Modeling and Simulation of a Real-world Application using NSA-DEVS

David Jammer[1,2], Peter Junglas[2*], Thorsten Pawletta[1], Sven Pawletta[1]

[1]Research Group Computational Engineering and Automation, University of Applied Sciences Wismar, Philipp-Müller-Straße 14, 23966, Wismar, Germany;

[2]PHWT-Institut, PHWT Vechta/Diepholz, Am Campus 2, 49356 Diepholz, Germany; **peter@peter-junglas.de*

SNE 33(4), 2023, 149-156, DOI: 10.11128/sne.33.tn.10662
Received: 2023-11-03; Revised: 2023-11-25
Accepted: 2023-11-26
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The recently proposed NSA-DEVS formalism uses infinitesimal time delays to combine the easy implementation of Mealy components from RPDEVS with the simple simulator structure of PDEVS in order to make DEVS a suitable foundation for complex component-based modeling tasks. To prove its general applicability, it is used here to implement a large real-world model describing a production line that consists of several furnaces, lathes and grinders. Using a Matlab implementation of the simulator algorithm, the model behaviour is analyzed carefully and it is found that only one of its over 400 infinitesimal time delay parameters has to be changed from its default value. This shows the soundness of the basic ideas of NSA-DEVS and its applicability for real-world examples.

Introduction

A well-known difficulty that often arises when using the discrete event approach is the occurence of events at the same time instant. There are two very different reasons for such a behaviour: The first one is the *accidental coincidence* of events, e. g. when two input events from different sources arrive at the same time, or when an event arrives at a component at the exact moment of a state change. In such cases the exact ordering of these events inside a concrete simulator is often of no consequence, therefore providing an opportunity for parallel execution. The other reason is a chain of events that is created by an initial event, causing an immediate (*transitory*) state change that leads to further events that spread through the system without delay. Even if all these events formally appear at the same time instant, their logical relation enforces a fixed ordering of such a *causal cascade*.

In the context of the widely used DEVS formalism [1] the behaviour of a model and its simulator are defined precisely. Different variants of the formalism provide specific mechanisms to fix the order of concurrent events, where necessary: Classic DEVS uses a *Select* function on the level of coupled components, while PDEVS introduces a *confluent state transition* function inside an atomic component. The revised PDEVS formalism (*RPDEVS*) [2] incorporates a direct Mealy structure with a generic state transition function and a refined simulator algorithm [3].

In order to simplify the modeling of causal cascades and the modeling of Mealy behavior, a new approach named NSA-DEVS (*Non-Standard Analysis DEVS*) was proposed in [4]. It is based on the physical intuition that the transport and the processing of events always need a certain amount of time, so that the simulation problems are actually due to oversimplification. But instead of introducing a plethora of small delay parameters, NSA-DEVS uses infinitesimal time delays. Therefore it defines time values as elements of the hyperreals ${}^*\mathbb{R}$, which is a mathematically well-defined field extension of $\mathbb{R}$ containing an infinitesimal $\varepsilon > 0$ [5].

The definition of a simulator [6] and the careful analysis of a set of standard examples [7] have shown the basic soundness of the new formalism. But to prove its general applicability, one has to test it with a large non-trivial example. For this purpose a model of a production line will be studied in the following that consists of several furnaces, lathes and grinders. The model includes the material flow, several process controllers and basic physical properties.

After a short recapitulation of the NSA-DEVS formalism, the model and its implementation in Matlab will be described, highlighting some special atomic and coupled components. Finally, the simulation of the model and careful analysis of the results will clarify the main question: How many of the delay parameters have to be changed from their default values, and how difficult is it to find them? Or, in other words: Is the NSA-DEVS formalism suitable for real-world applications?

1 The NSA-DEVS Formalism

For the conveniance of the reader the basic definitions of the NSA-DEVS model specification will be given here. A more detailed description, the connection with the PDEVS and RPDEVS formalisms and the definition of the abstract simulator can be found in [6].

