Preprints of the 9th Vienna International Conference on Mathematical Modelling
Vienna, Austria, February 21-23, 2018

RPDEVS: Revising the Parallel Discrete Event System Specification

Franz Preyser * **Bernhard Heinzl** * **Wolfgang Kastner** *

* *Automation Systems Group, Vienna University of Technology, A-1040 Vienna, Austria (e-mail: {Franz.Preyser, Bernhard.Heinzl, Wolfgang.Kastner}@tuwien.ac.at).*

Abstract: In this work, we present a Revised Parallel DEVS (RPDEVS) formalism. The Classic Discrete Event System Specification (DEVS) and Parallel DEVS (PDEVS) formalisms do not support modelling of 'true' mealy behaviour, i.e. reacting to an input message immediately with an output message. Instead, such behaviour has to be modelled via transitory states and multiple state updates. This not only increases model complexity, it also impedes reusability of model components in different contexts.
RPDEVS enhances PDEVS with the capability to model mealy behaviour directly. Hence, the *output function* λ can access the input bag. This introduces some challenges regarding the simulation algorithm which we will take a look at.
Further, the terms *algebraic loop* and *illegitimate model* will be discussed in the context of RPDEVS. It is shown that RPDEVS models which are free of algebraic loops are also legitimate. Finally, it will be demonstrated that like Classic DEVS and PDEVS, also RPDEVS provides closure under coupling.

Keywords: Discrete-event systems, Mathematical models, Modelling, Simulation, DEVS, PDEVS, RPDEVS, Algebraic loop, Transitory state, Mealy behaviour

1. INTRODUCTION

Discrete Event System Specification (DEVS) (Zeigler et al. (2000)) denotes a very generic mathematical formalism to describe the discrete behaviour of dynamical systems. DEVS allows a hierarchical and modular system description, which is of high value when modelling large and complex systems. There are various specialised and extended versions of DEVS, with Parallel DEVS (PDEVS) (Chow and Zeigler (1994); Zeigler et al. (2000)) being the most popular one.

Over the last decades, Classic DEVS and its variant PDEVS have established in the scientific community, which can be verified by the numerous publications and simulation engines existing around DEVS (Franceschini et al. (2014)). We applied both Classic DEVS (Preyser (2015)) and PDEVS (Raich et al. (2016)) in projects and experienced modelling difficulties with both formalisms (Preyser et al. (2016)). But also others describe similar problems (Traoré (2007), Cicirelli et al. (2007)).

DEVS follows a component-based approach where models can be designed in a modular-hierarchical manner. Each component has an inner state which is updated either due to the arrival of an input messages or due to internal dynamics. Difficulties arise when multiple state updates occur at one instant of simulation time. This is caused by multiple input messages which arrive concurrently (at the same instant of simulation time) in principal, however are processed sequentially, each causing an individual state update. The order in which concurrent input messages are processed depends on the structure of the model surrounding the component. Regarding re-usability of model components, it is crucial that their behaviour is independent of any surrounding topology. Consequently, when designing a reusable component, all possible sequential processing orders of concurrent input messages have to be considered. This can be very cumbersome. In Preyser et al. (2016), specific examples are given for DEVS-based models. However, concurrent events and multiple state updates seem to be a problem in Discrete Event System (DES) models in general (Junglas (2016)).

As the reuse of model components is essential for an efficient design of large-scale models, a modelling formalism should support re-usability and facilitate transparent behaviour of components. One way to achieve this is by allowing state updates to be executed only once per time instant and component, requiring all input messages to be available before that single state update. We altered the PDEVS formalism to fulfil these requirements, resulting in what we call Revised Parallel DEVS (RPDEVS).

In the following section PDEVS will be recapped shortly. Then, the core RPDEVS will be described, followed by a discussion of the terms *illegitimate* and *algebraic loop* in the context of RPDEVS. It is shown that models which are free of algebraic loops are also legitimate, before finally *closure under coupling* of RPDEVS is described.

2. PARALLEL DEVS

In Classic DEVS (Zeigler et al. (2000)), concurrent input messages at a model component are processed strictly sequentially. The processing order is not defined within

the component, but results from the definition of the coupling model, the component is used in. Consequently, when designing a re-usable model component, the modeller has to consider every possible processing order. PDEVS improves this situation significantly. However, components with *mealy behaviour* (that is reacting to an input message immediately with an output message), still have to be modelled using transitory states. Transitory states though, again cause sequential processing of concurrent events.

