *sequential select function* (to select one component to execute its transition/output functions, among imminent components). Considering a set of components C candidate for internal transition, the sequential select function has constraint $Select(C) \in C \cup \{\emptyset\}$, *i.e.*, only one component or no components can be selected among candidates.

2.2 Dynamic structure of dynamic systems using a discrete event specification

Both network and basic dynamic structure systems are presented here.

2.2.1 Basic dynamic structure

Definition 2.3. *Basic or atomic Dynamic Structure Discrete Event System Specification* (DYS-DEVS) *structure*

$$DYS\text{-}DEVS = (\mathcal{M}, \mathcal{S}, \tau)$$

Where each element $M \in \mathcal{M}$ is a structure $DEVS = (X, Y, S, \delta_{ext},, \delta_{int}, \lambda, ta)$, $\mathcal{S} = \amalg_{M \in \mathcal{M}} S_M$ is the disjoint union of their partial state sets, and $\tau : \mathcal{M} \times \mathcal{S} \to \mathcal{M} \times \mathcal{S}$ is *the structure transition function.* Structure functionτ takes a basic $DEVS$ and its state to a new basic $DEVS'$ and a new state (could

be the same also): $\tau(M, s) = (M', s')$. This represents *a basic change in structure* which *transforms a basic DEVS into a new basic DEVS'*, by changing its structure in some way (one or many elements of $(X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$) and initializing the state of the new *DEVS*. To use this representation the sets $\mathcal{M}$ (of *DEVS*, it can generate), $\mathcal{S}$ (of their states), and mapping τ (how the structure change occurs) are identified after in the manuscript.

At network level, basic structure components are authorized to modify the whole network. At local level, for a basic structure component, modifying its interface requires modifying related couplings (in the network) and related inputs/outputs (in another component). The impact of interface structure changes goes a little beyond the frontiers of the component. To account for these impacts, the concept of *external and internal models* is defined here.

Definition 2.4. In a basic *DYS-DEVS* $= (\mathcal{M}, \mathcal{S}, \tau)$ a model $M \in \mathcal{M}$ can be decomposed into an *external* model *part* M_{ext} and into an *internal model* part M_{int}, and structure transition function τ can be decomposed into an *external structure transition function* τ_{ext} and into an *internal structure transition function* τ_{int}, where:

- $M_{ext} = (X, Y)$ is changed by the *external structure transition function* $\tau_{ext}(M_{ext}, \delta_{ext}(s, e, x))$, and
- $M_{int} = (S, \delta_{ext}, \delta_{int}, \lambda, ta)$ is changed by *internal structure transition function* $\tau_{int}(M_{int}, \delta_{int}(s))$.

Example 2.1. *Internal* structure changes of a basic *DYS-DEVS*.

Assume basic component *DYS-DEVS* $= (\mathcal{M}, \mathcal{S}, \tau)$ is in state (M, s), with $M_{int} = (S, \delta_{ext}, \delta_{int}, \lambda, ta)$ its internal model, when it receives input $x =$ change from another component. Then, a new state is obtained as $\delta_{ext}(s, e, x) =$ changeInternalStructure. New structure $M'_{int} = (S', \delta'_{ext}, \delta_{int}, \lambda, ta')$ is obtained as $\tau_{int}(M_{int}, s) = (M'_{in}, s')$. Notice that differences between structures M_{int} and M'_{int} consist in new sets S', new external transition function δ'_{ext}, and new time advance function ta'.

2.2.2 Dynamic structure network

Definition 2.5. *Dynamic Structure Discrete Event Network System* (DYS-DEN) *structure*

$$DYS\text{-}DEN = (\mathcal{N}, \mathcal{S}, \tau)$$

Where $\mathcal{N} = \{(X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)\}$ is the set of network structures, where each component $d \in D$ is an atomic dynamic structure model $M_d = (\mathcal{M}_d, \mathcal{S}_d, \tau_d)$, $\mathcal{S} = \Pi_{N \in \mathcal{N}} S_N$ is the disjoint union of partial state sets of network structures, with $S_N = \Pi_{d \in D} S_d$ the partial state set of a network $N \in \mathcal{N}$ is the crossproduct of the partial state sets of its components, and $\tau : \mathcal{N} \times \mathcal{S} \to \mathcal{N} \times \mathcal{S}$ is the *structure transition function* of the *network*.

Next example briefly introduces the structure changes at network level.

Example 2.2. *Simple changes of network structure (*cf. *Figure 1)*

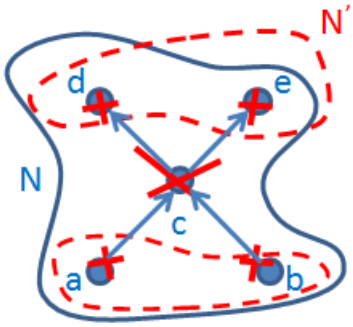

Figure 1: A simple change of network structure

Consider a simple network structure $N = (D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$, *where*
$D = \{a, b, c, d, e\}$, $I_c = \{a, b\}$, $I_d = \{c\}$, $I_e = \{c\}$, *and* $Z_{a,c} : Y_a \to X_c$, $Z_{b,c} : Y_b \to X_c$, $Z_{c,d} : Y_c \to X_d$, $Z_{c,e} : Y_c \to X_e$. *The state set of the components in the network is* $S = S_a \times S_b \times S_c \times S_d \times S_e$. *Now assume that each component state set is* $\{0, 1\}$. *Then, the state set of the network is* $S = \{0, 1\} \times \{0, 1\} \times \{0, 1\} \times \{0, 1\} \times \{0, 1\}$, *with,* e.g., *particular states* $(0, 0, 0, 0, 0), (1, 0, 0, 0, 0), ...$, *where the first component is for component a, the second component for component b, the third component for component c, etc.*

