Extended dynamic structure DEVS

Olaf Hagendorf
Hochschule Wismar
24 PUBLICATIONS **91** CITATIONS
SEE PROFILE

Thorsten Pawletta
Hochschule Wismar - University of Applied Sciences
103 PUBLICATIONS **309** CITATIONS
SEE PROFILE

Christina Deatcu
Hochschule Wismar
23 PUBLICATIONS **38** CITATIONS
SEE PROFILE

In: Proc. of 21th European Modelling & Simulation Symposium,
Puerto de la Cruz, Spain, September, 2009, Volume 1, 36-45

EXTENDED DYNAMIC STRUCTURE DEVS

Olaf Hagendorf [(a)], **Thorsten Pawletta** [(b)], **Christina Deatcu** [(c)]

[(a)] Liverpool John Moores University, School of Engineering, UK
[(a, b, c)] Hochschule Wismar, University of Applied Sciences: Technology, Business and Design, Germany

[(a)] oh@ibhagendorf.de, [(b)] thorsten.pawletta@hs-wismar.de, [(c)] christina.deatcu@hs-wismar.de

ABSTRACT
Since the first publication of DEVS, the formalism was enhanced and many extensions have been introduced. Every extension holds some advantages over the other, e.g. Parallel DEVS generalizes the specification and handling of concurrent events, DEVS with Ports enables a more structured modeling and Dynamic Structure DEVS introduces dynamic structure changes at coupled model level during simulation time. The extensions have one joint attribute: they are extending the Classic DEVS formalism and don't incorporate the advantages of each other. Hence, the decision on one DEVS extension inhibits the use of advantages of another one. This lack leads to the idea of a merging formalism to combine the advantages of different approaches. The Extended Dynamic Structure DEVS combines the Classic DEVS with some of the existing extensions: Parallel DEVS, Dynamic Structure DEVS and DEVS with Ports.

Keywords: Discrete Event Simulation, DEVS, DSDEVS, PDEVS, EDSDEVS

1. INTRODUCTION

The DEVS formalism was first introduced by Zeigler (Zeigler 1976) in the 1970s. In (Zeigler et.al. 2000) the authors classify this formalism, position and compare it with other, more established modeling and simulation formalisms. Several international research groups are working on the DEVS formalism and are regularly publishing results at the annual DEVS Symposium at Spring Simulation Conferences, European Modeling and Simulation Symposia and others. Wainer (Wainer 2009) maintains a list of available DEVS tools. The DEVS formalism is, in contrast to other modeling and simulation formalisms, not very widely used in industrial practice. This situation persists despite the fact that the theory is a well-founded, general formalism. It can only be assumed that one reason of the marginal acceptance is the type of available software tools (Pawletta et.al. 2006).

There are several publications to extend the application field or to ease the use of DEVS e.g. Parallel DEVS generalizes the specification and handling of concurrent events, DEVS with Ports enables a more structured modeling and Dynamic Structure DEVS introduces dynamic structure changes at coupled model level during simulation time and significantly eases the modeling of larger real systems. The extensions have one joint attribute: they are based on the Classic DEVS formalism and extending it in a specific direction. Hence, the decision on one DEVS extension inhibits the use of advantages and application fields of another one. This lack leads to the idea of a merging formalism to combine the advantages of different approaches and widen the application field of the resulting formalism.

The Classic DEVS formalism with the formal modeling concept and simulation algorithms is introduced in chapter 2. After a short introduction of a few DEVS extensions, three of them are described in detail in chapter 3. The fusion of Classic DEVS with the introduced extensions to the new Extended Dynamic Structure DEVS approach is presented with formal concept, simulation principles and algorithms in chapter 4. The conclusions in chapter 5 complete this contribution.

2. CLASSIC DEVS

DEVS is a modular, hierarchical modeling and simulation formalism. Every DEVS model can be described by using two different model types, atomic and coupled. Both model types have an identical, clearly defined interface through input and output ports. An atomic model describes the behavior of a non-decomposable entity via input/output events and event driven state transition functions. A coupled model describes the structure of a more complex model through the aggregation of several entities and their couplings. These entities can be atomic models as well as coupled models. The DEVS formalism consists of two parts: (i) a formal DEVS model definition and (ii) simulator algorithms.

2.1. Formal Concept

The formal Classic DEVS description defines coupled and atomic models as a combination of sets and functions. The description of an atomic model is a 7-tuple (Zeigler et.al. 2000):

am = (X, Y, S, δ_{ext}, δ_{int}, λ, ta)

- *X*, *Y* and *S* specify the sets of discrete inputs, outputs and internal states.

- δ_{ext}: $Q \times X \rightarrow S$ where $Q = \{(s,e) \mid s \in S, 0<e<t_{next}\}$
  The external state transition function δ_{ext} handles external input events.
- δ_{int}: $S \rightarrow S$
  The internal state transition function δ_{int} establishes a new internal state.
- λ: $S \rightarrow Y$
  The output function λ generates an output event depending on the internal state *S*.
- *ta*: $S \rightarrow \Re_0^+ \cup \infty$
  The time advance function *ta* schedules the time of the next internal event after each state transition.

Figure 1 shows the dynamic behavior of an atomic model.

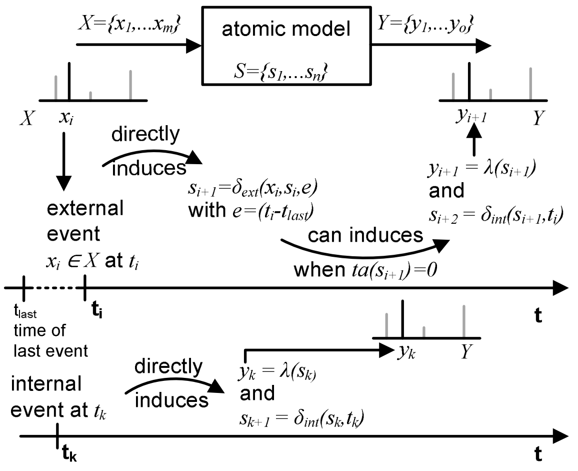


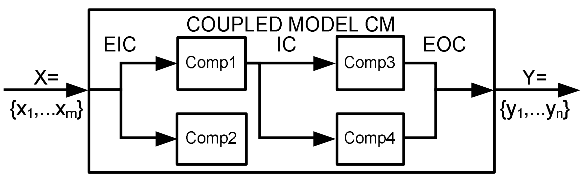
Fig. 1 Dynamic Behavior of an Atomic Model

The description of a coupled model is a 9-tuple (Zeigler et.al. 2000):

CM = (d_n, X, Y, D, { M_d }, EIC, EOC, IC, SELECT)

- d_n specifies the name of the coupled model.
- *X* and *Y* specify the sets of discrete inputs and outputs.
- *D* specifies the set of sub component names.
- $M_d \mid d \in D$
  M_d is the model of the sub component *d*
- *EIC*, *EOC* and *IC* are the sets of external input, external output and internal couplings.
- The *SELECT* function prioritizes concurrent internal events of sub components.