2.1 Atomic PDEVS

The basic component is the *atomic PDEVS*. An atomic PDEVS describes a system with the possibility to receive input messages, produce output messages and update the state variables describing the current internal state. Internal states have a maximum time to live. State updates are triggered either through the arrival of input messages (*external event*), due to the expiration of the life time of the current state (*internal event*), or due to an concurrent occurrence of both. An atomic PDEVS is described by

$$< X^b, S, Y, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta > \quad (1)$$

with the following meanings:

$X^b \ldots$ set of possible input bags
$Y \ldots$ set of possible outputs
$S \ldots$ set of possible states (=state space)
$ta : S \to [0, \infty] \ldots$ *time advance function*
$Q = \{(s, e) | s \in S, e \in [0, ta(s)]\}$
$\delta_{int} : \ S \to S \ldots$ *internal state transition function*
$\delta_{ext} : \ Q \times X^b \to S \ldots$ *external state transition function*
$\delta_{conf} : S \times X^b \to S \ldots$ *confluent state transition function*
$\lambda : S \to Y \ldots$ *output function*

According to the PDEVS abstract simulator (Chow et al. (1994)), whenever the life time $ta(s)$ of the current state $s \in S$ expires ($e = ta(s)$), the output function $\lambda(s)$ is called, potentially generating output messages. After that, a new state is computed, by either executing $\delta_{int}(s)$ or $\delta_{conf}(s, x^b)$. The latter if input messages arrived as well at current simulation time. If input messages arrive before the elapsed time e reaches the maximum time to live $ta(s)$, $\delta_{ext}(s, e, x)$ is executed, also resulting in a state update. Concurrently arriving input messages are gathered in a bag (multiset) $x^b \in X^b$, indicated with the superscript b, and processed in parallel in principle. Exceptions occur in models containing mealy components (see section 2.3).

It is interesting to note that there is a certain similarity to state automata. However, when taking a closer look at the output function λ in (1), it can be seen that PDEVS only allows *moore-like* behaviour. The output message $y = \lambda(s)$ does not depend on the current input, but only on the inner state of the system. Furthermore, λ is only called at internal and confluent events, but not at external events. Therefore, mealy behaviour, where $y = \lambda(s, x)$ depends not only on the state but also on the inputs, can not be modelled directly.

2.2 Coupled PDEVS

Atomic components may be coupled with each other in a way that output messages of one component become input messages of another component (see Figure 1).

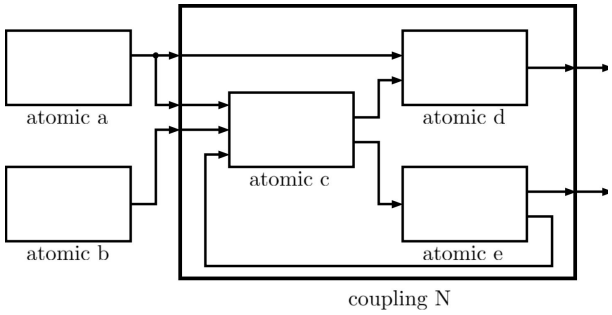

Fig. 1. PDEVS coupling.

Coupled PDEVS models can be used like atomic PDEVS in larger coupled models. This property, called *closure under coupling*, enables hierarchical model structuring. A coupled PDEVS N is described by the following elements:

$$< X^b, Y, D, \{M_d\}_{d \in D}, \{I_d\}_{d \in D_N}, \{Z_{i,d}\}_{i,d \in D_N} >$$

where $D_N = D \cup \{N\}$ and

$X^b \ldots$ set of possible input bags
$Y \ldots$ set of possible outputs
$D \ldots$ index set
$M_d \ldots$ child components of N for each $d \in D$
$I_d \subset D \cup \{N\} \ldots$ influencer set of d, $d \notin I_d$
$Z_{i,d} \ldots$ output translation function

$Z_{i,d} : Y_i \to X_d$ translates an output message y_i of component i into an input message $x_{i,d} \in x_d^b$ for component d, if i and d are child components of the coupled model N. If i is the coupled model itself ($i = N$), then $Z_{N,d} : X_N \to X_d$ transforms input messages x_N of the coupled model to input messages $x_{N,d} \in x_d^b$ for the child component d. On the other hand, if d is the coupled model ($d = N$), then $Z_{i,N} : Y_i \to Y_N$ transforms the output message y_i of i to an output message y_N of the coupled model N. If there is no coupling connection from components i to d ($i \notin I_d$), the result of $Z_{i,d}$ is always $\emptyset$.

In a coupled PDEVS, the internal events of all *imminent* components (components which experience an internal event) can be processed in parallel. Thereby, first λ is executed at each imminent component (λ-*phase*), producing a set of output messages. These output messages are routed to the receiving blocks, where they fill their input bags. The input bag x_d^b of component d is calculated as

$$x_d^b = Z_{N,d}(x_N) \ \uplus \biguplus_{i \in I_d \setminus \{N\}} Z_{i,d}(y_i),$$

where $y_i = \lambda_i(s_i)$ denotes the output of component i.

After the λ-phase follows the δ-*phase*, in which the state update function is executed at each block that has experienced an internal, external or a confluent event:
$s_{d,new} = \delta_{int}(s_{d,old}) \ / \ \delta_{ext}(s_{d,old}, e_d, x_d^b) \ / \ \delta_{conf}(s_{d,old}, x_d^b)$

2.3 Sequential Processing in PDEVS

At first glance, it seems as if all input messages arriving at the same simulation time at a component are processed in one single state update. However, as already elaborated in Preyser et al. (2016), if a model contains components with mealy behaviour (*mealy components*), sequential processing of concurrent input messages occurs.