At a particular instant, component c is removed. Then, network structure N *defined as* $(D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$ *changes to* N' *defined as* $(D', \{M'_d\}, \{I'_d\}, \{Z'_{i,d}\}, Select')$, *where* $D' = \{a, b, d, e\}$, $I'_c = I'_d = I'_c = \{\emptyset\}$, $\{Z'_{i,d}\} = \{\emptyset\}$, *and* $\{M'_d\} = \{M'_a, M'_b, M'_d, M'_e\}$. *The state set of the components in network* N' *is now* $S' = S_a \times S_b \times S_d \times S_e$. *Notice that components* a, b *and components* d, e *are impacted by the deletion of component c having respectively their output and input sets removed.*

Now suppose that these structure changes can be achieved by network structure transition function $\tau(N, s) = \tau(N, (1, 1, 0, 0, 0)) = N'$. *Assume also that value* 0 *means* "no change intention" *and value* 1 *means* "change *intention". Finally,* $\tau(N, (1, 1, 0, 0, 0))$ *means that both components a and b have both intention to make the network structure change to* N' *at the same time. It will be seen hereafter how this kind of* simultaneous local changes of structure *in the network can be* serialized.

Generally speaking, we will show that,

Proposition 2.1. A network $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$, where for each $d \in D$, M_d is a basic *DYS-DEVS* $_d = (\mathcal{M}_d, \mathcal{S}_d, \tau_d)$, is equivalent to a *resultant DYS-DEVS* $= (\mathcal{M}, \mathcal{S}, \tau)$.

3 Representation of usual formalisms

In *DSDEVS* formalism, there is only one component in every instance of $\mathcal{M}$ that makes the decision for the next instance. Structure transition function τ is defined by mimicking the changes achieved by the executive in one transition. On the other hand, *DynamicDEVS* formalism allows multiple local decision points but changes of network structure are done only at network level.

DYS-DEVS uses the usual dynamic mechanisms of *DEVS*, using states changes for synchronizing dynamic structure transitions. Serialization of structure changes is based on the notions of *static* and *dynamic single points of control for structure changes*.

Definition 3.1. A *static single point of control* consists of having only one dynamic structure component, always the same, in the whole network.

Definition 3.2. *A dynamic single point of control* consists of having many dynamic structure components in the whole network. However, at each time step, only one component can be authorized to achieve structure changes.

DSDEVS formalism is represented as a static point of control, while *Dyn-DEVS* formalism is represented as a dynamic point of control. More generally, it is shown that,

Proposition 3.1. *Different existing formalisms for dynamic structure DEVS can be represented in the Dynamic Structure Formalism Framework (*DYS-F*) by different choices of $\mathcal{M}$, $\mathcal{S}$ and τ.*

3.1 Equivalence of basic dynamic structure component and basic discrete event component

Theorem 3.1. *An atomic DYS-DEVS $= (\mathcal{M}, \mathcal{S}, \tau)$ is equivalent to an atomic DEVS $= (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$,* where the state set is $S = \mathcal{M} \times \mathcal{S}$, with $\mathcal{M}$ a set of basic *DEVS* models and $\mathcal{S}$ is the disjoint union of their state sets: $\mathcal{S} = \amalg_{M \in \mathcal{M}} S_M$.

Proof. We describe the dynamics of both *DYS-DEVS* and *DEVS* defining the elements of a *DEVS* in terms of the elements of a *DYS-DEVS*.

The internal transition function of a *DYS-DEVS* is

$$\delta_{int}(M, s) = \tau(M, \delta_{int,M}(s))$$

i.e., first apply the internal transition function of the current structure $M \in \mathcal{M}$ to state $s \in S_M$ to get new state $\delta_{int,M}(s)$, then apply the structure transformation to this pair to get a new structure and a new state (M', s').

Similarly, the external transition function is defined by:

$$\delta_{ext}(M, s, e, x) = \tau(M, \delta_{ext,M}(s, e, x))$$

The output function is defined by:

$$\lambda(M, s) = \lambda_M(s)$$

i.e., output function of the current structure $M \in \mathcal{M}$ sends current state $s \in \mathcal{S}$. The time advance function is

$$ta(M, s) = ta_M(s)$$

i.e., time advance of the current structure $M \in \mathcal{M}$ is applied to its state, $s \in \mathcal{S}$, to compute the occurrence time of next state change. □

Based on structure transition function τ, a basic *DYS-DEVS* changes a structure $M \in \mathcal{M}$, using partial state $s \in \mathcal{S}$. A new state is obtained by the execution of one of the two usual transition functions: $\delta_{ext,M}(s, e, x)$ or $\delta_{int,M}(s)$. Then, structure change depends on total state $(s, e) \in S \times \mathbb{R}_{\infty}^{0,+}$, and possibly on external input event $x \in X$.

3.2 Static single point

Centralized control of structure changes is investigated here. Structure changes are controlled only by one component. No other component can change the network structure.

Lemma 3.1. *Considering an initial network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$, *where the set of component indexes* $D = \{1, 2, ..., p\}$ *and the state set of the network is* $S = S_1 \times S_2 \times ... \times S_p$, *where* S_1 *is the state set of the* first component, S_2 *is the state set of the second component, etc., If a single point of control is at first component* DYS-DEVS$_1 = (\mathcal{M}_1, \mathcal{S}_1, \tau_1)$, *while each other component* $i \in D$, *with* $i > 1$ *is a basic component* $DEVS_i$, *the set of networks* $\mathcal{N}$ *is equivalent to a* resultant DYS-DEVS $= (\mathcal{M}, \mathcal{S}, \tau)$.

