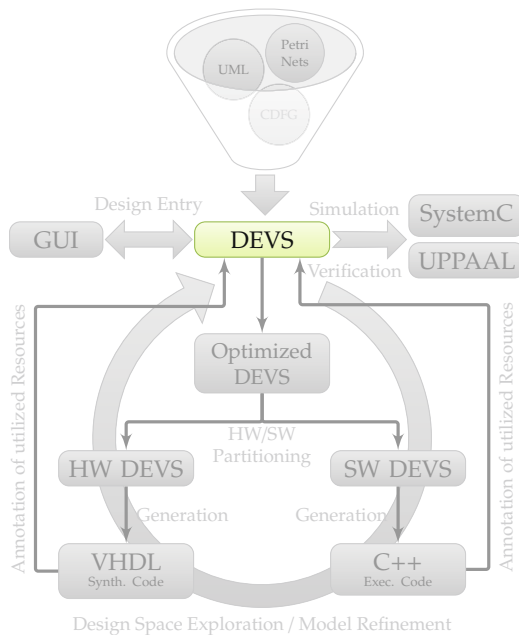**CHAPTER 2**

# Discrete Event System Specification

In this chapter, the classic DEVS formalism from Zeigler et al. will be introduced. It will be discussed in depth, which properties of the DEVS MoC impedes its exploitation for a hardware / software co-design flow. Thus, the SynDEVS MoC will be introduced which thoroughly refines the DEVS MoC to allow synthesis of SynDEVS models. At the end of this chapter, a graphical notation will be presented which is very beneficial regarding the readability of a model in contrast to the original pure mathematical notation.

## 2.1 Classic DEVS with Ports and Parallel DEVS Coupled Models

The DEVS formalism for discrete event systems introduced by Zeigler et al. in [ZKP00] is a strong mathematical foundation for specifying hierarchically, concurrently executed formal models. It affords a computational basis for specifying the behaviour of such models in a clear simplified manner. Zeigler et al. specified the so-called Classic DEVS with Ports, a structure for describing single, sequentially executed, timed automata for discrete event systems. Afterwards, this concept was further developed to cope with recent developments of computer architectures (i.e. the increasing degree of parallelization). Parallel DEVS Coupled Models was introduced allowing the modelling of concurrently executed, hierarchically, timed automata for discrete event systems. As a result, the Classic DEVS with Ports was embedded into Parallel DEVS Coupled Models by means of atomic components and will be detailed later on in Subsection 2.1.2. However, before discussing the atomic components, which implement the behaviour of the models, the Parallel DEVS in terms of parallel components will be introduced in the following section.

### 2.1.1 Parallel Components

Parallel components within DEVS allow to model hierarchies of concurrently executed components. A parallel component interconnects atomic or other parallel components with its input and output ports. Thus, a parallel component is a 8-tuple

$$\text{DEVS}_{\text{parallel}} = (P_{\text{in}}, P_{\text{out}}, X, Y, M, C_{\text{in}}, C_{\text{out}}, C_{\text{inner}}).$$

Parallel components have a finite set of input ports $P_{in}$ and output ports $P_{out}$. Further, each port has its own event value set defined in $X_{p \in P_{in}} \in X$ (input port event value sets) and $Y_{p \in P_{out}} \in Y$ (output port event value sets).

The inner components $M$ of a parallel component may be either other parallel components or atomic components. Additionally, inner components may be coupled with the input and output ports of each other or with the parallel component itself.

The sets $C_{in}$, $C_{out}$, and $C_{inner}$ specify port couplings of input ports and inner components ($C_{in}$), output ports and inner components ($C_{out}$), and in between inner components ($C_{inner}$). A port coupling is a pair of component and port tuples, i.e.

$$((c_{source}, p_{source}), (c_{destination}, p_{destination})).$$

The source and destination components $c_{source}$ and $c_{destination}$ are either inner components or the parallel component itself, i.e.

$$c_{source}, c_{destination} \in M \cup \left\{ DEVS_{parallel} \right\}.$$

The source port $p_{source}$ has to be an input or output port of the component $c_{source}$, i.e.