Two types of models are defined in all DEVS variants: an atomic model that describes the behaviour of a single component, and a coupled model, which specifies how models are combined to build a hierarchical structure. In NSA-DEVS an atomic model is given by a 7-tuple $< X, S, Y, \tau, ta, \delta, \lambda >$ with

X	set of input ports and values,
S	set of states,
Y	set of output ports and values,
$\tau \in {}^*\mathbb{R}^{>0}_{\text{fin}}$	input delay time,
$ta : S \to {}^*\mathbb{R}^{>0}_{\text{fin}} \cup \{\omega\}$	time advance function,
$\delta : Q \times X^+ \to S$	state transition function,
$\lambda : Q \times X^+ \to Y^+$	output function.

The input and output sets X, Y contain pairs of ports and values, where ports are given by names. The sets X^+, Y^+ consist of sets of values from X, Y to describe the simultaneous appearance of input or output values at different ports. The definition of the transition function δ and the output function λ contain the set $Q = \{(s,e) | s \in S, 0 < e \leq ta(s)\}$ that combines a state and the elapsed time e since the last transition.

As in RPDEVS, both event types (incoming event or internal event) lead to a call of λ followed by a change to a new state according to δ. The time advance function ta may be infinitesimal or infinite (using $\omega := 1/\varepsilon$, with ε as infinitesimal value), but it is always > 0, thereby excluding proper transitory states. The delay time τ between the arrival of a set of inputs and the call of λ and δ is generally an infinitesimal, often given by a default value $\tau_{def} = \varepsilon$.

A coupled NSA-DEVS model is defined as in PDEVS and RPDEVS. It consists of input and output ports and a set of atomic or coupled models, which are connected among themselves and to the external ports. Outputs are transported as usual and a coupled component has no additional input delays.

2 A Real-world DEVS Application

The model and its components described in this article were developed similar to [8, 9]. The model describes a production line which includes lathes, grinders and furnaces (cf. Figure 1). In this production line, in the first step, the raw parts supplied by a generator are processed by lathes. After lathing, the components are thermally treated. This is done first by a furnace for volume hardening (*furnace 1*) and then by a furnace for stress relief annealing (*furnace 2*). The two furnaces have no difference in design, but only in their temperature behaviour. The furnaces always process several parts at the same time, depending on the type of furnace. After heat processing, the components are finished by grinders. The manufacturing operations are decoupled via buffers with a maximum capacity of 400 parts. The buffers of the lathes and grinders are located in the coupled model of the respective manufacturing operation (cf. Figure 2). The buffers are organized in a coupled model whereby every single machine has its own buffer (cf. Figure 3). This model corresponds to example 4 from [7].

The lathe, grinder and furnace machine models have all the same internal structure according to [10]. The structure is divided into a physical model (PM), control model (CM) and material flow model (MF) (cf. Figure 4). The PM describes the physical relationships, e. g. the heat flows, by differential equations. The CM contains the local machine control, which takes into account various internal processing steps. In MF, the internal material flow is modeled in a process-oriented way, describing the parts as moving entities.

The machine models return the process variables electrical power, electrical energy and utilization over time. The buffers deliver information about the current load and a terminator reports the number of finished parts. These process variables are used to calculate the Key Performance Indicator (KPI) variables buffer stock (*pcStock*), throughput (*thrput*), load peak (*loadPeak*), utilization (*pcUtil*), energy per part (*eSpec*) and production time per part (*procTime*).

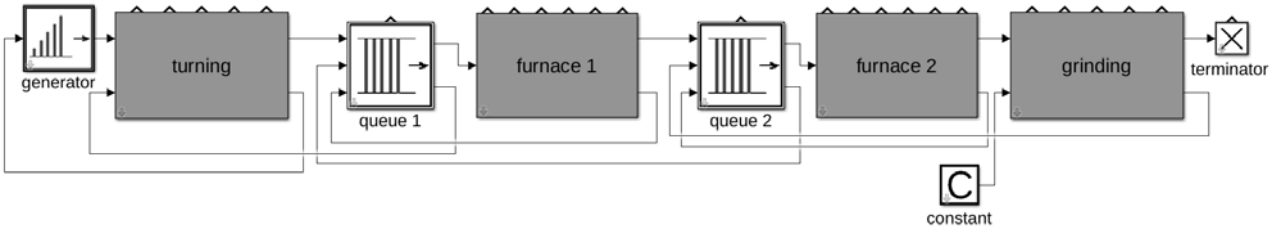

Figure 1: Top level structure of the production line.