Proof. A single point of control at first component *DYS-DEVS*$_1$, would be that τ always accounts for the state of *DYS-DEVS*$_1$ to make its decision; so $\tau(s_1, s_2, ..., s_p) = \tau_1(s_1)$ for some τ_1. Then, the image of τ depends on the new components added to the network or not (because the states of new components have to be initialized).

Denoting new structures as $N' = (X', Y', D', \{M'_d\}, \{I'_d\}, \{Z'_{i,d}\})$, structure transition function $\tau : \mathcal{M} \times \mathcal{S} \to \mathcal{M} \times \mathcal{S}$ reduces to one of the two maps:

1. For each non-created component $d \in D \cap D'$, $\tau : \mathcal{S}_1 \to \mathcal{M}$, with $\tau(\ldots, s_d, \ldots) = \tau_1(s_1) = N'$,
2. For each new component $i \in (D' - D)$ (created), initialized to initial state $s_{0,i}$, $\tau : \mathcal{S}_1 \to \mathcal{M} \times \mathcal{S}$, with $\tau(\ldots, s_i, \ldots) = \tau_1(s_1) = (N', (\ldots, s_{0,i}, \ldots))$.

□

Definition 3.3. A *DSDEVS* network[1] is a structure $DSDEN = (\chi, M_\chi)$, with *executive model* $M_\chi = (X_\chi, Y_\chi, S_\chi, \gamma, \Sigma^*, \delta_\chi, \lambda_\chi, ta_\chi)$, where a nework

structure $\Sigma \in \Sigma*$ is given by $\Sigma = \gamma(s_\chi) = (D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$ and $\gamma : Q_\chi \to \Sigma^*$, with $\chi \notin D$.

Corollary 3.1. *A DSDEVS network is equivalent to a DYS-DEVS* $= (\mathcal{M}, \mathcal{S}, \tau)$ *having a single point of control DYS-DEVS*$_1$ *in charge of the structure changes in a network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$*, where* $D = \{1, 2, ..., p\}$ *and for each component* $i \in D$*, with* $i > 1$*,* M_i *is a basic DEVS model.*

Single point of control DSP-DEVS$_1$ *can be described in terms of* executive model M_χ *with:* $\mathcal{M}_1 = \Sigma^*$*,* $\mathcal{S}_1 = S_\chi$*,* $D_\chi = \text{proj}_D(\gamma(\text{s}_\chi) \cup \{\chi\}$*,* and $\tau_1 = \gamma$.

In Continuous Flow System Specification (*CFSS*)[4], components *sample* directly their influencers' states (in a one-step process) while usual *DEVS* components have to request and receive their influencers' states (in a two-step process)[5]. Therefore, transforming a *CFSS* network into a *DEVS* one consists of mapping each original coupling into two couplings (one for request, one for answer). Another soution would be, contrary to *CFSS*, to break components' modularity through the *multicomponent* approach[6]. In the dynamic structure context, *DYS-DEVS* equivalence can be achieved preserving modularity (*at* dynamic structure *network controller level*) adding extra coupling (*cf. Barros' description*[5] for details).

3.3 Dynamic single point

Example 3.1. *Dynamic structure authorization by "token" passing*

Consider now a single point of control "passing" around, *activating*, the components - just like a "token" in network where each node gets a chance to send when it has the token. In this example the state set of Example 2.2 would be $\mathcal{S} = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ with the token (authorization) going from a to b to c back to a, and so on in a cyle. Instead of "sending" the node with the token can do any structure change with the global state being initialized to the next triple in the cycle. Here, only one component among the components of the network, can be activated at a time.

Theorem 3.2. *Consider an initial network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$*, where for each* $d \in D$*,* M_d *is a basic dynamic structure* DYS-DEVS$_d$ *and the state set of the network is* $S = S_1 \times S_2 \times ... \times S_p$*, where* S_1 *is the state set of the* first component DYS-DEVS$_1$*,* S_2 *is the state set of the second component* DYS-DEVS$_2$*, etc. If a dynamic single point of control is assigned* sequentially *and* cyclically *to each component* DYS-DEVS$_d$ *for structure changes on the components of the network, the set of networks* $\mathcal{N}$ *is equivalent to a* resultant DYS-DEVS $= (\mathcal{M}, \mathcal{S}, \tau)$.

Proof. Extending Lemma 3.1, it is simple to consider that a *cycle* of dynamic single points of control is recursively defined by global and local structure transition functions:

$$\begin{cases} \tau: (1,0,0,...,0) \mapsto (N', (0,1,0,...,0)) \\ \quad with\ \tau_1(1,0,0,...,0) = (N', (0,1,0,...,0)) \\ \tau: (0,1,0,...,0) \mapsto (N'', (0,0,1,...,0)) \\ \quad with\ \tau_2(0,1,0,...,0) = (N'', (0,0,1,...,0)) \\ \quad ... \\ \tau: (0,0,0,...,1,0) \mapsto (N^p, (0,0,0,...,1)) \\ \quad with\ \tau_p(0,0,0,...,1,0) = (N^p, (0,0,0,...,1)) \end{cases}$$

Then, the resultant $DYS\text{-}DEVS = (\mathcal{M}, \mathcal{S}, \tau)$ is defined with $\mathcal{M} = \mathcal{N}$, $\mathcal{S} = \{(1,0,0,...,0), (0,1,0,...,0), (0,0,1,...,0), ..., (0,0,0,...,1)\}, \tau(s) = s{+}1 mod_p$, with p the number of components and state $s = \{0,1\}^p$. □

Remark 3.1. This is different from $DSDEVS$, which does not allow explicitly changing a dynamic single point of control.