$$p_{source} \in \left\{ \begin{array}{ll} P_{out} \text{ of } c_{source} & \text{if } c_{source} \in M \\ P_{in} \text{ of } c_{source} & \text{if } c_{source} = DEVS_{parallel}. \end{array} \right.$$

Similarly, the destination port $p_{destination}$ has to be an output or input port of the component $c_{destination}$, i.e.

$$p_{destination} \in \left\{ \begin{array}{ll} P_{in} \text{ of } c_{destination} & \text{if } c_{destination} \in M \\ P_{out} \text{ of } c_{destination} & \text{if } c_{destination} = DEVS_{parallel}. \end{array} \right.$$

Thus, only port couplings between an event source and drain are allowed. An event source (drain) is either an output (input) port of an inner component or the input (output) port of the parallel component itself. Except this simple but generic rule of data-flow architectures, i.e., data-flow is represented as a data-dependent digraph [DFL74], port couplings are almost unconstrained. However, port couplings in terms of direct feedback loops are not allowed as well. Thus, output and input ports of the same component (i.e. $c_{source} = c_{destination}$) may not be connected together.
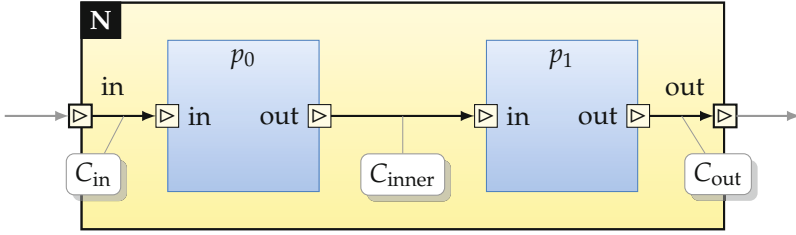
**Figure 2.1** Parallel component example with two inner atomic components

### Example

Figure 2.1 depicts an example of two coupled atomic components within a parallel component. The parallel component $N$ is

$$N = (P_{in}, P_{out}, X, Y, M, C_{in}, C_{out}, C_{inner})$$

where

$$
\begin{aligned}
P_{in} &= \{\text{"in"}\} \\
P_{out} &= \{\text{"out"}\} \\
P_{in,in} &= V \text{ (an arbitary set of values)} \\
P_{out,out} &= V \\
M &= \{(p_0, A), (p_1, A)\} \text{ ($A$ is another component definition)} \\
C_{in} &= \{((N, \text{"in"}), (p_0, \text{"in"}))\} \\
C_{out} &= \{((p_1, \text{"out"}), (N, \text{"out"}))\} \\
C_{inner} &= \{((p_0, \text{"out"}), (p_1, \text{"in"}))\}.
\end{aligned}
$$

### 2.1.2  Atomic Components

An atomic component implements the models behaviour in terms of a finite state machine with timed state transitions. It is denoted by an 11-tuple

$$\text{DEVS}_{atomic} = (P_{in}, P_{out}, X, Y, S, s_0, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \tau).$$

Atomic components have a finite set of input ports $P_{in}$ to receive events and output ports $P_{out}$ to emit events. Likewise the parallel component ports,

**Figure 2.2** Relation of internal state transition and output function execution in DEVS atomic components
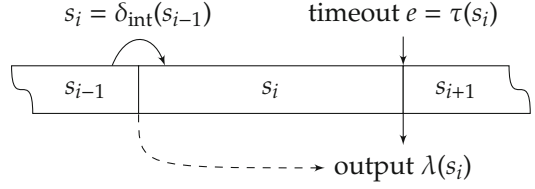
$$s_i = \delta_{\text{int}}(s_{i-1}) \qquad \text{timeout } e = \tau(s_i)$$

| $s_{i-1}$ | $s_i$ | $s_{i+1}$ |

$$\text{output } \lambda(s_i)$$

each port has its own event value set defined in $X_{p \in P_{\text{in}}} \in X$ (input port event value sets) and $Y_{p \in P_{\text{out}}} \in Y$ (output port event value sets).