Figure 2 depicts the relations of the elements of a Classic DEVS coupled model.



EIC	= {{CM.X,Comp1.X} {CM.X,Comp2.X}}
IC	= {{Comp1.Y,Comp3.X} {Comp1.Y,Comp4.X}}
EOC	= {{Comp3.Y,CM.Y} {Comp4.Y,CM.Y}}
D	= {“Comp1“, “Comp2“, “Comp3“, “Comp4“}

Fig. 2 Coupled Model Elements

The Classic DEVS approach supports the specification of behavioral system dynamics in atomic systems and the specification of static component aggregations in coupled systems. It is not possible to describe structural system dynamics at the coupled model level, i.e. the deletion or creation of components and couplings or changes of interfaces, although all necessary structural information is also available during simulation time. The only possibility to realize a structural system dynamic is to specify it with logical constructs at the atomic model level. However, this removes the advantages of reusability and model clarity and increases modeling complexity.

2.2. Classic DEVS Simulation

Beside the formal definition the second part of the Classic DEVS formalism is the description of abstract simulator algorithms for the execution of DEVS models. The algorithms are named abstract because they are implemented as a general pseudo code. The abstract simulator has a modular, hierarchical structure matching exactly the modular, hierarchical structure of a DEVS model. A DEVS model can be directly transformed into an executable simulator model using abstract simulator elements e.g. as shown in (Praehofer 1992; Zeigler et.al. 2000).

The abstract simulator approach consists of three different elements namely root coordinator, coordinator and simulator. Each atomic model is associated with a simulator element and each coupled model is associated with a coordinator element. The root coordinator is added to that structure as topmost ruling entity.

3. DEVS EXTENSIONS

Extensions of the Classic DEVS formalism increase the classes of system models that can be represented by DEVS. Several DEVS extensions are introduced e.g. in (Barros 1996; Chow et.al. 1994; Hagendorf et.al. 2006; Pawletta et.al. 1996; Praehofer 1992; Uhrmacher et.al. 1994; Wainer 2009; Zeigler et.al. 2000). An incomplete list of DEVS extensions recently presented is:

- DEVS with Ports: The port extension adds additional input and output ports to models.
- Parallel DEVS: Parallel DEVS (PDEVS) considers concurrent transition events.
- Dynamic Structure DEVS: Dynamic Structure DEVS (DSDEVS) enables changes during a simulation run. Several partial very different approaches exist. Dynamic structure extensions introduced by Barros (Barros 1996) and Pawletta (Pawletta et.al. 1996) keep the general structure of Classic DEVS modeling and simulation with additions to coupled model definitions but unchanged atomic model definitions. Other dynamic structure extensions e.g. an agent based DEVS (Uhrmacher et.al. 1994) introduce more extensive modifications.
- DSDEVS-hybrid: The extension of discrete state changes by continuous state changes as introduced by DSDEVS-hybrid enables a complete new

application field and can ease the modeling of several problems (Deatcu et.al. 2009).

- Real Time DEVS: The DEVS model is executed in real time rather than in model time. The time advance function delivers time intervals which allow uncertainty when an internal event has to take place.

The next sections introduce some of these DEVS extensions in more detail. They are used as basis of the subsequently introduced, unifying DEVS formalism.

3.1. DEVS with Ports

The introduction of ports into the Classic DEVS formalism makes modeling easier and the representation of information flow more clearly (Zeigler et.al. 2000). In Classic DEVS each model has only one single input and one single output port. All events are received and sent through these ports. With the port extension, a model has several input and output ports each dedicated for a specific task i.e. event type. A model can have several output ports which can be connected to input ports of other models as shown in figure 3. Hence, each event can use a dedicated, well defined routing path. The modeling becomes more structured; a model can become clearer and better understandable through differentiated interfaces.

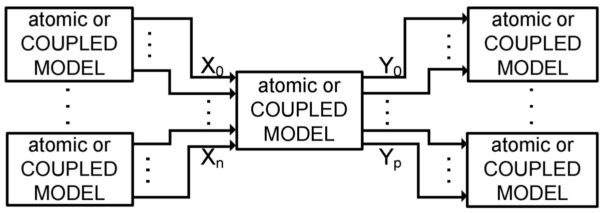


Fig. 3 Model with Multiple Input and Output Ports

The formal description of Classic DEVS with Ports largely remains the same except the extended definitions of *X, Y* for atomic and coupled models (Zeigler et.al. 2000):

$X = \{(p,v) \mid p \in \text{InputPorts}, v \in X_p\}$

$Y = \{(p,v) \mid p \in \text{OutputPorts}, v \in Y_p\}$

- *p* is the input or output port of the model
- *v* is a discrete value
- X_p and Y_p specify the sets of discrete inputs and outputs at port *p*

Whereas in Classic DEVS the coupling definitions consist of a sub model name as destination and source, respectively, for EIC and EOC and of a pair of sub model names for IC, the port extension necessitates a coupling definition extension, too:

- EIC = { (*input_port*, *d.input_port*) | *input_port* ∈ *InputPorts*, $d \in D$, *d.input_port*∈*InputPorts of* M_d }
- IC = { (d_i*.output_port*, d_k*.input_port*) | $d_i, d_k \in D$, d_i*.output_port*∈ *OutputPorts of* M_{d_i}, d_k*.input_port* ∈*InputPorts of* M_{d_k}, *i<>k* }
- EOC = { (*d.output_port*, *output_port*) | *d.output_port* ∈*OutputPorts of* M_d, $d \in D$, *output_por* ∈*OutputPorts*}

3.2. Parallel DEVS

Parallel DEVS (PDEVS) was introduced by Chow (Chow et.al 1994). It adds new elements and functions to the Classic DEVS formalism. It allows all imminent components to be activated simultaneously and enables sending their output to other components at the same time concurrently. Multiple outputs are combined in a bag which is sent as a whole to a model's external state transition function. A bag is similar to a set, containing an unordered set of elements, but allows multiple occurrences of an element. In Classic DEVS by contrast events are handled individually. In a PDEVS simulator (Zeigler et.al. 2000) during the *-message handling first all outputs are established before calling external and internal state transition functions. Each receiving component is responsible for examining and interpreting its combined inputs in the correct order. PDEVS gives the atomic model more control over the handling order of concurrent external and internal events. In Classic DEVS a super-ordinate component, the coupled model, is responsible for the execution order of concurrent internal events of different sub components using the *select* function. In PDEVS the order of simultaneous events is locally controllable at atomic model level with an additional, third state transition function, the confluent transition function δ_{con}. Hence, it merges the decision logic of execution order of concurrent events with the event handling functions at a same level.