To obtain mealy behaviour, a component has to change into a transitory state $\tilde{s}$ with $ta(\tilde{s}) = 0$ during δ_{ext} or δ_{conf} after an input message arrived (e.g. $\tilde{s} = \delta_{conf}(s_0, x^b)$). This leads to a repetition of the λ-phase ($y = \lambda(\tilde{s}) = \lambda(\delta_{conf}(s_0, x^b))$). Thereby, λ is executed at every component that is in a transitory state. Then also the δ-phase is repeated, in which state transitions are calculated at every influenced component. This cycle of repeating λ- and δ-phases continues until every component resides in a non-transitory state, before simulation time can advance.

Examples for mealy components are blocks that apply simple arithmetic functions to input messages, like multiplying them with a constant factor, or simple switch blocks routing messages to a selected output. Thus, to abstain from mealy components when creating a library of reusable model components is no serious option.

Example 1. Consider the coupled model in Figure 1, with atomic a experiencing an internal event at time t_1 while being in state $s_{a,0}$. Assuming atomic c to be of type mealy leads to the following sequence of events and function calls:

$$\begin{aligned}
1^{st}\lambda - phase:\ & x_{a,c} = Z_{a,c}(\lambda_a(s_{a,0})) \\
& x_{a,d} = Z_{a,d}(\lambda_a(s_{a,0})) \\
1^{st}\delta - phase:\ & s_{a,1} = \delta_{int,a}(s_{a,0}) \\
& \tilde{s}_{c,1} = \delta_{ext,c}(s_{c,0}, e_{c,1}, \{x_{a,c}\}),\ ta(s'_{c,1}) = 0 \\
& \tilde{s}_{d,1} = \delta_{ext,d}(s_{d,0}, e_{d,1}, \{x_{a,d}\}) \\
2^{nd}\lambda - phase:\ & x_{c,d} = Z_{c,d}(\lambda_c(\tilde{s}_{c,1})) \\
2^{nd}\delta - phase:\ & s_{c,1} = \delta_{int,a}(\tilde{s}_{c,1}),\ ta(s_{c,1}) \neq 0 \\
& s_{d,1} = \delta_{ext,d}(\tilde{s}_{d,1}, 0, \{x_{c,d}\})
\end{aligned}$$

Consequently, the new state of atomic d, which starts in state $s_{d,0}$ and receives the inputs $x_{a,d}$ and $x_{c,d}$ is

$$s'_{d,1} = \delta_{ext,d}\big(\delta_{ext,d}(s_{d,0}, e_{d,1}, \{x_{a,d}\}), 0, \{x_{c,d}\}\big).$$

However, with a different structure of the coupled model surrounding component d, the new state of d could also be

$$\begin{aligned}
s''_{d,1} &= \delta_{ext,d}\big(\delta_{ext,d}(s_{d,0}, e_{d,1}, \{x_{c,d}\}), 0, \{x_{a,d}\}\big),\ \text{or} \\
s'''_{d,1} &= \delta_{ext,d}\big(s_{d,0}, e_{d,1}, \{x_{c,d}, x_{a,d}\}\big).
\end{aligned}$$

For a reusable model component with transparent behaviour, it would be desirable to have one unique successor state if the component is in a given state and receives a certain set of input messages at one instant of simulation time, regardless of the components surroundings. To achieve this in the given example, the modeller would have to define the state transition functions in a way that $s'_{d,1} = s''_{d,1} = s'''_{d,1}$. Since the number of possible cases increases rapidly with the number of concurrent input messages, this complicates the design of re-usable components.

3. REVISED PARALLEL DEVS

The idea of RPDEVS is to call λ also at external events, and thereby allow λ to access the input bag x^b, and the *elapsed time* e. This permits direct modelling of mealy behaviour. During an iterative λ-phase, *all* concurrent input messages for a component are collected, before calculating the state transition only once for a particular point in simulation time.

3.1 Atomic RPDEVS

An atomic RPDEVS is defined by the 6-tuple

$$< X, S, Y, \delta, \lambda, ta > .$$

Thereby, δ and λ have the following form:

$$\begin{aligned}
\delta &: Q \times X \to S \\
\lambda &: Q \times X \to Y
\end{aligned} \tag{2}$$

In RPDEVS, λ is called at any kind of event, internal, external, and confluent. The explicit distinction between the different event types is dropped. The type of event can be determined by the content of the input bag (empty or not) and the value of the *elapsed time* e (equal $ta(s)$ or not) anyway, as described in Chow and Zeigler (1994), section 4, page 5: *"From the definition of the δ_{int}, δ_{conf}, and δ_{ext}, we see that they are special cases of a more generic transition function $\delta(s, e, x^b)$"*

As stated in Goldstein et al. (2013), it appears quite frequently in atomic DEVS models that the calculation of the output message in λ already includes (parts of) the calculation of the new state. However, since λ is not allowed to change the inner state of the system, these calculations have to be repeated in δ. This restriction on λ is even more crucial in RPDEVS, because λ may be called multiple times at an instant of time with only the last call being the valid one.