$S$ is a non-empty set of states. The initial state of the atomic component is $s_0 \in S \cap \{\varnothing\}$. Every state $s \in S$ has an associated timeout value $\tau(s)$ with $\tau : S \rightarrow \mathbb{R}^+$ describing the maximum retention time of the state. Hence, timeout values can be assigned into three timing classes depicted by Table 2.1.

These different timing classes are of particular interest regarding the models state change and output behaviour. Whenever a model stayed in a state for the duration defined by $\tau(s)$ a timeout event occur and a state change is performed. Then, the internal transition function $\delta_{\text{int}} : S \rightarrow S$ is executed determining the successor state $s' = \delta_{\text{int}}(s)$ of the current state $s$ with $s, s' \in S$. Furthermore, the model may emit a single output event described by the function $\lambda : S \rightarrow Y^b$. Please note that $Y^b$ is a set of all possible output event values $Y_{p \in P_{\text{out}}}$ where multiple occurrences of the same symbol are allowed, i.e. $y \in Y^b$ is a bag $y$ containing all emitted output events. Figure 2.2 depicts the relation between the internal state transition and the output function execution within atomic components in DEVS. After entering the state $s_i$ its output value $\lambda(s_i)$ is determined. Afterwards, the calculated value is emitted when the state is left due to a timeout event. Please note that a model may emit multiple output events at the same point in time, due to the zero-timeout of states. All output events which occur in zero-timeout states are collected within a bag and, afterwards, they are simultaneously transmitted altogether.

A state change is also performed when an input event is received. Then,

**Table 2.1** Different timing classes of the state timeout function $\tau$

| Timing Class | Set of timing values $\subset \mathbb{R}^+$ |
|---|---|
| Zero-timeout | $\{0\}$ |
| Real-timeout | $\mathbb{R}^+ \setminus \{0, \infty\}$ |
| Non-timeout | $\{\infty\}$ |

the external transition function $\delta_{\text{ext}} : S \times X \times \mathbb{R}^+ \to S$ is executed. Thus, whatever event (*timeout* or *external*) occurs first, the corresponding transition function is executed and the state change is performed within the atomic component. However, for the special case that at the same point in time, both, input and timeout events occur, it is not decidable whether the internal or external transition function should be executed. In such a case the target state cannot be properly computed.

Zeigler et al. introduced in [ZKP00] an additional transition function to circumvent this DEVS MoC design flaw: The confluent transition function $\delta_{\text{con}} : S \times X \times \mathbb{R}^+ \to S$. This function is executed whenever this special case occur and, thus, making the model's state change behaviour decidable. Taken together, the successor state $s'$ is determined by

$$s' = \begin{cases} \delta_{\text{int}}(s) & \text{if } e = \tau(s) \\ \delta_{\text{ext}}(s, x, e) & \text{if at least a single input event occured and } e \neq \tau(s) \\ \delta_{\text{con}}(s, x, e) & \text{otherwise.} \end{cases}$$

In addition, $\delta_{\text{con}}$ can be classified by three fundamentally different kinds of confluent transition functions:

$$\delta_{\text{con}}(s, x, e) = \begin{cases} \delta_{\text{int}}(s) \text{ or } \delta_{\text{ext}}(s, x, e) & \text{(Type 1)} \\ \delta_{\text{int}}(\delta_{\text{ext}}(s, x, e)) \text{ or } \delta_{\text{ext}}(\delta_{\text{int}}(s), x, e) & \text{(Type 2)} \\ f(s, x, e). & \text{(Type 3)} \end{cases}$$

Type 1 confluent transition functions globally decide which event (i.e. internal or external event) is irrelevant and should not be processed at all. Type 2 functions globally decide which event has a higher priority and should be processed in favour. Type 3 functions are the common case where the successor state can be freely defined without reusing the existing $\delta_{\text{int}}$ or $\delta_{\text{ext}}$ functions of the model. Taken together, Type 1 is embedded within Type 2 and Type 3. Likewise, Type 2 is embedded in Type 3. Thus, it is Type 1 $\subset$ Type 2 $\subset$ Type 3. Please note that each $\delta_{\text{con}}$ type has an increasing design complexity but allows higher expressiveness of the model in comparison to the preceding subtype.