According to the extensions of PDEVS an atomic model is defined by the following 8- tuple (Chow et.al. 1994):

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

- *X*, *Y* and *S* specify the sets of discrete input events, output events and sequential states.
- $\delta_{ext}: Q \times X^b \rightarrow S$ where X^b is a bag covering elements of *X* and $Q = \{ (s,e) \mid s \in S, 0<e<t_{next} \}$
  The external state transition function δ_{ext} handles a bag covering external inputs $X^b = \{x_i \mid x_i \in X\}$.
- $\delta_{int}: S \rightarrow S$
  The internal state transition function δ_{int} establishes a new internal state.
- $\delta_{con}: S \times X^b \rightarrow S$
  The confluent transition function δ_{con} handles the execution sequence of δ_{int} and δ_{ext} functions during concurrent external and internal events.
  - The confluent function definition $\delta_{con}(s, X^b) = \delta_{ext}(\delta_{int}(s), 0, X^b)$ with $\delta_{ext}(s, e, X^b)$ is equivalent to the Classic DEVS behavior with a higher prioritized handling of internal events.
  - The alternative confluent function defintion $\delta_{con}(s, X^b) = \delta_{int}(\delta_{ext}(s, ta(s), X^b))$ with $\delta_{int}(s)$ first handles external events.
  - The execution of the confluent function with an empty bag $\delta_{con}(s, null)$ calls directly the internal transition function δ_{int}.
- $\lambda: S \rightarrow Y^b$ where Y^b is a bag covering elements of *Y*
  The output function λ generates a bag covering

outputs $Y^b = \{ y_i \mid y_i \in Y \}$ depending on the internal state S.

- $ta: S \rightarrow \Re_0^+ \cup \infty$
  The time advance function *ta* schedules the time of the next internal event after each state transition.

The definition of a coupled model for PDEVS is the same as for Classic DEVS except for the absence of the *select* function (Zeigler et.al. 2000):

CM = (d_n, X, Y, D, { M_d }, EIC, EOC, IC)

The execution of a PDEVS model is carried out similarly to Classic DEVS with some changed details in the message handling (Zeigler et.al. 2000).

3.3. Dynamic Structure DEVS

Several approaches extend the Classic DEVS to Dynamic Structure DEVS (DSDEVS). Barros (Barros 1996) and Pawletta (Pawletta et.al. 1996) introduce two DSDEVS variants with an extension of the coupled model definition while the atomic model definition remains unchanged. With theses extensions the coupled model is able to change its structure during simulation time. Uhrmacher (Uhrmacher et.al. 1994) introduces an agent based approach. It defines extensions for both atomic and coupled systems. Another approach is Cell-DEVS, a combination of cellular automata with the DEVS formalism where each cell consist of a single DEVS model (Wainer 2001).

The different types of extensions are carried out due to different application fields or problem definitions e.g. a typical Cell-DEVS application field is social and environmental modeling and simulation. The approaches of Barros and Pawletta are extending the classic formalism without changing its overall principle and thus without changing the general application field of Classic DEVS. The DSDEVS approach of Pawletta enables several options to specify structural dynamics:

- Creation, destruction, cloning and replacement of sub components
- Exchange of a sub component between two coupled models
- Changing coupling definitions of a coupled system

The DSDEVS approach extends the coupled model definition but the atomic model definition stays unchanged. During the simulation time a coupled model can change its structure. Each structure can be seen as a structure state s_i with $s_0, s_1, ..., s_n \in S_{DS}$. A structure state s_i describes all structure relevant elements of a coupled model. Additionally a structural state set H_{DS} can store further structure information e.g. the number of structure changes at the present time or the current structure number. External or internal events, handled by the additional state transition functions $\delta_{x\&s}$ and δ_{int} at coupled model level, induce structure state changes and as a result model structure changes. This dynamic structure extension of Classic DEVS was developed with a regard to hybrid systems, i.e. systems with continuous and discrete event dynamic. In the following only the relevant aspect for discrete event systems are taken into account.

A DSDEVS coupled model is defined by the following 6-tuple (Pawletta et.al. 1996):

CM_{DS} = (d_{ds}, S_{DS}, $\delta_{x\&s}$, δ_{int}, λ, ta)

- d_{ds} specifies the name of the coupled model.
- According to the above definition of a coupled model, its structure consists of sets of sub components and coupling relations. Structure changes mean modifications of these sets. Obviously, the sets of sub systems and coupling relations could be interpreted as a structure state. The set of sequential structure states $\{s_0, s_1, ..., s_n\} = S_{DS}$ defines all structure variants of the variable structure coupled model CM_{DS}. Structure state changes can be induced by handling external or internal events of the coupled model itself or by state events i.e. output events of subordinated components. A structure state is defined by a 9-tuple:
  s_i = (X, Y, H_{DS}, D, { M_d }, EIC, EOC, IC, select)
  - *X* and *Y* specify the sets of discrete input and output events. The sets exactly match the sets *X* and *Y* in Classic DEVS. As an extension of DSDEVS the coupled model can directly handle external input events and can create external output events itself.
  - The set H_{DS} represents additional structure related state variables. They are equivalent to the state variable set *S* of an atomic model.
  - *D* specifies the set of sub component names.
  - $M_d \mid d \in D$
    M_d is the model of the sub component *d* of the coupled model CM_{DS}. The set $\{ M_d \}$ defines all sub components of CM_{DS}.
  - *EIC*, *EOC* and *IC* are the external input, external output and internal couplings.
  - The function *select* prioritizes concurrent internal events of the coupled model itself and its sub components.
- $\delta_{x\&s}$: $Q_{DS} \times X \rightarrow H_{DS}$ where $Q_{DS} = \{(h,e) \mid h \in H_{DS}, 0<e<t_{next}\}$
  The external and state transition function $\delta_{x\&s}$ handles external input events and state events i.e. output events of sub components. However it is unreasonable to make changes in the set of sub components or the coupling relations by this function directly. This could lead to ambiguous event handling because external events could simultaneously influence the dynamic of sub components and the structure state. Consequently the $\delta_{x\&s}$ function is only allowed to modify structure related state variables in the set H_{DS}.
- δ_{int}: $S_{DS} \rightarrow S_{DS}$
  The internal transition function δ_{int} changes the structure state s_i to s_{i+1} and as a result induces a structure change of CM_{DS}. The execution of output function λ and internal transition function δ_{int} is induced by a time driven internal event.
- λ: $S_{DS} \rightarrow Y$

The output function λ generates output events depending on the state S_{DS}.

- $ta: S_{DS} \rightarrow \Re_0^+ \cup \infty$
  As with the dynamic of atomic models, internal events are scheduled by the time advance function *ta*. After each state transition the next internal event is established by the time advance function.