Nevertheless, there is a way to reuse state calculations in δ which have already been executed in λ. For this purpose, alternatively to the definition in (2), δ and λ may be specified as follows:

$$\begin{aligned}
\delta &: S_\delta \times S_\lambda \times [0, \infty) \times X \to S_\delta,\ (s_\delta, s_\lambda, e, x^b) \mapsto s_\delta \\
\lambda &: S_\delta \times [0, \infty) \times X \to Y \times S_\lambda,\ (s_\delta, e, x^b) \mapsto (y, s_\lambda)
\end{aligned}$$

Here, the state s of a system is separated into two parts $s = (s_\delta, s_\lambda)$, where λ is only allowed to read s_δ and to write s_λ. Thus, if λ already performs calculations also necessary in δ, the (intermediate) results can be stored in s_λ to be later used in δ. Regarding practical implementation, it is crucial that s_λ is re-assigned in every call of λ to prevent using an old non-valid value in δ.

As explained in section 3.2, in RPDEVS determining the component's output messages is an iterative process, in which λ may be called multiple times at the same point in simulation time. For this process to terminate, it is necessary to assure that every call of λ always uses the same information about the system state. This is why λ only depends on s_δ.

3.2 Coupled RPDEVS

The definition of a coupled RPDEVS N is equal to the definition of a coupled PDEVS:

$$< X_N^b, Y_N, D, \{M_d\}_{d \in D}, \{I_d\}_{d \in D_N}, \{Z_{i,d}\}_{i,d \in D_N} > \tag{3}$$

Whenever components in a coupled RPDEVS model are imminent, λ is calculated for each of them, initially assuming an empty input bag ($\lambda(s, ta(s), \emptyset)$). The produced output messages are routed to the receiving components, filling their input bags. This causes a (re-)calculation of λ at every receiving component, again potentially producing output messages.

The input bag x_d of a component d is divided into sub-input bags ${}_ix_d$, one for each influencer $i \in I_d$. If λ_i is recalculated at an influencer i producing the output y_i, the content of the corresponding sub-input bag ${}_ix_d$ is replaced

by $Z_{i,d}(y_i)$. Only if this replacement effectively changed ${}_ix_d$, a recalculation also of λ_d at the receiver d is triggered. In this way, it is assured that the iterative λ-*phase* will quit at some point, as long as the model is free of algebraic loops (see section 3.3). Only after the content of all input bags has stabilised, the state transitions are calculated.

For better readability, in the following the superscript b at input bags x^b will be omitted.

Let us take a closer look at the iterative λ-phase of a coupled model N. We assume n sub-components $D = \{1, 2, \ldots, n\}$. At an instant of time t_j, before any events are processed, the state of the system is defined to be the temporal left limit:

$$\boldsymbol{s}(t_j) = \boldsymbol{s_{j-1}} = \lim_{t \nearrow t_j} \boldsymbol{s}(t) = (s_{1,j-1}, s_{2,j-1}, \ldots, s_{n,j-1}).$$

An exception is the simulation start time t_0 where the system state is a given initial state $\boldsymbol{s_0}$

$$\boldsymbol{s}(t_0) = \boldsymbol{s_0} = (s_{1,0}, s_{2,0}, \ldots, s_{n,0}).$$

The initial input bags of the components

$$\boldsymbol{x_j^0} = (x_{1,j}^0, x_{2,j}^0, \ldots, x_{n,j}^0)$$

are either empty ($x_{d,j}^0 = \emptyset$) or, if external input messages from outside the coupling exist ($x_{N,j} \neq \emptyset$) they are $x_{d,j}^0 = Z_{N,d}(x_{N,j})$.

If components are imminent (i.e. $\exists d \in \{1, 2, \ldots, n\} : e_{d,j} = ta(s_{d,j-1})$), λ is executed at these components producing a set of output messages

$$\boldsymbol{y_j^0} = (y_{1,j}^0, y_{2,j}^0, \ldots, y_{n,j}^0),$$

with

$$y_{d,j}^0 = \begin{cases} \lambda(s_{d,j-1}, e_{d,j}, x_{d,j}^0) & \text{if } e_{d,j} = ta(s_{d,j-1}) \\ \emptyset & \text{else} \end{cases}$$

and corresponding new input messages

$$\boldsymbol{x_j^1} = (x_{1,j}^1, x_{2,j}^1, \ldots, x_{n,j}^1),$$

with $x_{d,j}^1 = Z_{N,d}(x_{N,j}) \uplus \biguplus_{i \in I_d \setminus \{N\}} Z_{i,d}(y_{i,j}^0),\ d \in D$. If any non-empty output messages are generated ($\exists d \in D : y_{d,j}^0 \neq \emptyset$), they trigger events at the receiving components and a second λ-iteration will be conducted, in which λ is (re-)calculated at every receiving component. If there are mealy components involved, this results in an updated vector of output messages

$$\boldsymbol{y_j^1} = (y_{1,j}^1, y_{2,j}^1, \ldots, y_{n,j}^1)$$

with

$$y_{d,j}^1 = \begin{cases} \lambda(s_{d,j-1}, e_{d,j}, x_{d,j}^1) & \text{if } x_{d,j}^1 \neq x_{d,j}^0 \\ y_{d,j}^0 & \text{else} \end{cases}$$