## Example

A simple atomic component is shown below. The model receives an event with value $n$ from its input port and calculate within 5 ns an output event

**Figure 2.3** Trajectories for the example atomic DEVS model $M$. Input and output events are shown in relation to the model state.
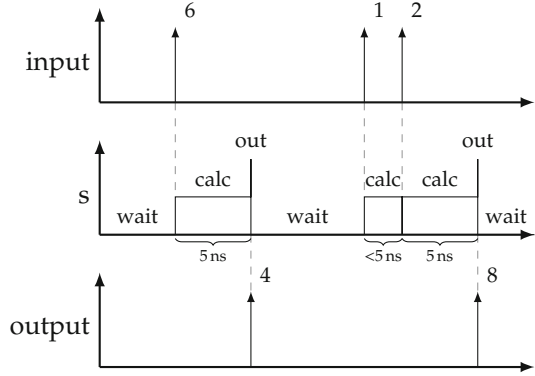
with value $10 - n$. The atomic component $M$ is

$$M = (P_{\text{in}}, P_{\text{out}}, X, Y, S, s_0, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, \tau)$$

where

$$
\begin{aligned}
P_{\text{in}} &= \{\text{"input"}\} \\
P_{\text{out}} &= \{\text{"output"}\} \\
P_{\text{in,input}} &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \\
P_{\text{out,output}} &= P_{\text{in,input}} \\
S &= \{\text{wait, calc, out}\} \times P_{\text{in,input}} \\
s_0 &= (\text{wait}, 0) \\
\delta_{\text{int}}\left((\tilde{s}, n)\right) &= \begin{cases} (\text{out}, n) & \text{if } \tilde{s} = \text{calc} \\ (\text{wait}, n) & \text{otherwise} \end{cases} \\
\delta_{\text{ext}}\left((\tilde{s}, n), x, e\right) &= \begin{cases} (\text{calc}, x) & \text{if } \tilde{s} = \text{wait} \\ (\tilde{s}, x) & \text{otherwise} \end{cases} \\
\delta_{\text{con}}(s, x, e) &= \delta_{\text{ext}}\left(\delta_{\text{int}}(s), x, e\right) \\
\lambda\left((\text{out}, n)\right) &= 10 - n \\
\tau\left((\tilde{s}, n)\right) &= \begin{cases} 5\,\text{ns} & \text{if } \tilde{s} = \text{calc} \\ 0\,\text{ns} & \text{if } \tilde{s} = \text{out} \\ \infty & \text{otherwise.} \end{cases}
\end{aligned}
$$

Figure 2.3 depicts the trajectories of the model for some input events. 5 ns after the first input event with the value 6 is received, an output event

with value 4 is emitted. Please note that the given model $M$ discards input events if another input event is received within the output value calculation duration, i.e. state $s =$ calc. This case is demonstrated by the second and third input events with values 1 and 2 in Figure 2.3. This behaviour is due to the encoding of the $\delta_{\text{int}}$ function. The receipt of the third input event changes the state from ("*calc*", 1) to $\delta_{\text{int}}(("calc"", 1), 2, e) = ("calc"", 2)$ for $0\,\text{ns} < e < 5\,\text{ns}$.

## 2.2 Synthesizable DEVS

The original DEVS formalism has some downsides in regard to automatic hardware or software synthesis of models. These downsides are event bags for events, port couplings with multiple sources, inexplicit specified output behaviour of confluent transitions, and disallow of port back couplings. Therefore, the SynDEVS formalism is introduced defeating these weaknesses of the original formalism. Its main goal is to preserve as much as possible of the original DEVS expressiveness without loosing the capability of synthesis. Before representing a formal definition of SynDEVS the downsides of original DEVS are discussed in detail in the next subsections. Then, SynDEVS is introduced including some additional features, i.e. component variables, port/variable/conditional expressions, and a graphical representation of atomic components.