Definition 3.4. A $DynamicDEVS$ network is a structure

$$DynNDEVS = (X, Y, n_{init}, \mathcal{N}(n_{init}))$$

Where X, Y are input and output event sets, $n_{init} \in \mathcal{N}(n_{init})$ is the initial structure, and $\mathcal{N}(n_{init})$ the least (minimum) set having the structure $\{(D, \rho_N, \{dynDEVS_i\}, \{I_i\}, \{Z_{i,j}\}, Select)\}$ with:

- $D, \{I_i\}, \{Z_{i,j}\}$ as defined previously,
- $\rho_N : S \to \mathcal{N}(n_{init})$ is the network transition function with $S = \Pi_{i \in D}(\Pi_{m \in dynDEVS_i} S^m)$, with $dynDEVS_i$ the *dynamicDEVS* model $i \in D$,
- $dynDEVS_i = (X_i, Y_i, m_{init,i}, \mathcal{M}_i(m_{init,i}))$, with:
  - X_i, Y_i the input and output event sets,
  - $m_{init,i} \in \mathcal{M}_i(m_{init,i})$ the initial model, and
  - $\mathcal{M}_i(m_{init,i})$ the least (minimum) set of *internal* structure $\{(S_i, \delta_{ext,i}, \delta_{int,i}, \rho_{\alpha,i}, \lambda_i, ta_i)\}$ of usual atomic $DEVS$, except $\rho_{\alpha,i} : S_i \to \mathcal{M}_i(m_{init,i})$ the model transition function.
- $Select : 2^D - \{\emptyset\} \to D$ is the *sequential select function.*

Corollary 3.2. *Using a single dynamic point of control, a network* $DynNDEVS = (X, Y, n_{init}, \mathcal{N}(n_{init}))$ *can be represented by an initial network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$ *where* $D = \{1, 2, ..., p\}$, M_1 *is a dynamic structure network* DYS-DEN$_1$ *and each other component* $d \in D$, *with* $d > 1$ *is a basic dynamic structure component* DYS-DEVS$_d$.

A $DynNDEVS$ *network operates along two sequential steps: (i) Locally, basic components* $dynDEVS_i$ can change only their *internal model* according to their model transition function $\rho_{\alpha,i} : S_i \to \mathcal{M}_i(m_{init,i})$, *then (ii)* DynNDEVS *network can change its structure (interfaces* (X_i, Y_i) *and* $D, \{I_i\}, \{Z_{i,j}\}$,

adding/removing components) through the network transition function $\rho_N : S \to \mathcal{N}(n_{init})$, *with* $S = \Pi_{d \in D}(\Pi_{m \in dynDEVS_i} S^m)$.

Each basic $dynDEVS_i = (X_i, Y_i, m_{init,i}, \mathcal{M}_i(m_{init,i})$ *can be represented by a* DYS-DEVS$_d = (\mathcal{M}_d, \mathcal{S}_d, \tau_d)$ *with correspondences:* $\mathcal{M}_d = \mathcal{M}_i(m_{init,i})$, *with* $M_d \in \mathcal{M}_d$ *restricted to internal model* $M_{int,d} = (S_d, \delta_{ext,d}, \delta_{int,d}, \lambda_d, ta_d)$, $\mathcal{S}_d = S_i$, $\tau_d = \rho_{\alpha,i}$ *restricted to* $\tau_d : S_i \to \mathcal{M}_i(m_{init,i})$. $DynNDEVS$ *is represented by dynamic structure network* $DYS\text{-}DEN_1 = (\mathcal{N}_1, \mathcal{S}_1, \tau_1)$, *with correspondences:* $\mathcal{N}_1 = \mathcal{N}(n_{init})$, $\mathcal{S}_1 = \Pi_{N \in \mathcal{N}} S_N$ *with* $S_N = \Pi_{i \in D} S_i = S_1$, $\tau_1 = \rho_N$ *restricted to* $\tau_1 : S_1 \to \mathcal{N}(n_{init})$.

Remark 3.2. The Dynamic Structure Formalism Framework allows representing a $DynamicDEVS$ network, limitating: (i) local structure changes to be only internal structure changes of atomic models, and (ii) global structure changes to be achieved only by the dynamic structure network.

Another class of dynamic structure systems consists of *mobile agents*. Modeling mobile agents has been done using the Heterogeneous Flow System Specification (*HFSS*) formalism, which combines with the Continuous Flow System Specification (*CFSS*) to represent continuous flow systems and *DEVS[7]*. A set of connected networks (each one embedding an executive) sequentially add, transmit, and then destroy a single migrating agent. It can be easily shown that this is equivalent to a *dynamic sequential single point of control*, each point achieving only self-changes of structure. For the same class of mobile agents, a *DEVS*-based formalism has been proposed: Mobile DEVS (*MDEVS*)[8]. In this formalism, many agents can be "added, transmitted, and then destroyed" in the networks. It can be shown that this formalism is also equivalent to the case of *dynamic single points of control*.

4 Decentralization of structure change operations

Using a dynamic single point of control allows enhancing decentralization at two levels:

1. Globally: Having each dynamic structure component operating at network level. This is already a step toward decentralization with respect to usual dynamic structure formalisms (which are centralizing network operations). However, we will see that this approach can be considerered as *partially modular*.

2. Locally: Having each dynamic structure component operating at interface and couplings levels (here with influencee permission). This *new approach* can be considerered as *fully modular*.

4.1 Counter-arguments to usual dynamic structure modularity

The hierarchy of systems specification[6] is grounded on components *modularity*: The state of components can only be changed: (i) *externally* by another compo-

nent, through interface interactions, or (ii) *internally* by the component itself. In computer programming this is called *encapsulation.* In dynamic structure systems, changing other-structure remains a major issue. Changing self-structure can impact the structure of the network and of other components (*e.g.,* deleting self-output requires deleting corresponding coupling and other-input of influencee components). Then, because of structure change propagation it is hard to ensure structure consistency at component and network level.