Mind that $\lambda(s_{d,j-1}, e_{d,j}, x_{d,j}^1) \neq \lambda(s_{d,j-1}, e_{d,j}, x_{d,j}^0)$ only for mealy components. The corresponding updated vector of input messages is

$$\boldsymbol{x_j^2} = (x_{1,j}^2, x_{2,j}^2, \ldots, x_{n,j}^2)$$

with $x_{d,j}^2 = Z_{N,d}(x_{N,j}) \uplus \biguplus_{i \in I_d \setminus \{N\}} Z_{i,d}(y_{i,j}^1),\ d \in D$. Now, for all components d with $x_{d,j}^2 \neq x_{d,j}^1$, λ is (re-)calculated again. This process repeats until

$$(y_{1,j}^{k-1}, y_{2,j}^{k-1}, \ldots, y_{n,j}^{k-1}) = (y_{1,j}^k, y_{2,j}^k, \ldots, y_{n,j}^k).$$

and consequently

$$(x_{1,j}^k, x_{2,j}^k, \ldots, x_{n,j}^k) = (x_{1,j}^{k+1}, x_{2,j}^{k+1}, \ldots, x_{n,j}^{k+1})$$

for a $k \in \mathbb{N}$. If such a value k does not exist, the model is called *illegitimate*. Otherwise, the model is *legitimate* and the content of all input bags 'stabilises', resulting in final input bags $\boldsymbol{x_j} = (x_{1,j}, x_{2,j}, \ldots, x_{n,j}) = (x_{1,j}^k, x_{2,j}^k, \ldots, x_{n,j}^k)$ for time instant t_j. Then, with these input bags, the state transition function $\delta_d(s_{d,j-1}, e_{d,j}, x_{d,j})$ can be applied for each component that has experienced an event at time t_j, resulting in the new state $(s_{1,j}, s_{2,j}, \ldots, s_{n,j})$ of the system.

Now, the question arises, what it needs for a RPDEVS model to be not *illegitimate*.

3.3 Algebraic Loops and Illegitimate Models

For every instant of time t_j and every pair of components $i, d \in D$, we can define an

Definition 2. ***input transport function***

$$\begin{aligned} \xi_{i,d,j} &: X_i \to X_d, \\ x_{i,j}^k &\mapsto \xi_{i,d,j}(x_{i,j}^{k-1}) = Z_{i,d}\left(\lambda_i(s_{i,j-1}, e_{i,j}, x_{i,j}^{k-1})\right) \end{aligned}$$

For a given input bag x_i^{k-1} at component i, $\xi_{i,d,j}$ calculates the part of the input bag of component d that is produced by component $i \in D$ at time t_j. For the case of i being the coupled model N itself, the input transport function is defined as

$$\begin{aligned} \xi_{N,d,j} &: X_N \to X_d, \\ x_{N,j} &\mapsto \xi_{N,d,j}(x_{N,j}) = Z_{i,d}(x_{N,j}). \end{aligned}$$

As explained in section 3.2, there may be multiple λ-iterations, especially if mealy components are involved. After the first λ-iteration, the input bag of a component d can be calculated as

$$x_{d,j}^1 = \biguplus_{i \in I_d} \xi_{i,d,j}(x_i^0) = \xi_{N,d,j}(x_{N,j}) \ \uplus \biguplus_{i \in I_d \setminus \{N\}} \xi_{i,d,j}(x_{i,j}^0).$$

After the $k-th$ λ-iteration, the input bag is

$$x_{d,j}^k = \biguplus_{i \in I_d} \xi_{i,d,j}(x_{i,j}^{k-1}) = \xi_{N,d,j}(x_{N,j}) \ \uplus \biguplus_{i \in I_d \setminus \{N\}} \xi_{i,d,j}(x_{i,j}^{k-1}).$$

Next, we define for an instance of time t_j, and every $d \in D$, and $k \in \mathbb{N}^+$, the

Definition 3. ***input bag function*** $\chi_{d,j}^k$ ***of order*** k

$$\chi_{d,j}^k : X_N \times X_1 \times \cdots \times X_n \to X_d, \text{with}$$

$$\chi_{d,j}^k(x_N, x_1, \ldots, x_n) = \begin{cases} \biguplus_{i \in I_d} \xi_{i,d,j}(x_i), \text{ for } k = 1, \text{ and else} \\ \biguplus_{i \in I_d} \xi_{i,d,j}\left(\chi_{i,j}^{k-1}(x_N, x_1, \ldots, x_n)\right) \end{cases}$$

Notice that

$$\begin{aligned} \chi_{d,j}^1(x_{N,j}, x_{1,j}^0, \ldots, x_{n,j}^0) &= x_{d,j}^1 \quad \text{and} \\ \chi_{d,j}^k(x_{N,j}, x_{1,j}^l, \ldots, x_{n,}^l) &= x_{d,j}^{l+k} \quad \text{for } l \in \mathbb{N} \end{aligned} \tag{4}$$

Further, we define a coupled RPDEVS model (3) to be

Definition 4. ***free of algebraic loops***

at time t_j, if $\forall d \in D,\ k \in \mathbb{N},\ x_i \in X_i,\ i \in D_N$ it fulfils:

$$\chi_{d,j}^k(x_N, x_1, \ldots, x_d, \ldots, x_n) = \chi_{d,j}^k(x_N, x_1, \ldots, \tilde{x}_d, \ldots, x_n)$$

$$\forall x_d \neq \tilde{x}_d \in X_d$$

In other words, a model is free of algebraic loops, if for every component d the content of its input bag after an arbitrary number of λ-iterations is not influenced by its (initial) content before these λ-iterations.