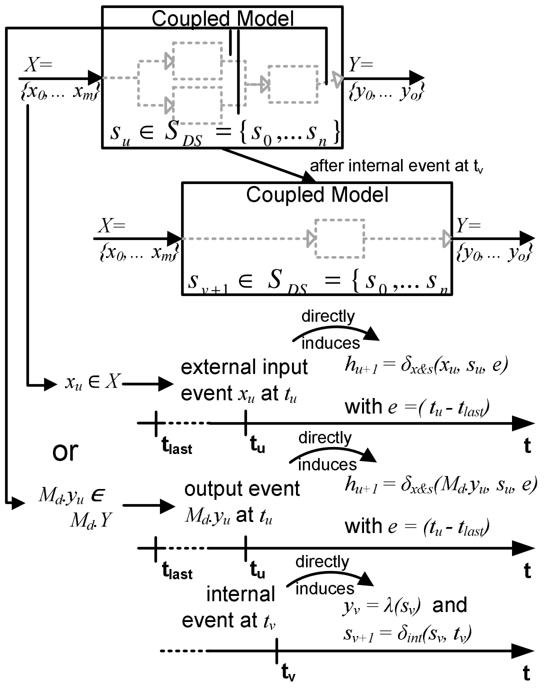


Fig. 4 Dynamic Behavior of a DSDEVS Coupled Model

The dynamic behavior of an atomic model is identical to the behavior in Classic DEVS. Figure 4 shows the dynamic behavior of a dynamic structure coupled model. The figure depicts two external input events and one internal event. Reasons for an input event handling can be an external input event at the input port of the coupled model itself or an external output event at the output port of a sub component M_d of the coupled model. The handling of both events by the coupled model is identically. As a result of an event the structure related state variable set H_{DS} can be changed and with the concluding call of the time advance function an immediate internal event can be induced. An internal event is handled by a coupled model similar to the internal event handling of an atomic model, i.e. the event handling can induce a change of the state sets S and S_{DS}, respectively.

4. EXTENDED DYNAMIC STRUCTURE DEVS

Chapters 3 and 4 introduce the Classic DEVS formalism and several DEVS extensions. This work aims to bring together all introduced approaches and to combine their advantages and application fields. In (Zeigler et.al. 2000) a first step into this direction is undertaken, the introduced PDEVS formalism is a combination of the original PDEVS and DEVS with Ports. The Extended Dynamic Structure DEVS (EDSDEVS), proposed here, combines the extensions: Classic DEVS with PDEVS, DSDEVS and DEVS with Ports. The selection of the extensions is carried out to ensure the preservation of the generic modeling and simulation principles of Classic DEVS. The fusion results in a DEVS formalism with the following main characteristics:

- Modular, hierarchical and dynamic structure modeling and simulation formalism,
- Formal description by sets and functions,
- Exact definition of simulation algorithms,
- Dynamic behavior description in atomic models,
- Dynamic structure description in coupled models,
- Exact behavior definition of concurrent events,
- Substantial similarity between real system and model.

The next sections focus on the formal concept of EDSDEVS modeling with formal descriptions, dynamic behavior descriptions and introduction of the simulation concept with abstract simulator algorithms.

4.1. Formal Concept

The EDSDEVS formal descriptions of coupled and atomic models as a combination of sets and functions are structured similar to the Classic DEVS formal description. The EDSDEV atomic model am_{EDS} is defined as an 8- tuple:

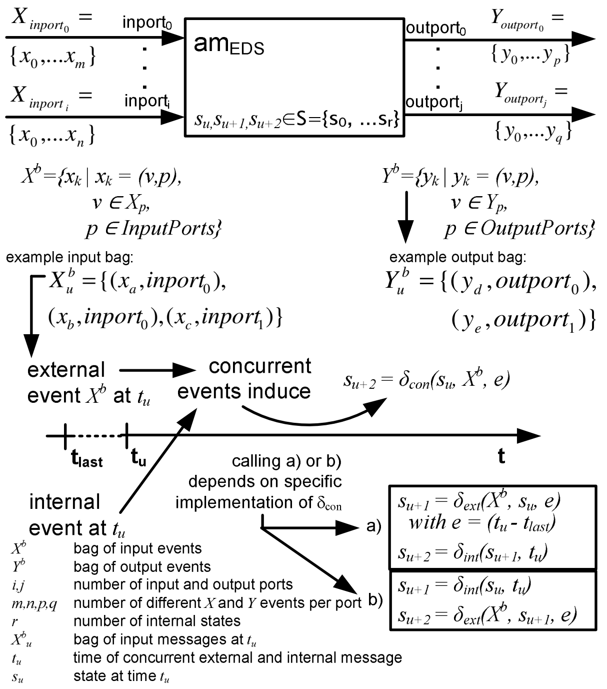
$am_{EDS} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$

- $X = \{(p,v) \mid p \in \text{InputPorts}, v \in X_p\}$
  $Y = \{(p,v) \mid p \in \text{OutputPorts}, v \in Y_p\}$
  The definitions of both sets are identical to the definitions in DEVS with Ports.
- *S* specifies the set of internal states and is identical to internal state set *S* of a Classic DEVS atomic model.
- $\delta_{ext}: Q \times X^b \rightarrow S$ with $X^b = \{x_i \mid x_i = (p,v),\ p \in InputPorts,\ v \in X_p\}$ and $Q = \{(s,e) \mid s \in S,\ 0 < e < t_{next}\}$
  The external state transition function δ_{ext} handles a bag covering external inputs. Each input consists of a pair of a discrete input $v \in X_p$ and an input port $p \in InputPorts$. The set X_P is the set of discrete inputs at port *p* and *InputPorts* is the set of input ports of model am_{EDS}. The function δ_{ext} can induce an internal event with a rescheduling of the time of the next internal event. This extended definition of δ_{ext} is a fusion of the δ_{ext} definitions of PDEVS and DEVS with Port.
- $\delta_{int}: S \rightarrow S$
  The internal state transition function δ_{int} can establish a new internal state. The execution of output function λ and internal state transition function δ_{int} is induced by a time driven internal event. The time of an internal event is established by the time advance function *ta*. The definition is identical to the definition in Classic DEVS.
- $\delta_{con}: S \times X^b \rightarrow S$
  The confluent transition function δ_{con} handles the execution order of δ_{int} and δ_{ext} functions during

concurrent external and internal events. In spite of the same function signature $\delta_{con}(s, X^b)$ the parameter X^b is different to that in the PDEVS definition as described in section 3.2. Anyhow the three δ_{con} definitions from there also apply here. This extended definition of δ_{con} is based on the PDEVS δ_{con} function definition. Unlike in PDEVS the function has to handle a bag covering inputs, each consisting of a discrete input and input port pair.

- $\lambda: S \rightarrow Y^b$ whit $Y^b = \{y_i \mid y_i = (p, v), \; p \in OutputPorts, v \in Y_p\}$
  The output function λ can generate a bag covering outputs Y^b. In spite of the same function signature $Y^b = \lambda(s)$ the function result Y^b is different to that in the PDEVS definition as described in section 3.2. The function result is a bag covering outputs $Y^b = \{ y_i \mid y_i = (p, v) \}$ each consisting of a pair of discrete output $v \in Y_p$ and output port $p \in OutputPorts$. The set Y_P is the set of discrete outputs at port p and *OutputPorts* is the set of output ports of model *am*. If and which outputs are generated depends on the internal state S. This extended definition of λ is based on the PDEVS λ function definition. Unlike in PDEVS the function generates a bag covering outputs each consisting of discrete output and output port pairs as introduced in DEVS with Ports.
- $ta: S \rightarrow \Re_0^+ \cup \infty$
  The time advance function *ta* schedules the time of the next internal event after each state transition. The definition is identical to Classic DEVS.