As depicted in previous section, one solution is to have only one (network) component in charge of coupling changes (*DynDEVS*) or all structure changes (*DSDEVS*). Authors' philosophy could be sum up by argument: "only the network can change the interface structures of its components to ensure modularity". However, it can be argued a major counter-argument:

> *Allowing networks to achieve structure/state change is a holistic change of perspective while the usual hierarchy of systems specification is purely reductionnist (the network having no ability to change structure/state being merely a composition of dynamic components).*

Then, allowing networks to change the structure/state of components could also be considered as a violation of the modularity concept simply because then components are not the only ones to change their state.

Having a static point of control is a simplification of purely autonomous systems only interacting through interfaces. Allowing many components to change each-other structure requires defining synchronization interaction protocols that can rapidly become complex to implement. In the next subsections we define such protocols for elementary structure change operations. These mechanisms can be automated and combined to achieve multiple structure changes. Now let's first achieve a first step towards decentralization having each dynamic structure component being able to operate at network level.

4.2 Global structure change operations

Theorem 3.2 already showed that considering an initial network $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$, where each component $d \in D$ is a basic dynamic structure component $DYS\text{-}DEVS_d$, each network structure $N \in \mathcal{N}$ can be reached by a resultant DYS-DEVS $= (\mathcal{M}, \mathcal{S}, \tau)$. However, the global state of components is used locally for component selection thus decreasing control autonomy.

Here, the whole system is simplified ensuring network structure consistency and having more autonomy at component level. Each component $d \in D$ of the network is a dynamic structure network component $DYS\text{-}DEN_d = (\mathcal{N}_d, \mathcal{S}_d, \tau_{int,d})$ with $\tau_{int,d} : \mathcal{S}_d \to \mathcal{N}_d$. Notice that *stucture transition functions* $\tau_{int,d}$ are *internal* ones, *i.e.*, based on internal state transitions.

Theorem 4.1. *Consider an initial network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$, *where for each* $d \in D$, M_d *is a dynamic structure network* DYS-DEVN$_d = (\mathcal{N}_d, \mathcal{S}_d, \tau_{int,d})$, *the state set of the network is* $S = S_1 \times S_2 \times ... \times S_p$, *where* S_1 *is the state set of the* first component DYS-DEVN$_1$, S_2 *is the state set of the second component* DYS-DEVN$_2$, *etc. If a dynamic single point of control is successively assigned to only one component* DYS-DEVN$_d = (\mathcal{N}_d, \mathcal{S}_d, \tau_{int,d})$, *the set of networks* $\mathcal{N}$ *is equivalent to a* resultant DYS-DEVS $= (\mathcal{M}, \mathcal{S}, \tau_{int})$.

Proof. The main difference with Theorem 4.1 is that there is *no cycle* of dynamic single points of control. Remember now that the execution of each *internal stucture transition* $\tau_{int,d}$ is driven by an internal state transition $\tau_{int,d}(\delta_{int,d}(s_d))$. A component candidate for a structure change is thus candidate first for an internal transition. And here is the interesting point in usual *DEVS*, at each global state transition, only one component, among candidates for internal transitions[2], is chosen by *Select* function. Therefore, only one candidate for structure change is chosen at each global state transition avoiding structure conflicts.

Finally, each resultant structure change transition consists of the execution of one dynamic structure network component $d^* = Select(IMM)$, i.e.,

1. For each non-created component $i \in D \cap D'$, $\square$ $\tau(\ldots, s_i, \ldots) = \tau_{d^*}(s_{d^*}) = N'$.

2. For each new component $i \in (D' - D)$, initialized to initial state $s_{0,i}$, $\tau(\ldots, s_i, \ldots) = \tau_{d^*}(s_{d^*}) = (N', (\ldots, s_{0,i}, \ldots))$.

4.3 Local structure change operations

Here come the tricky structure change operations achieved by basic components. To ensure structure consistency, at both local and global levels, synchronization mechanisms are defined.

4.3.1 Dynamic structure synchronization

To ensure modularity, components cannot change other-interfaces. As for state changes, structure changes can only be asked through interface interactions and achieved by the component itself. Special input "query" and special output "done" of basic dynamic structure components are used for change synchronization.

Definition 4.1. Structure change synchronization conditions:

[2] *Candidates for internal transition compose the imminent set* $IMM = \{\sigma_d \mid d \in D \wedge \sigma_d = ta(s)\}$, with σ_d *the time remaining to the next event* $\sigma_d = ta_d(s_d) - e_d$, and $ta_d(s_d)$ the *time advance* of a component model.

- Each dynamic structure component has extra structure query/done interface.
- Each dynamic structure influencer has query outgoing coupling and done incoming coupling with all its influencees.
- Dynamic structure components can:
  - change its *external structure* and corresponding *outgoing couplings* after request/done protocol (querying corresponding influencee to add/remove corresponding input),
  - change its internal structure,
  - create/remove other components,
  - query its influencees to perform structure changes.

Proposition 4.1. *Changing other-structure can only be achieved through interface interactions. This respects totally modularity concept as defined in the hierarchy of systems specification[6].*

However, changing the external structure of a component as well as adding/removing a coupled component requires the compliance of impacted influencees as well as updating network structure while ensuring that this whole structure change sequence cannot be interrupted. To achieve this goal, a *synchronization mechanism* can be used.

Definition 4.2. Lock synchronization of structure changes *is depicted in Figure 2 for two components. Component* a *aims at achieving a structure change impacting the structure of influencee component* b. *This follows the sequence:*

1. Component a sends a *query* message to component b to change structure,
2. Component b changes self-structure to comply with the new structure aimed by component a,
3. Component b sends a *done* message to component a,
4. Finally, component a changes self-structure and updates network structure.