### 2.2.1 Output Port Event Bags

The introduction of event bags within original DEVS is one of the main issues which impede the synthesis of models into hardware or software. It is possible at atomic components to emit an arbitrary (possibly infinite) amount of events over a single output port within a single point in time. Therefore, all emitted events for the same source (i.e. an output port) are collected in a bag, i.e. a multiset[1]. Representing such a bag in hard- or software would require a possibly infinite amount of memory resources to store all emitted events. Occasionally, this behaviour is not synthesizable. The following atomic component $M_\infty$ depicts such a computable but non-synthesizable DEVS model which outputs a recursively enumerable set of

---

[1] In mathematics, a multiset (or bag) is a generalization of a set where multiple instances of the same member may appear in the set. Other proposed names for multisets are listed in [Knu98, pp. 694].

**Figure 2.4** Trajectories
for the example atomic
DEVS model $M_\infty$. Output
events are shown in re-
lation to the model state.
The output behaviour is
computable but not syn-
thesizable as it emits an
unbound count of events.



events.

$$M_\infty = (P_{in}, P_{out}, X, Y, S, s_0, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \tau)$$

where

$$
\begin{aligned}
P_{in} &= \varnothing \\
P_{out} &= \{\text{"output"}\} \\
X &= \varnothing \\
P_{out,output} &= 1 \\
S &= \{\text{wait}, \text{out}\} \times \mathbb{N} \times \mathbb{N} \\
s_0 &= (\text{wait}, 1, 1) \\
\delta_{int}\left((\tilde{s}, n, m)\right) &= 
\begin{cases}
(\text{out}, n, 1) & \text{if } \tilde{s} = \text{wait} \\
(\text{out}, n, m+1) & \text{if } \tilde{s} = \text{out} \wedge m < n \\
(\text{wait}, n+1, 1) & \text{otherwise}
\end{cases} \\
\delta_{ext}\left(s, x, e\right) &= s_0 \\
\delta_{con}(s, x, e) &= s_0 \\
\lambda\left((\text{out}, n, m)\right) &= 1 \\
\tau\left((\tilde{s}, n, m)\right) &= 
\begin{cases}
1\,\text{ns} & \text{if } \tilde{s} = \text{wait} \\
0\,\text{ns} & \text{otherwise.}
\end{cases}
\end{aligned}
$$

Figure 2.4 depicts the output event count in relation to the state of the
model. $M_\infty$ toggles between the "wait" and "out" state. The "out" state
outputs within a single point in time up to $n \in \mathbb{N}$ events. The "wait"
state advances the time by 1 ns and increments the upper bound $n$ by one
each time. Taken together, the model $M_\infty$ has a computable (i.e. $n$ events
after $n$ ns) but non-synthesizable output behaviour. On the other hand,

DEVS models with infinite bag size (i.e. without an upper bound) are certainly impractical themselves. One could not even simulate the model behaviour: The simulation would not terminate as infinite events would have to be generated. Thus, it is sufficient to focus on DEVS models with a bounded output behaviour, i.e. models where an upper bound for the emitted event count within a single point in time exists. Such models can be easily transformed into equivalent DEVS models which only emit a single event over each output port within a single point in time. Before this model transformation is discussed in detail some definitions and theorems must be introduced to understand the model transformation.

**Definition 1 ($\delta_{int}$ reachable).** Let $\{s_0, \ldots, s_n\} = \tilde{S} \subseteq S$ be a subset of an atomic component's states $S$; then the state $s_n$ is called $\delta_{int}$ reachable in $\tilde{S}$ from $s_0$ if, and only if, a finite $\delta_{int}$ transition sequence exists with

$$
\begin{aligned}
s_1 &= \delta_{int}(s_0) \\
s_2 &= \delta_{int}(s_1) = \delta_{int}\left(\delta_{int}(s_0)\right) \\
&\vdots \\
s_n &= \delta_{int}(s_{n-1}) = \underbrace{\delta_{int}(\delta_{int}(\ldots \delta_{int}(s_0)\ldots))}_{n \text{ times}}.
\end{aligned}
$$