Figure 5 shows the dynamic behavior of an atomic EDSDEVS model am_{EDS}. At time t_u the confluent transition function δ_{con} handles two concurrent events. The first event contains a bag covering external inputs received by the atomic model am_{EDS}. The figure depicts an example bag covering three external inputs received at two different input ports. A concurrent internal event at t_u was scheduled by the previous execution of the time advance function *ta*. Depending on the specific implementation of function δ_{con} sequence a) or b) is executed. The execution of λ creates a bag covering outputs. The shown bag Y_u^b covers two outputs.



$X_{inport_0} = \{x_0, \dots x_m\}$
inport$_0$
$X_{inport_i} = \{x_0, \dots x_n\}$
inport$_i$
am$_{EDS}$
$s_u, s_{u+1}, s_{u+2} \in S = \{s_0, \dots s_r\}$
outport$_0$
outport$_j$
$Y_{outport_0} = \{y_0, \dots y_p\}$
$Y_{outport_j} = \{y_0, \dots y_q\}$
$X^b = \{x_k \mid x_k = (v,p), v \in X_p, p \in InputPorts\}$
$Y^b = \{y_k \mid y_k = (v,p), v \in Y_p, p \in OutputPorts\}$
example input bag:
$X_u^b = \{(x_a, inport_0), (x_b, inport_0), (x_c, inport_1)\}$
example output bag:
$Y_u^b = \{(y_d, outport_0), (y_e, outport_1)\}$
external event X^b at t_u
concurrent events induce
$s_{u+2} = \delta_{con}(s_u, X^b, e)$
t_{last}
t_u
t
internal event at t_u
calling a) or b) depends on specific implementation of δ_{con}
a)
b)
$s_{u+1} = \delta_{ext}(X^b, s_u, e)$ with $e = (t_u - t_{last})$
$s_{u+2} = \delta_{int}(s_{u+1}, t_u)$
$s_{u+1} = \delta_{int}(s_u, t_u)$
$s_{u+2} = \delta_{ext}(X^b, s_{u+1}, e)$
X^b bag of input events
Y^b bag of output events
i,j number of input and output ports
m,n,p,q number of different X and Y events per port
r number of internal states
X^b_u bag of input messages at t_u
t_u time of concurrent external and internal message
s_u state at time t_u

Fig. 5 Dynamic Behavior of an Atomic EDSDEVS Model

An Extended Dynamic Structure DEVS coupled model is defined by the following 7-tuple:

$CM_{EDS} = (d_n, S_{EDS}, \delta_{x\&s}, \delta_{int}, \delta_{con}, \lambda, ta)$

- d_n specifies the name of the coupled model.
- In the EDSDEVS formalism the coupled model structure consists not only of sets of sub components and coupling relations as in DSDEVS but also of additional interface definitions i.e. input and output port definitions. The set of sequential structure states $\{s_0, s_1, \dots, s_n\} = S_{EDS}$ has to define all structure variants of the coupled model CM_{EDS}. Two model structure variants can vary in different interface definitions, in contrast to DSDEVS where each model has a non-variable interface with a single input and a single output port. Hence, a structure state has to incorporate interface definitions with sets of input and output ports additionally to the structure state definition as introduced in section 3.3. An EDSDEVS structure state is defined by a 10-tuple:
  $s_i = (X, Y, H_{EDS}, D, \{ M_d \}, InputPorts, OutputPorts, EIC, EOC, IC)$
  - *X* and *Y* specify the sets of discrete input and outputs. The sets exactly match the extended definitions of *X* and *Y* as introduced in DEVS with Ports.
  - The sets H_{EDS}, D and M_d exactly match the sets H_{VS}, D and M_d of the DSDEVS formalism introduced in section 3.3.
  - *InputPorts* and *OutputPorts* specify the sets of input and output port names of the coupled model CM_{EDS}. These two elements of the structure state s_i are introduced by the EDSDEVS formalism.

- *EIC*, *EOC* and *IC* are the external input, external output and internal couplings of CM_{EDS}. The definition of the coupling relations exactly match the definition as introduced with the DEVS with Ports extension.

- $\delta_{x\&s}$: $Q \times X^b \rightarrow H_{EDS}$ where X^b is a bag covering input, input port pairs and $Q = \{(h,e) \mid h \in H_{EDS}, 0<e<t_{next}\}$
  The external and state transition function δ_{ext} handles a bag covering inputs, each consisting of a pair of a), b) or c):
  a) A discrete input $v \in X_p$ and an input port $p \in$ *InputPorts*. The set X_P is the set of discrete inputs at port p and *InputPorts* is the set of input ports of model CM_{EDS}.
  b) A discrete output $v \in M_d.Y_p$ and an output port $p \in M_d.OutputPorts$ where M_d is the model of the sub component d of the coupled model CM_{EDS}. The set $M_d.Y_P$ is the set of discrete outputs at port p and $M_d.OutputPorts$ is the set of output ports of model M_d.
  c) A discrete input $v \in M_d.X_p$ and an input port $p \in M_d.InputPorts$ where M_d is the model of the sub component d of the coupled model CM_{EDS}. The set $M_d.X_P$ is the set of discrete inputs at port p and $M_d.InputPorts$ is the set of input ports of model M_d.

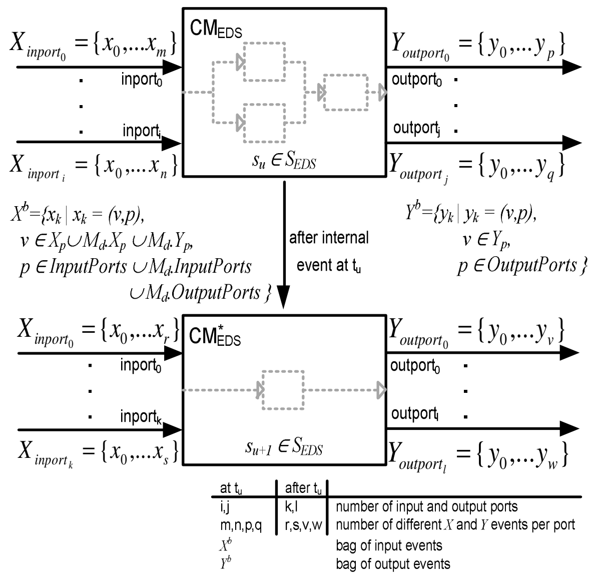
This extended definition of δ_{ext} is a fusion and extension of the δ_{ext} definitions of DSDEVS, PDEVS and DEVS with Ports. In DSDEVS only state events induced by output events of sub components are handled. However, an output port can have coupling relations to multiple input ports. In this case there is a difference in the handling of a single output event of a single source sub model or multiple input events of different destination sub models. Hence, the external and state transition function of EDSDEVS can handle both output and input events. However, the functionality is in accordance with the description of the DSDEVS external and state transition function $\delta_{x\&s}$.