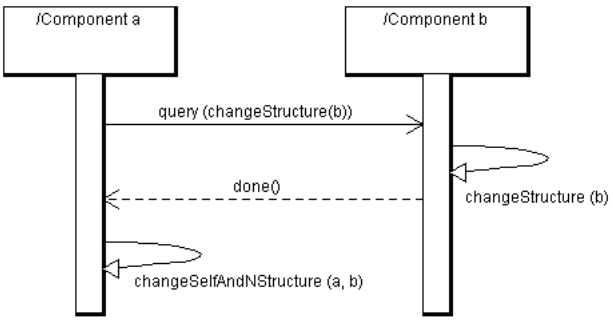

Figure 2: UML interaction diagram for request/done synchronization protocol for dynamic structure change between component a and component b.

In the next sections this synchronization protocol is applied to addition and deletion operations.

4.3.2 Addition operations

Example 4.1. A component a adds *query/done coupling* with a component b.

There are no query/done couplings between component a and component b. However, as all dynamic structure components, components a and b have existing query/done interfaces. As described in Figure 3, component a needs first to self-add a query outgoing coupling with component b. After, component a requests component b to self-add a done outgoing coupling. Finally, component b confirms the addition operation sending a *done* confirmation to component a.

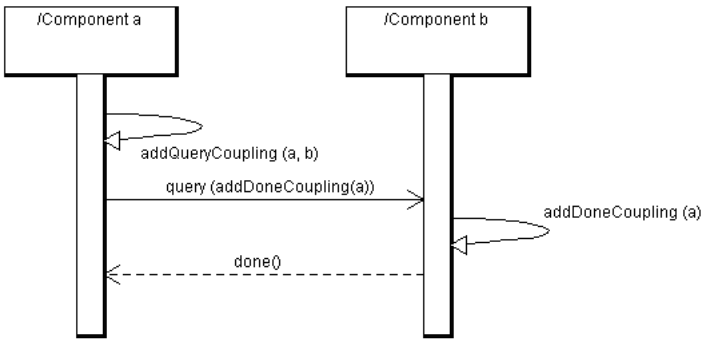

Figure 3: UML interaction diagram for request/done coupling addition between component a and component b.

Example 4.2. A component a adds *an outgoing state coupling* with a component b.

As described in Figure 4, component a needs first to request component b to add corresponding state input. After, component b confirms the addition operation sending a *done* confirmation to component a. Finally, component a self-adds corresponding output and outgoing state coupling with component b.

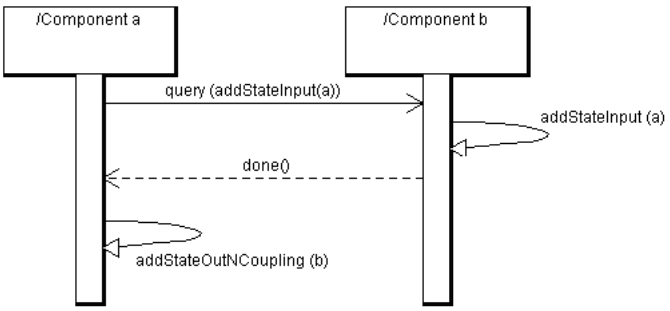

Figure 4: UML interaction diagram for outgoing state coupling addition between component a and component b.

4.3.3 Deletion operations

Example 4.3. Mutual deletion of query/done outgoing couplings between components a and b.

As described in Figure 5, As for outgoing state coupling addition, component a needs first to request component b to delete corresponding input. Once component a receives the done confirmation from component b, it self-deletes its output and outgoing coupling to component b. For symmetry reasons, component b self-deletes corresponding done output and outgoing couplings to component b.

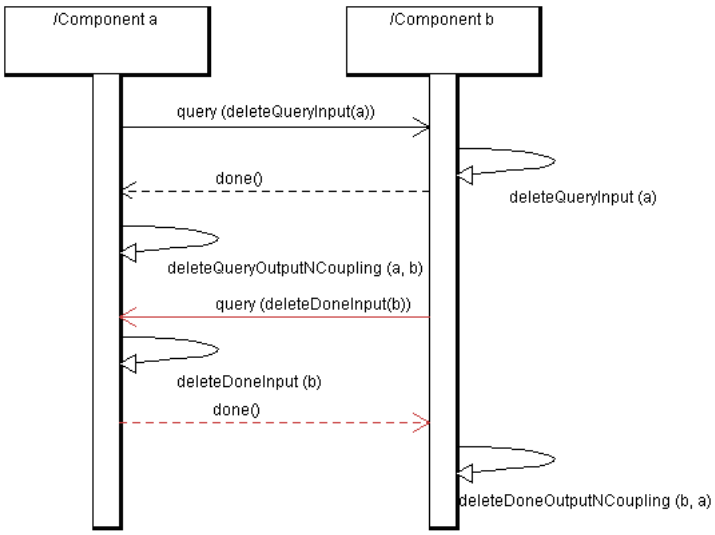

Figure 5: UML interaction diagram for mutual request/done coupling deletion between component a and component b.

Example 4.4. A component b deletes itself.

As described in Figure 6, component b queries first all its influencees (component a) to self-delete their inputs from component b. After this deletion, component a sends a done message after which component b deletes all its outgoing couplings and outputs to component a. After, component a follows the same protocol to remove its outputs and outgoing couplings to component b. Finally, component b deletes itself.