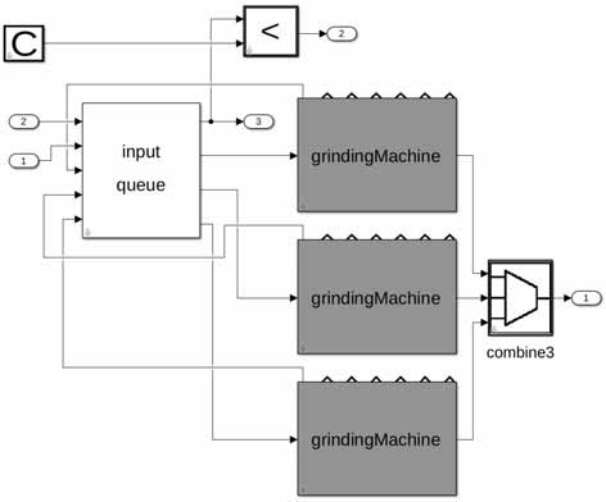

Figure 2: Substructure of the grinding component in Fig. 1.

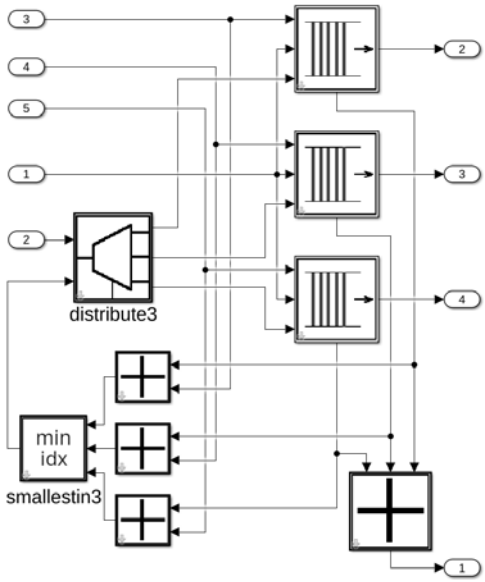

Figure 3: Substructure of the input queue in Fig. 2.

The entire model consists of 391 atomic models and 88 coupled models and is organized in 5 hierarchical levels.

3 Atomic Components

Using the Matlab-based NSA-DEVS simulator from [11], an atomic model is implemented as a class, which defines the transition and the output function as methods `delta(obj,e,x)` and `lambda(obj,e,x)` and the time advance function as method `ta(obj)`. The argument `obj` is the handle (reference) to the object, allowing access to its state, `e` is the time since the last transition, given as a two-dimensional vector `[a, b]` denoting a hyperreal value $a + b\varepsilon$, and `x` is a structure, containing the current input as field/value pairs, where the field is identical to the port name. The constructor always defines the name of the component, the input delay τ and a debug flag, as well as optional model parameters.

Since NSA-DEVS retains the mealy-type behaviour of RPDEVS, standard computational components can be implemented easily. The main difference to components used in a continuous modeling environment is due to the event-based paradigm applied here: At least for components with more than one input port, one needs a state variable for every input port to store incoming values, because an input is only defined at the time of the corresponding input event. Taking this into account, one can implement a simple adding component by providing a δ function that just stores the input values, a *ta* function that always returns `[inf, 0]`, and a λ function that returns the sum of the input values, using the stored values, where necessary.

A standard library of atomic components has been built for the models described in [7] and the production line, containing mathematical and routing components, simple source and sink components and several logistics related components such as a queue and a server.