Theorem 5. If a RPDEVS model is *free of algebraic loops*, it is *legitimate*, that is, it exists a $k \in \{1, 2, \ldots, n\}$ with:

$$(x_{1,j}^k, x_{2,j}^k, \ldots, x_{n,j}^k) = (x_{1,j}^{k+1}, x_{2,j}^{k+1}, \ldots, x_{n,j}^{k+1})$$

Proof. As in the following, we will restrict our considerations to a fixed point in simulation time t_j, the index j will be omitted to improve readability.

In preparation for the proof, some definitions are made. First, we define the

Definition 6. ***influencer set*** I_d^k ***after*** k ***λ-iterations*** of a component $d \in D$ as

$$I_d^k = \begin{cases} I_d & , k = 0 \\ I_d^{k-1} \cup \bigcup\limits_{i \in I_d^{k-1}} I_i & , k > 0 \end{cases}$$

From the definition it is clear that $I_d^k \subseteq I_d^{k+1} \subseteq D$, and further that either $I_d^k \subsetneq I_d^{k+1}$ and thus $|I_d^{k+1}| \geq |I_d^k| + 1$, or that $I_d^k = I_d^{k+1}$. In the second case, we can deduced that also $I_d^{k+2} = I_d^{k+1} \cup \bigcup_{i \in I_d^{k+1}} I_i = I_d^k \cup \bigcup_{i \in I_d^k} I_i = I_d^{k+1}$, and therefore that $I_d^{k+l} = I_d^k$, $\forall l \in \mathbb{N}$. Consequently, $|I_d^k|$ must be strictly increasing with k until it sticks at a final value. This final value has to be smaller or equal $|D| = n$, because $I_d^k \subseteq D$ and thus we know that it will be reached at least for $k = n$ due to the strict monotony ($I_d^n = I_d^{n+1}$). Further, it can be derived that $\forall i \in I_d,\ I_i^n \subseteq I_d^n$.

Second, for an arbitrary set of initial input bags

$$\boldsymbol{x}^0 = (x_N, x_1^0, \ldots, x_n^0),$$

and for the corresponding vector of input bags after k λ-iterations

$$\boldsymbol{x}^k = (x_N, x_1^k, \ldots, x_n^k),$$

we define

$$\boldsymbol{x}_Z^k = (x_N, \tilde{x}_1^k, \ldots, \tilde{x}_n^k), \text{for } Z \subseteq D$$

with

$$\tilde{x}_d^k = \begin{cases} x_d^0 & \text{if } d \in Z \\ x_d^k & \text{else} \end{cases}$$

For example for $Z = D:\ \boldsymbol{x}_Z^k = \boldsymbol{x}^0$.

Now, consider a coupled RPDEVS with initial input bags $\boldsymbol{x}^0$. We will show that the input bag of a component d after $n+1$ λ-iterations is equal to the bag after n λ-iterations.

$$\begin{aligned} x_d^n &= \chi_d^n(x_N, x_1^0, \ldots, x_d^0, \ldots, x_n^0) = \chi_d^n(\boldsymbol{x}^0) \\ x_d^{n+1} &= \chi_d^n(x_N, x_1^1, \ldots, x_d^1, \ldots, x_n^1) = \chi_d^n(\boldsymbol{x}^1) \end{aligned}$$

Due to the assumption about our model to be free of algebraic loops, we know that the value x_d^1 does not influence x_d^{n+1}, and thus we can replace it by x_d^0: $\chi_d^{n-1}(\boldsymbol{x}^1) = \chi_d^{n-1}(\boldsymbol{x}_{\{d\}}^1)$. Furthermore, all input bags x_c^1 from components $c \notin I_d^n$ which do not influence component d at all can also be replaced by x_c^0. Therefore, we get for $Z_d = (D \setminus I_d^n) \cup \{d\}$:

$$x_d^{n+1} = \chi_d^n(\boldsymbol{x}^1) = \chi_d^n(\boldsymbol{x}_{Z_d}^1) = \biguplus_{i_1 \in I_d} \xi_{i_1,d}\left(\chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_d}^1)\right) \quad (5)$$

If $Z_d = D$, which would either mean $I_d^n = \emptyset$, or $I_d^n = \{d\}$, we are finished, as in this case

$$x_d^{n+1} = \chi_d^n(\boldsymbol{x}_{Z_d}^1) = \chi_d^n(\boldsymbol{x}^0) = x_d^n$$

Otherwise, for all $i_1 \in I_d$ we define $Z_{i_1} = Z_d \cup (D \setminus I_{i_1}^n) \cup \{i_1\}$. Because $i_1 \in I_d^n$ implies $i_1 \notin Z_d$, we know that $Z_d \subsetneq Z_{i_1}$, and therefore $|Z_d| < |Z_{i_1}| \leq n$.