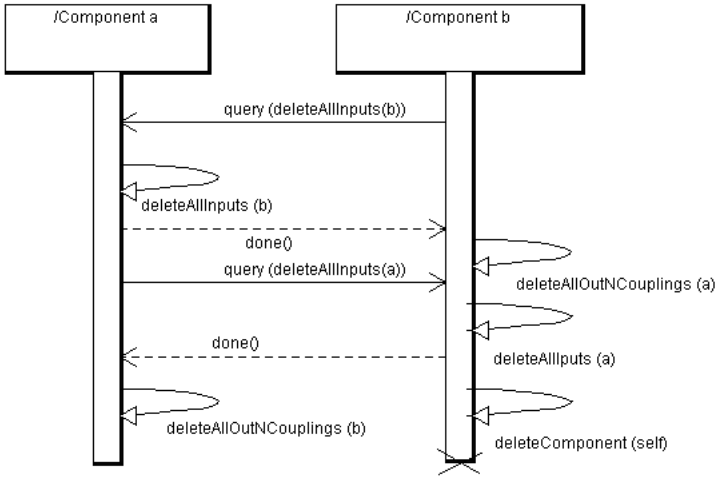

Figure 6: UML interaction type diagram of request/response protocol for component b self-deletion.

4.3.4 Independence of local structure changes

Proposition 4.2. *Based on a query/done message exchange protocol, a* structure change lock *is a synchronization mechanism ensuring: (i) no interferences between external structure changes, and (ii) structure consistency at network level.*

Lemma 4.1. *Local external structure changes do not interfere.*

Proof. At each global transition, only one imminent component is selected: $d^* = Select(IMM)$, with imminent components $IMM = \{\sigma_d \,|\, d \in D \wedge \sigma_d = ta(s)\}$. The basic lock synchronization mechanism between two dynamic structure components (cf. Figure 2) follows a zero time advance sequence. First, imminent component i^* is selected to send a *query message* to an influencee $j \in I_{i^*}$. The latter achieves an external structure change transition $(M'_{ext,j}, s'_j) = \tau_{ext,j}(M_{ext,j}, \delta_{ext,j}(s_j, e_j, x_j))$ and schedules an internal transition $\delta_{int,j}(s_j)$. At the same time, if component j receives another *query message*, as in classic $DEVS$, $\delta_{ext,j}(\delta_{int,j}(s_j), 0, x_j)$, internal transition $\delta_{int,j}(s_j)$ is executed first, and component j^* sends the *done message* to initial querying component $i \in I_{j^*}$, which executes its external structure transition function. The latter first changes the external structure of component $i \in D$ as $(M'_{ext,i}, s'_i) = \tau_{ext,i}(M_{ext,i}, \delta_{ext,i}(s_i, e_i, x_i))$ and finally updates network structure based on new structures $M'_{ext,i}$ and $M'_{ext,j}$, *i.e.*, $N' = \tau_{ext,i}(N, \delta_{ext,i}(s_i, e_i, x_i))$. □

Lemma 4.2. *Local internal structure changes do not interfere.*

Proof. Obvious from the definition of internal models (cf. Definition 2.4). □

Theorem 4.2. *Local dynamic structure changes do not interfere.*

Proof. Obvious from Lemma 4.2 and Lemma 4.1. □

Theorem 4.3. *Considering an initial network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$*, where each component* $d \in D$ *is a basic dynamic structure component* DYS-DEVS$_d$*, and where there are dynamic single local points of control of* structure changes of models $M_d = (M_{ext,d}, M_{int,d})$ *, the set of networks* $\mathcal{N}$ *is equivalent to a* resultant DYS-DEVS $= (\mathcal{M}, \mathcal{S}, \tau)$.

Proof. As local structure changes do not interfere (*cf.* Theorem 4.2),

1. For each non-created component $d \in D \cap D'$, $\tau(..., M_d, s_d, \dots) = (..., \tau_d(M_d, s_d), ...) = N'$, □
2. For each new component $d \in (D' - D)$, initialized to initial state $s_{0,d}$: $\tau(..., M_d, s_d, \dots) = (..., \tau_d(M_d, s_d), ...) = (N', (..., s_{0,d}, ...))$.

4.3.5 Closure under coupling

Theorem 4.4. DYS-DEVS *formalism is closed under coupling, i.e., considering an initial network* $N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$*,where each component* $d \in D$ *is a basic dynamic structure component* DYS-DEVS$_d$*, and where there are dynamic single points of control of* structure changes*, the set of networks* $\mathcal{N}$ *is equivalent to a* resultant DEVS $= (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$.

Proof. Let the *time remaining to the next event* $\sigma_d = ta_d(s_d) - e_d$, with $ta_d(s_d)$ the *time advance* of a component model M_d, s_d its current state, e_d its *time elapsed time since the last event.* Then, the time advance of the resultant is $ta(s) = min\{\sigma_d, | d \in D\}$.

External transitions $s' = \delta_{ext}(s, e, x)$ at resultant level can be expressed at component level by:

$$s'_d = \begin{cases} \delta_{ext,d}(s_d e_d, x_d) & if \quad d \in D \cap D',\ N \in I_d,\ x_d \neq \emptyset \\ s_{d,0} & if \quad d \in (D - D') \\ s_d & otherwise \end{cases}$$

Internal transitions $s' = \delta_{int}(s)$ at resultant level can be expressed at component level by:

$$s'_d = \begin{cases} \delta_{ext,d}(s_d e_d, x_d) & if \quad d \in D \cap D',\ d \in I_{d^*},\ x_d \neq \emptyset \\ \delta_{int,d}(s_d) & if \quad d \in D \cap D',\ d^* = d \\ s_{d,0} & if \quad d \in (D - D') \\ s_d & otherwise \end{cases}$$

□

4.3.6 Legitimacy

General closure under coupling yields a *DEVS* which is not necessarily legitimate - there could be a loop of components that activate each other without advancing time (each having a transient (zero-time) state to output and then waiting for input). Hence, the same situation can hold for the non dynamic structure part of a *DYS-DEVS*, considering dynamic structure operations at network level. Therefore, conditions of *DYS-DEVS* legitimacy have to be exposed.