- δ_{int}: $S_{EDS} \rightarrow S_{EDS}$
  ta: $S_N \rightarrow \Re_0^+ \cup \infty$
  The internal state transition function δ_{int}, and the time advance function *ta* exactly match the functions of the DSDEVS formalism.

- δ_{con}: $S_{EDS} \times X^b \rightarrow S_{EDS}$
  The confluent transition function δ_{con} handles the execution sequence of δ_{int} and δ_{ext} functions during concurrent external and internal events. The EDSDEVS formalism introduces the confluent transition function also at coupled model level due to the fusion of PDEVS and DSDEVS. An EDSDEVS coupled model handles external, state and internal events. Hence and in contrast to PDEVS, in EDSDEVS concurrent external and internal events can occur also at coupled model level. Consequently, a confluent transition function to handle concurrent events is necessary at this level. The functionality is in accordance with the description of the confluent transition function δ_{con} at atomic model level in this section.

- λ: $S_{EDS} \rightarrow Y^b$
  The output function λ generates a bag covering outputs $Y^b = \{y_i\}$ depending on state S_{EDS}. An output y_i consists of a pair of discrete output $v \in Y_p$ and output port $p \in$ *OutputPorts*. The set Y_P is the set of discrete outputs at port p and *OutputPorts* is the set of output ports of model CM_{EDS}. The output function λ in the EDSDEVS formalism merges three sources:
  - The output function λ at coupled model level is introduced by DSDEVS.
  - The definition of the function creating a bag covering outputs is based on PDEVS.
  - The output event structure with pairs of output/output port is introduced by DEVS with Ports.

Figure 6 shows the dynamic behavior of a coupled EDSDEVS model CM_{EDS}. At time t_u the confluent transition function δ_{con} handles concurrent external and internal events. The first event is a bag covering inputs received at input ports by the coupled model CM_{EDS}. A concurrent internal event at t_u was scheduled by the last execution of the time advance function. Depending on the specific implementation of function δ_{con} sequence a) or sequence b) is executed. The execution of the internal state transition function δ_{int} can change the structure state s_u to s_{u+1} or s_{u+1} to s_{u+2} and therefore the model structure of CM_{EDS} to CM_{EDS}^*. The execution of the output function λ creates a bag covering outputs Y_u^b. The depicted example bag Y_u^b covers two outputs.


CM_EDS
$X_{inport_0} = \{x_0,...x_m\}$
inport_0
inport_i
$X_{inport_i} = \{x_0,...x_n\}$
$s_u \in S_{EDS}$
$Y_{outport_0} = \{y_0,...y_p\}$
outport_0
outport_j
$Y_{outport_j} = \{y_0,...y_q\}$
$X^b=\{x_k \mid x_k = (v,p), v \in X_p \cup M_d.X_p \cup M_d.Y_p, p \in InputPorts \cup M_d.InputPorts \cup M_d.OutputPorts\}$
after internal event at t_u
$Y^b=\{y_k \mid y_k = (v,p), v \in Y_p, p \in OutputPorts\}$
CM*_EDS
$X_{inport_0} = \{x_0,...x_r\}$
inport_0
inport_k
$X_{inport_k} = \{x_0,...x_s\}$
$s_{u+1} \in S_{EDS}$
$Y_{outport_0} = \{y_0,...y_v\}$
outport_0
outport_l
$Y_{outport_l} = \{y_0,...y_w\}$
at t_u | after t_u
i,j | k,l | number of input and output ports
m,n,p,q | r,s,v,w | number of different X and Y events per port
X^b bag of input events
Y^b bag of output events

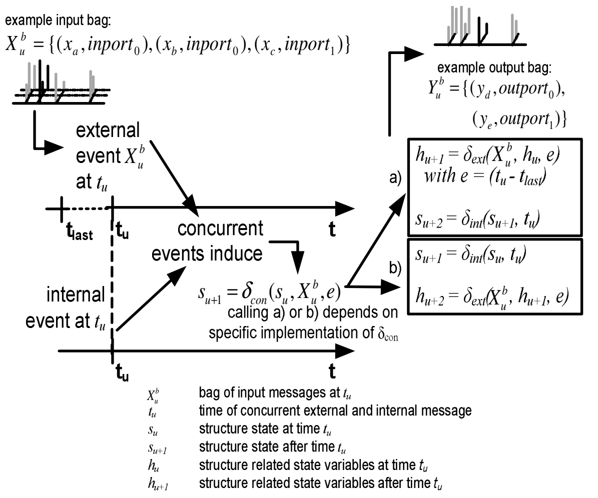

example input bag:
$X_u^b = \{(x_a, inport_0), (x_b, inport_0), (x_c, inport_1)\}$
example output bag:
$Y_u^b = \{(y_d, outport_0), (y_e, outport_1)\}$
external event X_u^b at t_u
concurrent events induce
internal event at t_u
t_{last}
t_u
t
$s_{u+1} = \delta_{con}(s_u, X_u^b, e)$
calling a) or b) depends on specific implementation of δ_{con}
a)
$h_{u+1} = \delta_{ext}(X_u^b, h_u, e)$ with $e = (t_u - t_{last})$
$s_{u+2} = \delta_{int}(s_{u+1}, t_u)$
b)
$s_{u+1} = \delta_{int}(s_u, t_u)$
$h_{u+2} = \delta_{ext}(X_u^b, h_{u+1}, e)$
X_u^b bag of input messages at t_u
t_u time of concurrent external and internal message
s_u structure state at time t_u
s_{u+1} structure state after time t_u
h_u structure related state variables at time t_u
h_{u+1} structure related state variables after time t_u

Fig. 6 Dynamic Behavior of a Coupled EDSDEVS Model

4.2. EDSDEVS Simulation

The simulation engine for EDSDEVS models is a combination and extension of the simulation algorithms of Classic DEVS, PDEVS and DSDEVS. The message handling of coordinators are largely similar to simulators. Each coordinator holds its own time of next internal event in t_{next_c} and searches the minimum time of next internal event in t_{next} of sub components and in its own t_{next_c}.