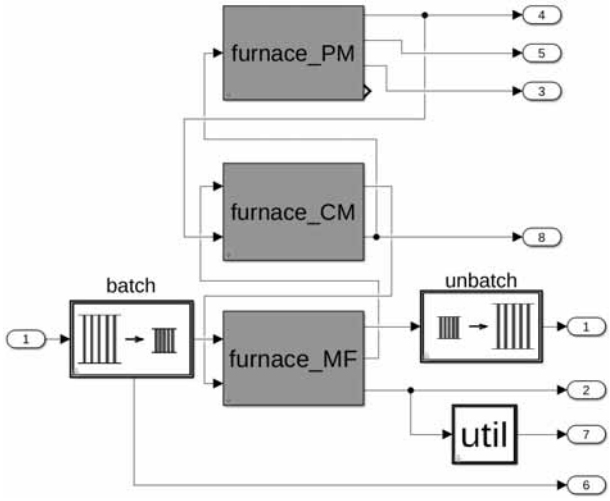

Figure 4: Substructure of the furnace components in Fig. 1.

Several of them have been described in [7], among them an atomic model `ToWorkspace` that can be connected to an output port and copies the incoming values to a global output variable, which can be analysed after the simulation run. For the modeling of the furnace a batch component has been added that combines a given num-ber of incoming entities into one batch entity, together with a corresponding unbatch atomic model.

Since some physical variables of the production line are modelled by differential equations, e. g. the temper-ature inside a furnace or the electric current in a grind-ing machine, an integrator component is needed in-side the NSA-DEVS environment. For this purpose the Quantized-State-Systems (QSS) method is used [12], which has already been implemented inside a Matlab-based DEVS simulator [13].

In addition, a few atomics have been added that are specific to the production l ine: a special server component, controller components for all machine types and a few conveniance components, which could have been built as coupled models from atomics in the standard library. Altogether the production line model uses 15 types of atomics from the standard library and 7 from its own library.

4 Coupled Components

In the NSA-DEVS simulator used here, a coupled model is not implemented as a class, but simply given by a constructor function, which defines a ll elements of the coupled-model specification: its internal atomic and coupled models and their connections among themselves and to the external ports. Furthermore, it assigns its atomic models to simulator modules and its coupled models – including the currently defined one – to coordinator modules (cf. [6]).

Especially for large coupled models, the programming of such a constructor function is tedious and error prone. Therefore a graphical model generator is provided – similar to the approach in [14] – that creates the function from a graphical description. For this purpose, atomic models are represented in Simulink libraries as blocks that only contain the external ports, using masks to define their parameters (cf. Figure 5). Coupled models can then be defined a s s tandard S imulink subsystems consisting entirely of such blocks, subsystems and external ports. The model generator creates all needed constructor functions, where the top-level model can be directly run in the simulator. The corresponding model of the production line is shown in Figure 1. It consists of three atomics – a generator, a constant and a terminator – and coupled systems for the furnaces, lathes, grinders and intermediate queues.

At the inner hierarchy levels many models look very much like similar models in continuous simulation environments like Simulink. A good example is the component that computes the furnace temperature using the simple differential equation

$$C_O \frac{dT}{dt} = k_A(T - T_e) + P_{\text{heating}} - P_{\text{unload}} \,.$$

It is built exactly like a corresponding Simulink model (cf. Figure 6).

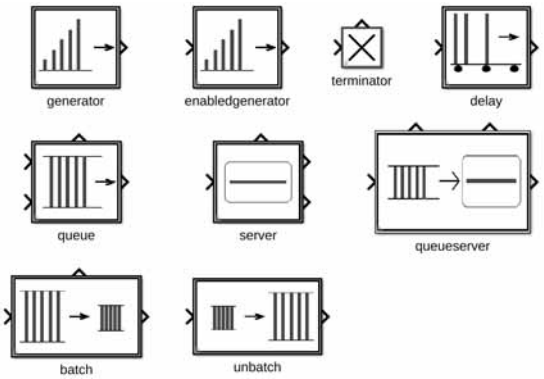

Figure 5: Library of logistics related components.