For every $i_1 \in I_d$, $\chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_d}^1)$ in (5) can be calculated as:

$$\chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_d}^1) = \chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_{i_1}}^1) = \biguplus_{i_2 \in I_{i_1}} \xi_{i_2,i_1}\left(\chi_{i_2}^{n-2}(\boldsymbol{x}_{Z_{i_1}}^1)\right) \quad (6)$$

For all $i_1 \in I_d$ with $Z_{i_1} = D$, we are finished, as for those i_1 we get

$$\chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_d}^1) = \chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_{i_1}}^1) = \chi_{i_1}^{n-1}(\boldsymbol{x}^0),$$

and for all other $i_1 \in I_d$ we define $Z_{i_2} = Z_{i_2} \cup (D \setminus I_{i_2}^n) \cup \{i_2\}$ for all $i_2 \in I_{i_1}$. Now we can repeat the arguments from above regarding the calculation of $\chi_{i_2}^{n-2}(\boldsymbol{x}_{Z_{i_1}}^1)$ in (6):

$$\chi_{i_2}^{n-2}(\boldsymbol{x}_{Z_{i_1}}^1) = \chi_{i_2}^{n-2}(\boldsymbol{x}_{Z_{i_2}}^1) = \biguplus_{i_3 \in I_{i_2}} \xi_{i_3,i_2}\left(\chi_{i_3}^{n-3}(\boldsymbol{x}_{Z_{i_2}}^1)\right)$$

For the index sets we have $Z_d \subsetneq Z_{i_1} \subsetneq Z_{i_2} \subsetneq \ldots$. The case $Z_d = \emptyset$ has already been treated above. Consequently, $|Z_{i_l}| \geq l + 1$ and therefore, at least for i_{n-1} it is $Z_{i_{n-1}} = D$ $\forall i_{n-1} \in I_{i_{n-2}}$. This implies

$$\chi_{i_{n-1}}^1(\boldsymbol{x}_{Z_{i_{n-2}}}^1) = \chi_{i_{n-1}}^1(\boldsymbol{x}_{Z_{i_{n-1}}}^1) = \chi_{i_{n-1}}^1(\boldsymbol{x}^0),$$

and, because of

$$\chi_{i_{n-2}}^2(\boldsymbol{x}_{Z_{i_{n-2}}}^1) = \biguplus_{i_{n-1} \in I_{i_{n-2}}} \xi_{i_{n-1},i_{n-2}}\left(\chi_{i_{n-1}}^1(\boldsymbol{x}^0)\right) = \chi_{i_{n-2}}^2(\boldsymbol{x}^0)$$

the argument $\boldsymbol{x}^0$ of the χ-functions propagates back until finally

$$\chi_{i_1}^{n-1}(\boldsymbol{x}_{Z_d}^1) = \biguplus_{i_2 \in I_{i_1}} \xi_{i_2,i_1}\left(\chi_{i_2}^{n-2}(\boldsymbol{x}^0)\right) = \chi_{i_1}^{n-1}(\boldsymbol{x}^0)$$

and thus

$$x_d^{n+1} = \biguplus_{i_1 \in I_d} \xi_{i_1,d}\left(\chi_{i_1}^{n-1}(\boldsymbol{x}^0)\right) = \chi_d^n(\boldsymbol{x}^0) = x_d^n$$

□

3.4 Closure Under Coupling

Like both Classic and Parallel DEVS, RPDEVS also provides *closure under coupling*, which we now will show.

Consider an arbitrary coupled RPDEVS N as defined in (3). Closure under coupling means, that N can be substituted by an atomic RPDEVS

$$\alpha =< X_\alpha, S, Y_\alpha, \delta, \lambda, ta > . \quad (7)$$

In order to proof closure under coupling all parts of the definition of α have to be provided by using the functions and sets defining the n components and the couplings of N.

Input and output sets, the state of the system, and the time advance function can be defined straightforward:

$$\begin{aligned} X_\alpha &= X_N \\ Y_\alpha &= Y_N \\ s &= (s_1, e_1,\ s_2, e_2,\ \ldots,\ s_n, e_n) \\ ta(s) &= min\{ta(s_i) - e_i :\ i \in \{1, \ldots, n\}\} \end{aligned}$$

For the output function λ and the state transition function δ, we define the following terms for a given instant of time:

$IMM := \{d : ta(s_d) = e_d\}$ ... set of imminents
$INF := \{d : x_d \neq \emptyset\}$ ... set of influenced components

Thereby, x_d is the final input bag of component d, resulting from the λ-iteration:

$$x_d = \chi_d^n(x_N, \emptyset, \ldots, \emptyset)$$

With this definitions, we can specify λ as

$$\lambda(s, e, x_N) = \biguplus_{d \in I_N} \xi_{d,N}(x_d), \tag{8}$$

and the state transition function δ as

$$\delta(s, e, x_N) = (s'_1, e'_1,\ s'_2, e'_2,\ \ldots,\ s'_n, e'_n), \tag{9}$$

with $\forall d \in D$:

$$(s'_d, e'_d) = \begin{cases} (\delta_d(s_d, e_d + ta(s), x_d), 0) & d \in IMM \cup INF \\ (s_d, e_d + ta(s)) & \text{otherwise} \end{cases}$$