Figure 7 depicts an EDSDEVS model example with associated simulation model elements i.e. root coordinator, coordinator and simulator instances, message handling and model function calls. The overall structure is very similar to the Classic DEVS simulation model execution except for additions at the coordinators and associated coupled models. Because of complexity and clarity selected situations are shown in sections:

i. (Figure 7a) initialisation phase with i-message handling:
During the initialisation phase model component's init functions are called because of an i-message handling similar to Classic DEVS. Additionally, after structure changes during the simulation phase the init function is called too.

ii. (Figure 7b) *-message handling created due to an internal event of model *am2*:
The root coordinator advances the simulation clock and a *-message is firstly created. The message is sent to the successor coordinator instance of coupled model *CM1* (not depicted). This coordinator instance compares the actual simulation time *t* with its own next internal event time stored in t_{next_c} and determines that it is not responsible for handling this event. Hence, the event is forwarded to the successor coordinator instance of *CM2*. The coordinator instance is again not responsible for handling the message itself but knows that a sub component scheduled the event. The coordinator instance will then forward the message to the appropriate simulator instance associated with *am2*. The simulator instance of *am2* calls the model functions λ and δ_{int}. A result of calling λ could be a y-message sent back to the subordinate coodinator instance of *CM2*. This coordinator instance reacts with the call of the model function $\delta_{x\&s}$ of *CM2* and a message forward to the simulator instance of *am3* due to an appropriate IC coupling.

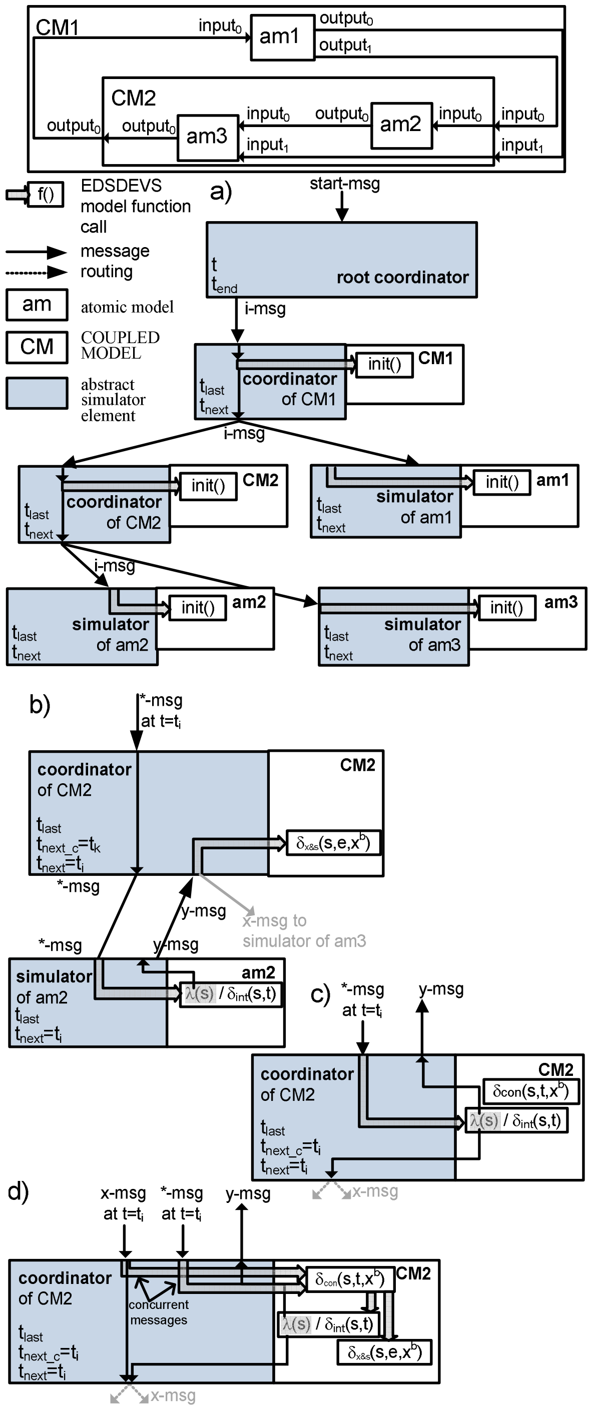


Fig. 7 EDSDEVS Model Example with Simulation Model Elements and Message Flow during Initialization and Simulation Phases

iii. (Figure 7c) *-message handling created due to an internal event of model *CM2*:

The depicted situation is similar to 7b except that the coordinator instance of *CM2* determines that simulation time *t* and its t_{next_c} are equal. Hence, it has to handle the *-message itself with calling λ and δ_{int} model functions of *CM2* with the possibility of generating a y-message sent to a sub component and/or superordinated coordinator instance and of changing its sequential structure state S_{EDS}.

iv. (Figure 7d) concurrent event handling with the confluent transition function δ_{con}:
The figure depicts the handling of concurrent external and internal messages by the coordinator instance of *CM2*. The confluent function of *CM2* is called to handle the concurrent messages. Depending on the specific implementation of δ_{con} the external transition function $\delta_{x\&s}$ and internal transition/output functions δ_{int}, respectively, are firstly called. The external message is concurrently handled by the function δ_{con} and forwarded to the simulator instance of sub component *am2* as an x-message due to an appropriate EIC. Calling the output function λ could cause an y-message sent to a sub component and/or superordinated coordinator instance.

Listings 1 and 2 show the pseudo codes of the EDSDEVS simulator components.

```
variables:
    $t_{last}$   // time of last event
    $t_{next}$   // time of next int state event
    am     // associated atomic model

// t=0 init at simulation start
// t>0 init after structure change
when receive i-msg(t)at time t
    am.init(t)
    $t_{last}$ := t
    $t_{next}$ := am.ta()

when receive *-msg(t) at time t
    if t <> $t_{next}$
        error: bad synchronisation
    y_bag := am.$\lambda$()
    send y_bag in a y-msg to parent coord.

when receive x-msg(t, x_bag) at time t
  with value x_bag containing x.value und
  x.port pairs
    if not ($t_{last} \le t \le t_{next}$)
        error: bad synchronisation
    if t=$t_{next}$ and x_bag is not empty
        //concurrent ext. & int. event
        am.$\delta_{con}$( t, x_bag)
    else if t=$t_{next}$ and x_bag is empty
        // internal event
        am.$\delta_{int}$(t)
    else
        // external event
        am.$\delta_{ext}$( t-$t_{last}$, x_bag)
    end if
    $t_{last}$ := t
    $t_{next}$ := $t_{last}$ + am.ta()
```