Theorem 4.5. *A* DYS-DEN $= (\mathcal{N}, \mathcal{S}, \tau)$ *is legitimate* (i.e., *corresponding dynamic structure operations always terminate) if each network* $N \in \mathcal{N}$ *is legitimate, the resultant being also legitimate.*

Proof. A *DEVS M* is *legitimate* under following conditions[6]:

1. M is finite (partial state set S is finite): Every cycle in the state diagram of internal transitions δ_{int} contains a non-transitory state $ta(s) > 0$ (necessary and sufficient condition).

2. M is infinite: There is a positive lower bound on the time advances, *i.e.*, $\exists b \, \forall s \in S$, $ta(s) > b$ (sufficient condition).

Although, it has been proved in Theorem 4.2 that confluent dynamic structure operations do not interfere, for sake of simplicity it is assumed here that there are no confluent dynamic structure operations for each network $N \in \mathcal{N}$. Then, at each time, each component can be concerned by only one dynamic structure operation.

Also, *it is assumed that each network* $N \in \mathcal{N}$ *is legitimate,* i.e., *each corresponding resultant does not get stuck in time and specifies a well-defined dynamic system.*

In a network, among basic dynamic structure operations, self-deletion (*cf.* Example 4.4) consists of *9* consecutive internal and external transitions. It is the longest sequence of basic dynamic structure operations. Each other basic dynamic structure operation terminates in fewer (zero-time) transitions. To show this, both internal and external dynamic structure changes can be considered. Being independent, for one component $d \in D$, changing its internal model $M_{int,d}$ consists merely of *1* transition: $(M'_{int,d}, s'_d) = \tau_{int,d}(M_{int,d}, \delta_{int,d}(s_d))$. Depending on the interaction of one requesting component $i \in D$ and one answering component $j \in D$, changing external model $M_{ext,i}$ implies changing external model $M_{ext,j}$. This consists of a basic *lock synchronization message exchanges* (*cf.* Figure 2), *i.e.*:

1. Two transitions for component $i \in D$:
   (a) $\delta_{int,i}(\mathsf{request})$,
   (b) $(M'_{ext,i}, s'_i) = \tau_{ext,i}(M_{ext,i}, \delta_{ext,i}(\mathsf{request}, 0, \mathsf{done}))$.
2. Two transitions for component $j \in D$:
   (a) $\delta_{int,j}(\mathsf{done})$,
   (b) $(M'_{ext,j}, \mathsf{done}) = \tau_{ext,j}(M_{ext,j}, \delta_{ext,j}(s_j, e_j, \mathsf{request}))$.

Hence, changing external models consists of *4* zero-time transitions. Finally, self-deletion of a component $i \in D$ consists of summing the following steps:

1. Mutually changing both external models $M_{ext,j}$ with $i \in I_j$ (removing corresponding input/output of component $j \in D$ and outgoing couplings to component $i \in D$) and external model $M_{ext,i}$ (removing corresponding input/output of component $i \in D$ and outgoing couplings to component $j \in D$) - *8* zero-time transitions;
2. Self-deletion finally consisting of the deletion of internal model $M_{int,i}$ (including the update of network structure) - *1* zero-time transition.

Considering a DYS-DEN $= (\mathcal{N}, \mathcal{S}, \tau)$, *where each network* $N \in \mathcal{N}$ *is legitimate, corresponding dynamic structure operations always terminate individually in less than* 9 zero-time transitions, *then the resultant is legitimate.* □

5 Conclusion and perspective

Using single point encapsulated control functions this article proves that a fully modular decentralization of dynamic structure systems is possible while keeping the approach simple enough. Futhermore, a new way of integrating formalisms and specifying dynamic structure discrete event systems is proposed.

The goal of this work is really to preserve and to participate to the diversity of the dynamic structure research field. Modeling the interactions between structure and state dynamics is not easy. However, this should not be an excuse for constraining too much the control mechanisms. Otherwise, it is well known that too much constraints kills diversity and usually leads to the sterilization of a field. It is hoped that this contribution will be the occasion to share new perspectives.

A first perspective concerns the implementation of abstract simulators to automate request/done message exchange protocol. A second perpective concerns the generalization of single points of control to multiple points of control allowing many structure changes to occur in parallel.

References

[1] F. J. Barros. Modelling Formalisms for Dynamic Structure Systems. *ACM*

Transactions on Modelling and Computer Simulation (TOMACS), 7:501 – 515, 1997.

[2] Adelinde Uhrmacher. Dynamic structures in modeling and simulation: a reflective approach. *ACM Trans. Model. Comput. Simul.*, 11(2):206–232, 2001.

[3] Xiaolin Hu, Bernard P. Zeigler, and Saurabh Mittal. Variable structure in devs component-based modeling and simulation. *Simulation*, 81(2):91–102, February 2005.

[4] F. J. Barros. Modeling and simulation of dynamic structure heterogeneous flow systems. *SIMULATION*, 78(1):18–27, January 2002.

[5] Fernando J. Barros and Bernard P. Zeigler. Model interoperability in the discrete event paradigm: Representation of continuous models. *Modeling and Simulation Theory and Practice*, pages 103–126, 2003.

[6] H. Praehofer B. P. Zeigler, T. G. Kim. *Theory of Modeling and Simulation.* Academic Press, 2000.

[7] Fernando J. Barros. A formal representation of hybrid mobile components. *Simulation*, 81(5):381–393, 2005.

[8] G.H. Kim and T.G. Kim. Framework for modeling/simulation of mobile agent systems. In *Proceedings of the 2000 Conference on AI, Simulation and Planning in High Autonomy Systems*, pages 53–59, 2000.