4. DISCUSSION AND CONCLUSION

After recapping PDEVS and describing how sequentially processing of inputs occurs in PDEVS in example 1, we introduced RPDEVS. Using RPDEVS instead of PDEVS in example 1 would result in the following function calls:

$$\begin{aligned}
1^{st}\lambda - iteration:\ & x_{a,c} = Z_{a,c}(\lambda_a(s_{a,0}, e_{a,1}, \emptyset)) \\
& x_{a,d} = Z_{a,d}(\lambda_a(s_{a,0}, e_{a,1}, \emptyset)) \\
2^{nd}\lambda - iteration:\ & x_{c,d} = Z_{c,d}(\lambda_c(s_{c,0}, e_{c,1}, \{x_{a,c}\})) \\
& y_d = \lambda_d(s_{d,0}, e_{c,1}, \{x_{a,d}\}) \\
3^{rd}\lambda - iteration:\ & y_d = \lambda_d(s_{d,0}, e_{c,1}, \{x_{a,d}, x_{c,d}\}) \\
\delta - phase:\ & s_{a,1} = \delta_a(s_{a,0}, e_{a,1}, \emptyset) \\
& s_{c,1} = \delta_c(s_{c,0}, e_{c,1}, \{x_{a,c}\}) \\
& s_{d,1} = \delta_d(s_{d,0}, e_{d,1}, \{x_{a,d}, x_{c,d}\})
\end{aligned}$$

Compared to the sequence in PDEVS, instead of having alternating λ- and δ-phases, we have repeating λ-iterations followed by one single δ-phase. In the δ-phase the final input bags are available. This allows modelling components with mealy behaviour without having to use transitory states. It is still possible to change into a transitory state in RPDEVS, forcing multiple δ-phases (i.e. multiple state updates) in one instant of time. However, it is not necessary any more in order to achieve mealy behaviour, which puts the general necessity of transitory states into question. As long as no transitory states are used in RPDEVS, there is no sequential processing of concurrent input messages. Components can be modelled independently of their possible surroundings and thus, show transparent behaviour. When defining λ and δ for a reusable component, only the component's state and the possible sets of input messages need to be considered, without having to keep in mind that there may be a partially sequential processing of concurrent input messages.

In PDEVS, the δ function always has to store the history of the current λ-δ-iteration in the state, as succeeding calls of λ and δ have to consider which input messages arrived already in former iterations and which output messages have been produced then. Moreover, output messages generated in former iterations may became invalid after the arrival of additional inputs in a subsequent iteration.

In RPDEVS, λ becomes more complex than in PDEVS, as it is equipped with additional arguments (input bag, elapsed time). However, when defining λ in RPDEVS, preceding calls in the current λ-iteration do not have to be considered and formerly created output messages are overwritten automatically by the simulation algorithm.

For a complete introduction of RPDEVS, the formal definition of an abstract simulator is missing. Such a definition already exists and will be provided in the future. Nevertheless, a proof-of-concept implementation of a concrete RPDEVS simulator is already available with *PowerRPDEVS* [1] . PowerRPDEVS is PowerDEVS with a reprogrammed simulation engine that supports RPDEVS.

REFERENCES

Chow, A.C.H. and Zeigler, B.P. (1994). Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In *Proc. of Winter Simulation Conference'94*, 716–722. Society for Computer Simulation International.

Chow, A.C., Zeigler, B.P., and Kim, D.H. (1994). Abstract simulator for the parallel DEVS formalism. In *Proc. of the Fifth Annual Conference on AI, and Planning in High Autonomy Systems*, 157–163.

Cicirelli, F., Furfaro, A., and Nigro, L. (2007). Conflict management in PDEVS: an experience in modelling and simulation of time petri nets. In *Proc. of Summer Computer Simulation Conference (SCSC'07)*, 349–356. Society for Computer Simulation International.

Franceschini, R., Bisgambiglia, P.A., Touraille, L., Bisgambiglia, P., and Hill, D. (2014). A survey of modelling and simulation software frameworks using discrete event system specification. In *Proc. of 2014 Imperial College Computing Student Workshop*, 40–49.

Goldstein, R., Breslav, S., and Khan, A. (2013). Informal DEVS conventions motivated by practical considerations. In *Proc. of Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, 10:1–10:6.

Junglas, P. (2016). Pitfalls using discrete event blocks in simulink and modelica. In *Tagungsband des Workshops der ASIM/GI-Fachgruppen STS und GMMS 2016*, 90–97. ARGESIM Verlag Wien, Hochschule Hamm-Lippstadt 2016.

Preyser, F.J. (2015). An approach to develop a user friendly way of implementing DEV&DESS models in PowerDEVS. Masterthesis, Vienna University of Technology.

Preyser, F.J., Heinzl, B., Raich, P., and Kastner, W. (2016). Towards extending the parallel-DEVS formalism to improve component modularity. In *Beiträge zum Work. der ASIM/GI-Fachgruppen STS und GMMS 2016*, 83–89. ARGESIM Verlag Wien, Hochschule Hamm-Lippstadt 2016.

Raich, P., Heinzl, B., Preyser, F., and Kastner, W. (2016). Modeling techniques for integrated simulation of industrial systems based on hybrid PDEVS. In *Proc. of 2016 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 1, 1–6. Vienna.

Traoré, M.K. (2007). Easy DEVS. *In Proc. of the 2007 spring simulation*, 1, 214–216.

Zeigler, B.P., Praehofer, H., and Kim, T.G. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

[1] available at: *https://sourceforge.net/projects/powerrpdevs/*