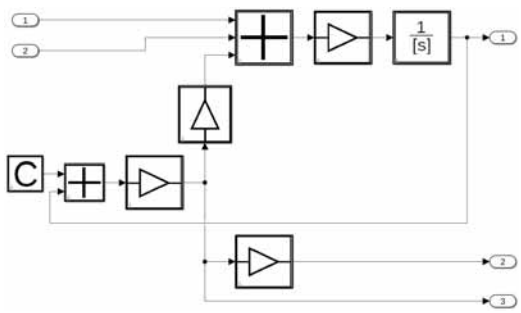

Figure 6: Model for the computation of the furnace temperature.

Though this correspondence may sometimes be helpful for the definition of suitable atomics and the construction of coupled models, it is basically superficial. The main difference is, of course, the meaning of the connecting lines: In continuous simulation environments a line transports a signal that has always a value, while in a DEVS environment a value is only defined at the moment of an input event. This has practical consequences for the concrete modeling, e. g. for the question, whether it is possible to connect an output port directly to several input ports (*1:N connection*) or vice versa (*N:1 connection*).

Using a signal paradigm as in Simulink, the 1:N direction clearly is possible, distributing the value to all input ports, while the reverse direction is incorrect, since in this case the input value is not well defined. In PDEVS both directions are allowed: Several events can arrive at the same input port, even simultaneously, while an output event is copied automatically by the coordinator, when it is distributed to several components. In NSA-DEVS the situation is different again: While N:1 connections are possible in principle, the abstract simulator does not support the simultaneous arrival of input events at the same port. If this is required – e. g. for modeling of multi-value logic components [15] – one has to use corresponding connection atomics explicitely. The 1:N direction is handled by the coordinator as in PDEVS and will often be used, e. g. to attach `ToWorkspace` components directly to a line or to distribute events transporting simple values. If the data of an event is interpreted as an entity, as in transactional-based modeling, it would be better though to include explicit copy components to make the intention clear.

5 Testing and Running the Model

We will now try to run the complete production line model and interpret the simulation results. The main point of interest here is, how to set all the infinitesimal parameters. They consist primarily of the 391 input delay times τ. Furthermore one needs 12 additional delay parameters for the transitory states that are used in the queue, unbatch and combine atomics, which will be denoted as τ_D. All these values are usually predefined and set to the value $\tau_{def} = \varepsilon$. From the analysis of a few example models in [7] the following situations have been identified, where one has to change some of the parameters from their default value:

a.) Input events that appear during the input delay time of a component, overwrite a previous input value at the same port, which sometimes is useful, but more often not. A particular example is a combine component that serializes concurrent incoming inputs: These should be output with a sufficiently large delay time, so that subsequent components can process them one after the other.

b.) To make a queue-server combination work, the blocking signal from the server has to arrive before a second entity is output by the queue. In standard situations it is sufficient to set the delay time of the "transitory" queue state to 2ε, therefore this value is defined as default in the library queue block.

c.) In loops containing several sequences of components the order of concurrent events depends on the total delay times along different paths. If one wants to implement a specific ordering, one can slow down some paths by increasing appropriate delays.

Since problems due to wrong delay values often lead to missing entities, a simple test strategy is to insert a fixed amount of input entities and run the model, until all entities should have reached the output. We start with default parameters, i. e. all τ and τ_D values are set to ε, except the τ_D values of queue atomics, which are set to 2ε. The simulation run shows that no entities reach the final terminator and that the total amount `pcStock` of entities in internal queues goes down to zero, i. e. all entities are lost.

Taking into account the lesson from a.) the culprits are quickly identified as the unbatch and combine atomics: They release entities in groups with only the default

delay in between, so that these entities overwrite each other during the input delay of the following component. An obvious solution seems to be to enlarge the τ_D values of both unbatch and both combine atomics to 2ε. Running the modified model, the simulator hangs and must be terminated manually. This can happen, when the overwriting of inputs happens in a loop without reaching a result.

To localize the problem, one has to look more closely at the sequence of events using the debugging possibilities of the simulator. Debugging a complex discrete-event based application is always a difficult endeavour. Therefore one usually starts by reducing the complexity and looking at test benches for basic subsystems. Though this approach has been followed here to find the usual bugs, it is not sufficient to fix all τ values, because some important loops only show up in the complete model.