**Definition 2 ($\delta_{con}$ reachable).** Let $\{s_0, \ldots, s_n\} = \tilde{S} \subseteq S$ be a subset of an atomic component's states $S$; then the state $s_n$ is called $\delta_{con}$ reachable in $\tilde{S}$ from $s_0$ if, and only if, a finite $\delta_{con}$ transition sequence exists with

$$
\begin{aligned}
s_1 &= \delta_{con}(s_0) \\
s_2 &= \delta_{con}(s_1) = \delta_{con}\left(\delta_{con}(s_0)\right) \\
&\vdots \\
s_n &= \delta_{con}(s_{n-1}) = \underbrace{\delta_{con}(\delta_{con}(\ldots \delta_{con}(s_0)\ldots))}_{n \text{ times}}.
\end{aligned}
$$

**Definition 3 (Cycle free).** Let $\{s_0, \ldots, s_n\} = \tilde{S} \subseteq S$ be a subset of an atomic component's states $S$ where $s_n$ is $\delta_{con}$ or $\delta_{int}$ reachable in $\tilde{S}$ from $s_0$; then this transition sequence is called *cycle free* if, and only if, each state $s_i \in \tilde{S}$ is not $\delta_{int}$ (or $\delta_{con}$) reachable in $S$ from itself.

**Definition 4 (Zero-timeout chain).** Let $\{s_0, \ldots, s_n\} = \tilde{S}_{zerotimeout} \subseteq S$ be a subset of an atomic component's states. Then, these states are called a

*zero-timeout chain* if, and only if, the timeout for every state is zero (i.e. $\forall s_i \in \tilde{S}_{\text{zerotimeout}}.\tau(s_i) = 0$) and $s_n$ is $\delta_{\text{int}}$ (or $\delta_{\text{con}}$) reachable in $\tilde{S}_{\text{zerotimeout}}$ from $s_0$.

**Theorem 1.** *Within an atomic component the longest cycle free zero-timeout chain with length n determines the upper bound n of possible output events which may occur within a single point in time at each output port.*

*Proof.* Within original DEVS an output event may be emitted in two cases: (I) a timeout event occurred alone or (II), both, a timeout event and an external event occurred together. In both cases a state change is initiated and the successor state $s'$ is determined either by the $\delta_{\text{int}}$ function, i.e. case (I), or the $\delta_{\text{con}}$ function, i.e. case (II). However, if we want to emit more then a single event within a single point in time we have to subsequently generate timeout events by setting the timeout of the state $s'$ to $\tau(s') = 0$. Thus, the proof must be split up into two parts: Firstly, it must be proven that a zero-timeout chain with length $n$ can emit $n$ output events and secondly, it must be proven that this zero-timeout chain can emit $n$ output events at the utmost.

Without loss of generality we may assume that there exists a longest cycle free zero-timeout chain of states $\{s_0, \ldots, s_n\} = \tilde{S}_{\text{zerotimeout}} \subseteq S$ with length $n$. Thus, each state $s_i \in \tilde{S}_{\text{zerotimeout}}$ within this chain may emit an output event *once* at each of the output ports. Taken together, after the atomic component entered this zero-timeout chain it may emit up to $n$ events within a single point in time at each of the output ports.

Assume to the contrary that there may exist another cycle free zero-timeout chain which emit $n' > n$ events. Then, there must be another $\delta_{\text{int}}$ (or $\delta_{\text{con}}$) transition within $\tilde{S}_{\text{zerotimeout}}$ which emit at least one additional output event. Such an additional transition could be a $\delta_{\text{int}}$ (or $\delta_{\text{con}}$) transition (I) within two states from $\tilde{S}_{\text{zerotimeout}}$ or (II) from a state of $\tilde{S}_{\text{zerotimeout}}$ to a state in $S$. However, the first case conflicts with the cycle free property of the zero-timeout chain, contradicting our assumption. Case (II) would make this new cycle free zero-timeout chain longer than the original one, again contradicting our assumption.

Taken both parts together, a zero-timeout chain of length $n$ may emit up to $n$ output events and there may not exist another zero-timeout chain with the same length emitting more then $n$ events. Thus, within an atomic component the longest cycle free zero-timeout chain with length $n$ determines the upper bound $n$ of possible output events which may occur within a single output port. $\square$

SynDEVS Co-Design Flow
A Hardware / Software Co-Design Flow Based on the
Discrete Event System Specification Model of Computation
Molter, H.G.
2012, XXVI, 198 p. 57 illus., 45 illus. in color., Softcover