Listing 1 Pseudo Code of an EDSDEVS Simulator

```
variables:
    $t_{last}$   // time of last event
    $t_{next}$   // minimal time of next int.
           // state event of coupled model
           // or sub component
    $t_{next\_c}$ // time of next int state event
           //of the coupled model itself
    CM     // associated coupled model with
           // CM.$s_t$ current structure state
    IMM    // imminent children
    mail   // output mail bag

// t=0 init at simulation start
// t>0 init after structure change
when receive i-msg(t)at time t
    CM.init(t)
    foreach sub component $M_d \in$ CM.$s_t$.M
        send i-msg(t) to $M_d$
    $t_{last}$ := t
    // determine time of next scheduled
    // internal state event of coupled
    // model itself
    $t_{next\_c}$ := CM.ta()
    // determine minimum time of next
    // scheduled internal state events of
    // coupled model and all subcomponents
    $t_{next}$ := min( $t_{next\_c}$, { $M_d$.$t_{next}$ | $M_d$
     $\in$ CM.$s_t$.M } )

when receive *-msg(t) at time t
    if t <> $t_{next}$ & t<>$t_{next\_c}$
        error: bad synchronisation
    // internal state event of CM
    if t=$t_{next\_c}$
        y_bag := CM.$\lambda$()
        send bag of value/output port pairs
         in a y-msg to parent coordinator
    // internal state event of a subcomp.
    else if t=$t_{next}$
        // find all subcomps with $t_{next}$==t
        IMM:={$M_d$ |$M_d \in$ CM.$s_t$.M $\wedge$ $M_d$.$t_{next}$= t}
        foreach $M_d$ in IMM
            send *-msg(t) to $M_d$

when receive x-msg(t, x_bag) at time t
  with x_bag containing x.value/x.port
  pairs
    if not ($t_{last} \le t \le t_{next\_c}$)
        error: bad synchronisation
    if t=$t_{next\_c}$ and x_bag is not empty
        // concurrent ext. and int. event
        CM.$\delta_{con}$( t, x_bag)
    else if t=$t_{next\_c}$ and x_bag is empty
        CM.$\delta_{int}$( t )  // int. event
    else
        CM.$\delta_{x\&s}$( t-$t_{last}$, x_bag) //ext. event
    end if
    // get all subcomponents $M_{d*}$ with an
    // appropriate EIC
    receivers:=subcomponents{$M_d$|$M_d \in$CM.$s_t$.M}
     with {coupling|coupling$\in$CM.$s_t$.EIC}
    // forwards x-msg to all appropriate
    // subcomponents
    foreach subcomponent $M_{d*}$ in receivers
        // ext. event of subcomponent
        CM.$\delta_{x\&s}$( t-$t_{last}$, x_bag)
        send x-msg(t, x_bag, $M_{d*}$.p) to $M_{d*}$
         at port p
    foreach subcomponent $M_{d*}$ in IMM and not
     in receivers
        // send empty bag without inputport
        send x-msg(t, NULL, NULL) to $M_{d*}$
    $t_{last}$ := t
    $t_{next\_c}$ := $t_{last}$ + CM.ta()
    $t_{next}$ := min($t_{next\_c}$,{$M_d.t_{next}$|$M_d \in$CM.$s_t$.M})

when receive y-msg(t, y_bag, d) at time t
with y_bag with value/port pairs from d
    // collect all y-msgs from all subcomp
    if d is not the last not reporting d
     in IMM
        add (y_bag, d) to mail
        mark d in IMM as reporting
    // all subcomps now handled their *msg
    else if d is the last not reporting d
     in IMM
        CM.$\delta_{x\&s}$(t-$t_{last}$, mail)
        // check ext. coupling to form sub-
        // bag of parent output
        y_bag$_{parent}$ = NULL
        foreach d in mail where (y_bag and
         d) has an appropriate EIC
            add y_bag to y_bag$_{parent}$
        send y-msg(t, y_bag$_{parent}$,, CM) to
         parent model
        // check IC to get children $M_{d*}$
        // with an appropriate IC who
        // receives a sub bag
        receivers := subcomponents{$M_d$|d in
            mail, $M_d \in$CM.$s_t$.M} with
            {coupling|coupling$\in$CM.$s_t$.IC}
        foreach subcomp $M_{d*}$ in receivers
            creates sub bag x_bag from mail
             with elements where $M_{d*}$ is
             receiver
            send x-msg(t, x_bag) to $M_{d*}$
            mark d in IMM as sending
        foreach sub component $M_{d*}$ in IMM
         where $M_{d*}$ is not sending
            send x-msg(t, NULL) to $M_{d*}$
    $t_{last}$ := t
    $t_{next\_c}$ := $t_{last}$ + CM.ta()
    $t_{next}$ := min( $t_{next\_c}$, { $M_d.t_{next}$ | $M_d$
     $\in$ CM.$s_t$.M } )
```

Listing 2 Pseudo Code of an EDSDEVS Coordinator

5. CONCLUSIONS

The EDSDEVS formalism introduced in this contribution is a fusion of Classic DEVS with several extensions. This approach is an as generic as possible modeling and simulation formalism based on DEVS. It widens significantly the application area of DEVS modeling and simulation. Further extensions are desirable and essential. To establish a widely accepted modeling and simulation approach extensions for parallel computing and graphical modeling are necessary. There are also approaches for hybrid DEVS extensions i.e. the support of continuous state changes. These proposals are recommended as further research.

REFERENCES

Barros, F.J., 1996. *Modeling and Simulation of Dynamic Structure Discrete Event Systems: A General Systems Theory Approach.* Thesis (PhD), University of Coimbra

Chi, S.D., 1997. Model-based Reasoning Methodology Using the Symbolic DEVS Simulation *Trans. of SCS*, 14(3): p.141-152

Chow, A.C., Zeigler, B.P., 1994. Parallel Devs: A Parallel, Hierarchical, Modular Modeling Formalism. *Proceedings of the 1994 Winter Simulation Conference*, LakeBuenaVista/FL, USA

Deatcu, C., Pawletta, T., Hagendorf, O., Lampe, B., 2009. Considering Workpieces as Integral Parts of a DEVS Model. *Proceeding of 2009 EMSS - part of I3M Multiconference 2009* Teneriffa, Spain

Hagendorf, O., Pawletta, T., Pawletta, S., Colquhoun, G., 2006. An approach for modelling and simulation of variable structure manufacturing systems. *Proceeding of the 2006 ICMR* Liverpool, UK

Pawletta, T., Lampe, B.P., Pawletta, S., Drewelow, W., 1996. Dynamic structure simulation based on discrete events. *ASIM-Mitteilungen Nr.53, 9. Workshop Simulation and AI,* p.7-11, Ulm, Germany, 02.1996.

Pawletta, T., Deatcu, C., Pawletta, S., Hagendorf, O., Colquhoun, G., 2006. DEVS-Based Modeling and Simulation in Scientific and Technical Computing Environments *Proceedings of the 2006 Spring Simulation Conference*, Huntsville/AL, USA

Praehofer, H., 1992. *CAST Methods in Modelling*. Pichler, F., Schwärtzel, H. Springer Pub.

Uhrmacher, A.M., Arnold, R., 1994. Distributing and maintaining knowledge: Agents in variable structure environment. *5th Annual Conference on AI, Simulation and Planning of High Autonomy Systems*, p. 178-194

Wainer, G., Giambiasi, N., 2001. Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *SIMULATION Transactions of The Society for Modeling and Simulation International*, vol. 76, 01.2001

Wainer, G. A., 2009. *DEVS Tools.* Available from: www.sce.carleton.ca/faculty/wainer/standard/tools.htm [Accessed 06.2009]

Zeigler, B.P., 1976 *Theory of Modeling and Simulation.* 1st edition, John Wiley

Zeigler, B.P., Praehofer, H., Kim, T.G., 2000 *Theory of Modelling and Simulation.* 3rd edition, Academic Press