When setting a debug flag, the simulator outputs the current simulation time. This shows that the simulation is stuck in a loop at the first time, when a complete batch of entities leaves the second furnace (Figure 4) and enters the grinding section (Figure 2). The critical region therefore seems to be the `unbatch` atomic inside `furnace 2` and the `input_queue` component (Figure 3) inside `grinding`. Figure 8 shows this model part with a view of the internal structure of the coupled subsystems. Guessing from previous experience, one would suspect that the τ_D parameter of `unbatch` is too small. With a value of 10ε for the unbatch components in both furnace subsystems, the model works, no entities are lost, the output is as expected (Figure 7).

Since the problem encountered is typical for the behaviour of NSA-DEVS, we will show in detail, how one can use the debugging features of the simulator to find the concrete source of the error (cf. Figure 8). First, one adds `toWorkspace` components inside `input_queue` that show the number of entities in the three queues and the ids of outgoing entities. Unfortunately, their results are not directly available, when the simulation run is interrupted manually. Here, another debugging feature is useful: All atomics have `debug` flags, which can be switched on individually to create outputs of their input and output values and all state changes during the simulation run. This feature is used here for the six `toWorkspace` components and the `distribute3` component at the input of the `input_queue` coupling.

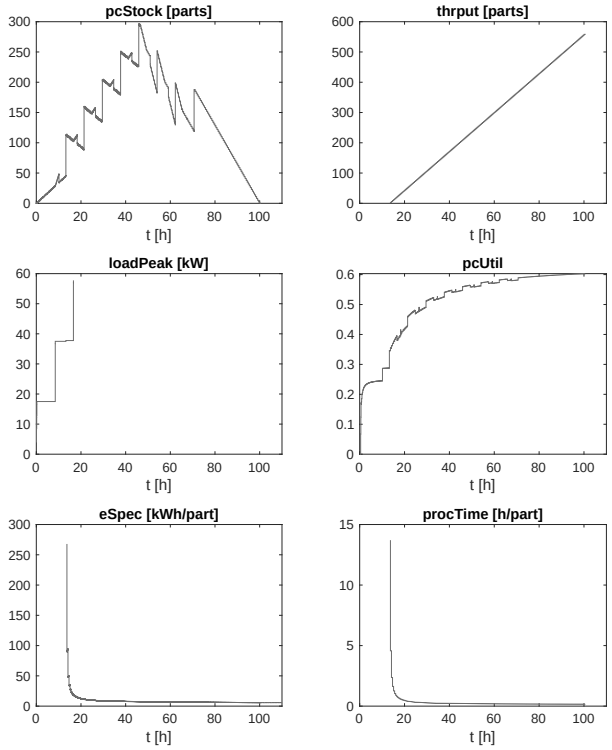

Figure 7: Simulation results showing KPI variables of the production line model.

Analyzing the debugging outputs of the working version, which uses $\tau_D = 10\varepsilon$ for the unbatch component, shows the following behaviour: The first three entities are distributed to the three queues, leave their queue immediately and enter the server inside the corresponding grinding machine. This leads to several changes of the `port` input of `distribute3`, which points to the currently shortest queue/server line. The input events at `port` are delayed, because the event cascade has to pass the queue, server, add and `smallestIn` atomics. Some of them are overwritten by the following ones, as can easily be seen in the debugging log: In such a case the lambda function of a component is called several times without an intervening call to the delta function. But this only effects intermediate values here and doesn't lead to erroneous behaviour. The next entities (no. 4, 5, ...) are stored in the queues, the associated new `port` values arrive and the `distribute3` component is ready each time, before the next entity enters the `input_queue`, due to the long τ_D-delay inside `unbatch`.

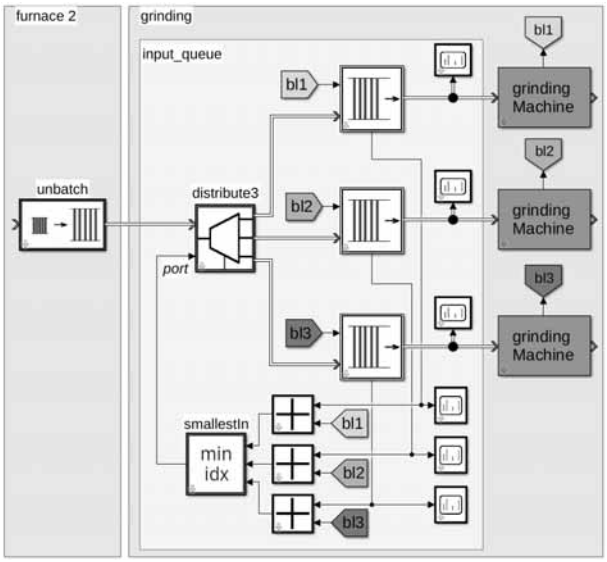

Figure 8: Critical part of the production line model.

The previously considered version with the smaller value of $\tau_D = 2\varepsilon$ in `unbatch` shows a completely different sequence of actions: The first two entities are distributed to queue 1, the third to queue 2, since these entities arrive before the `port` input could adapt to the changes.

After the first entities the queues are blocked. This reduces the delay chain of the `port` signal, because the servers inside the grinding machines are no longer involved, with drastic consequences: The new entities now arrive at the same time as new `port` values from the previous entity. Therefore the old values are overwritten and new lambda calls scheduled, without any intervening delta calls.

This leads to a loss of all entities, until the last of the batch arrives. But still the new `port` numbers come in, therefore the loop doesn't end even now.

This behaviour makes clear that τ_D of unbatch and similar components must always be large enough so that the following components and loops are ready, before the next entity arrives. Taking a larger value does no harm, but by adding up delays or just trial and error one finds, that using $6\,\varepsilon$ works here, but $5\,\varepsilon$ doesn't. Additionally, only the unbatch component of the second furnace has to be adapted, while a default value of 2ε for the first unbatch works perfectly.

6 Conclusions

The central point of this investigation is the question, whether one can use NSA-DEVS without tinkering with a huge number of additional parameter adjustments of τ for the input delays and τ_D for the delays of transitory states. The previous discussion has shown that a consistent introduction of default values is crucial here: While setting $\tau = \varepsilon$ and $\tau_D = \varepsilon$ seems to work in many cases, components that emit trains of output values with infinitesimal time distances, such as queue, batch and combine atomics, need special consideration. Often a value of $\tau_D = 2\varepsilon$ is working and should be predefined in corresponding library components.

In special situations one has to enlarge these delays, but concrete values depend on the model details. Using a larger delay as general default could reduce the number of cases, where the user has to adapt it, but this apparent simplification is delusive.

The complex application studied here contains a total of 403 delay parameters, of which only one parameter had to be changed from its default value. This clearly supports the expectation that modeling with NSA-DEVS is feasible without drowning in a multitude of delay parameter adjustments.

In a current PhD project, the production chain described in this article is used as an application example, where it is integrated into the structure of an Experimental Frame [1]. The goal of the experiment is to optimize the structure and parameters of the model. As a first step, a parameter study has been carried out using the Design of Experiment method. This extended model works without further adaptations.

Considerations about the correct ordering of simultaneous events are not specific to NSA-DEVS, but are inherent to discrete-event modeling in general. Accordingly, other DEVS variants have different ways, how to cope with these situations. We see the advantage of NSA-DEVS in the clear-cut and versatile description of the ordering using infinitesimal time values. To make this work in practical applications, one needs support by the simulation tools.

The Matlab-based NSA-DEVS simulator used here supplies several debugging tools ranging from simple time stamps over debug output of individual atomic components up to complete output of internal simulator messages.

The analysis has shown opportunities to improve the implementation of some components. A prominent example is the smallestIn atomic that could be modified to output new values only, if the previous value changes. This reduces the number of output events preventing the infinite loop that has been encountered before. A more drastic change would be to include only one queue before the grinding machines and to distribute the parts after the queue. This would simplify the event structure and might be a reasonable idea for a production line design. But the whole point is to provide a tool that works for all modeling ideas, not to restrict the modeling to the tool capacities.

An open question from [6, 7] was, whether one needs port specific input delay times. The answer after this study is a definitive "no": In all examples the default value of the input delay was sufficient for all input ports to get a reasonable model. A change of an input delay would be needed only to guarantee a certain order of unrelated input events. Besides, one could always resort to the workaround of inserting a simple delay component – e. g. a gain with factor 1 – before a specific input port.

This concludes the series of papers [4, 6, 7] that investigated the ideas behind NSA-DEVS as a foundation for component-based DEVS modeling. Building on RPDEVS [2], which made Mealy components simple and reliable, the NSA-DEVS approach was invented to add the robust modeling of causal event cascades. The definition of a simple abstract simulator, the careful analysis of standard examples and the implementation of a complex application have shown the soundness of its underlying ideas. The Matlab simulator and model base will be continually extended and provided freely from [11] to make NSA-DEVS-based modeling available for real-world applications.

References

[1] Zeigler BP, Muzy A, Kofman E. *Theory of Modeling and Simulation*. San Diego: Academic Press, 3rd ed. 2019.

[2] Preyser FJ, Heinzl B, Kastner W. RPDEVS: Revising the Parallel Discrete Event System Specification. In: *9th Vienna Int. Conf. Mathematical Modelling*. Wien. 2018; pp. 242–247.

[3] Preyser FJ, Heinzl B, Kastner W. RPDEVS Abstract Simulator. *SNE Simulation Notes Europe*. 2019; 29(2):79–84. doi: 10.11128/sne.29.tn.10473.

[4] Junglas P. NSA-DEVS: Combining Mealy Behaviour and Causality. *SNE Simulation Notes Europe*. 2021; 31(2):73–80. doi: 10.11128/sne.31.tn.10564.

[5] Goldblatt R. *Lectures on the Hyperreals*. New York: Springer. 1998.

[6] Jammer D, Junglas P, Pawletta T, Pawletta S. A Simulator for NSA-DEVS in Matlab. *SNE Simulation Notes Europe*. 2023;33(4):141–148. doi: 10.11128/sne.33.sw.10661.

[7] Jammer D, Junglas P, Pawletta T, Pawletta S. Implementing Standard Examples with NSA-DEVS. *SNE Simulation Notes Europe*. 2022;32(4):195–202. doi: 10.11128/sne.32.tn.10623.

[8] Larek R. Ressourceneffiziente Auslegung von fertigungstechnischen Prozessketten durch Simulation und numerische Optimierung (Resource-efficient Design of Manufacturing Process Chains by Simulation and Numerical Optimization). Ph.D. thesis, Universität Bremen. 2012.

[9] Schmidt A. Variantenmanagement in der Modellbildung und Simulation unter Verwendung des SES/MB Frameworks (Variant Management in Modeling and Simulation using the SES/MB Framework). Ph.D. thesis, Hochschule Wismar / Universität Rostock. 2019.

[10] Pawletta T, Schmidt A, Junglas P. A Multimodeling Approach for the Simulation of Energy Consumption in Manufacturing. *SNE Simulation Notes Europe*. 2017; 27(2):115–124. DOI: 10.11128/sne.27.tn.10377.

[11] CEA Wismar. *NSA-DEVS on GitHub*. `https://github.com/cea-wismar/NSA-DEVSforMATLAB`.

[12] Cellier FE, Kofman E. *Continuous System Simulation*. New York: Springer. 2006.

[13] Schwatinski T, Pawletta T. A Quantization-based ODE Approximation and HPP-LGCA Approach to ARGESIM Benchmark C17 'SIR-type Epidemic' in a DEVS Environment based on MATLAB. *SNE Simulation Notes Europe*. 2011;21(1):57–60. doi: 10.11128/sne.21.bn17.10053.

[14] Bergero F, Kofman E. PowerDEVS. A Tool for Hybrid System Modeling and Real Time Simulation. *Simulation: Transactions of the Society for Modeling and Simulation International*. 2011;87(1-2):113–132.

[15] IEEE Design Automation Standards Committee. *Std 1164-1993, IEEE Standard Multivalue Logic System for VHDL Model Interoperability*. IEEE, New York, USA. 1993.