# A Formal Semantics of UML-RT

Michael von der Beeck

BMW Group
Michael.Beeck@bmw.de

**Abstract.** The modeling language UML-RT, a dialect of the UML, supports the development of complex, hierarchical systems following a component-oriented approach. However, for a solid foundation of model analysis and model transformations a formal semantics definition of UML-RT is missing. Therefore, this paper presents a precise syntax and semantics definition of a sublanguage of UML-RT. This sublanguage puts an emphasis on the specification of complex, hierarchical state-based models. It considers atomic capsules - containing a statechart - and complex capsules that recursively consist of capsules communicating asynchronously with each other over connectors. Labeled transition systems are chosen as semantic domain, such that the UML-RT semantics can be defined in an SOS style a la Plotkin.

## 1 Introduction

Model-based software development using standard modeling languages represents a modern approach putting emphasis on the early development phases. Typical representatives are UML - constituting the de-facto modeling standard for industrial object-oriented applications - and UML-RT [13] - a dialect of UML especially designed for the development of distributed, embedded systems.

A great advantage of these modeling notations is given by their great variety of intuitive and mostly well-known graphical notations which support quite different kinds of information to be modeled: e.g. requirements, static structure, as well as interactive and dynamic behaviour. However, both languages - UML as well as UML-RT - suffer from insufficient semantics definitions lacking preciseness and completeness. The potential consequences are manyfold: Different persons might interprete the same model in different ways. Furthermore, the foundation for systematic, precise model analysis (e.g. consistency checks) and for model simulation or code generation is missing.

Some work aiming at a precise UML semantics definition has already been done or has at least been started. Due to its very comprehensive syntax, this is a long lasting, tedious task.

However, in this paper we consider UML-RT. More precisely, we deal with the syntax and semantics definition of a sublanguage of UML-RT. In this setting we focus on behavioural aspects modeled with UML-RT capsules: there are atomic capsules which reside on a statechart as well as complex capsules which can also contain a statechart, but which furthermore recursively contain a set of capsules communicating asynchronously with each other and with the surrounding capsule via connectors.

We choose labeled transition systems as semantic domain for UML-RT - the reason being twofold: on the one hand they are very appropriate for an operational semantics definition of (behavioural) modeling languages like UML-RT, on the other hand many equivalence and refinement notions well-known from the process algebra area [9,5] are defined with respect to labeled transition systems. Such notions are very appropriate means for precise systematic model analysis. They can e.g. be used to define consistency notions for UML-RT models.

The rest of the paper is structured as follows: In section 2 we precisely define the syntax of UML-RT models, whereas in section 3 we precisely define their semantics. Section 4 discusses related work. We conclude and discuss future work in section 5.

## 2 Syntax of UML-RT Models

We define the syntax of UML-RT models in two steps. At first we define the syntax of UML-RT Statecharts. Then we define the syntax of UML-RT capsules using the - already existing - syntax definition of UML-RT Statecharts. Note that we use the terms *UML-RT capsules* and *UML-RT models* as synonyms.

### 2.1 UML-RT Statechart Terms

UML-RT Statecharts is a visual language. However, for our aim to define a formal semantics, it is convenient to represent UML-RT Statecharts not visually but by textual terms. This is also done in related work for “classical” Statecharts [8,15] as well as for UML Statecharts [6,17].

Let $\mathcal{N}, \mathcal{T}, \Pi$ be countable sets of state names, transition names, and events, respectively. We denote events and actions by $a, b, c, \ldots$. For a set $M$ let $M^*$ denote the set of finite sequences over $M$. Then, the set $\mathsf{UML\text{-}SC}$ of *UML-RT Statechart terms* is inductively defined to be the least set satisfying the following conditions, where $n \in \mathcal{N}$.

1. **Basic term:** $s = [n]$ is a UML-RT Statechart term with $\mathsf{type}(s) = \mathsf{basic}$. Therefore $s$ is also called a *basic term.*
2. **Or-term:** If $s_1, \ldots, s_k$ are UML-RT Statechart terms for $k > 0$, $\rho = \{1, \ldots, k\}$, $l \in \rho$, $\mathsf{HT} = \{\mathsf{none}, \mathsf{deep}\}$, and $T \subseteq \mathsf{TR} =_{\text{df}} \mathcal{T} \times \rho \times \Pi \times (\Pi \cup \{\epsilon\}) \times \rho \times \mathsf{HT}$ with $\epsilon \notin \Pi$, then $s = [n, (s_1, \ldots, s_k), l, T]$ is a UML-RT Statechart term with $\mathsf{type}(s) = \mathsf{or}$. Therefore, $s$ is also called an *Or-term*. Here, $s_1, \ldots, s_k$ are the *subterms* of $s$, $T$ is the set of *transitions*[1] between the subterms of $s$, $s_1$ is the *default subterm* of $s$, $l$ is called the *active state index* of $s$ (or for short: the *index* of $s$), and $s_l$ is the *currently active* subterm of $s$ (or for short: $s_l$ is *active*). $\epsilon$ is called the *empty output*.
   Note that active state index $l \in \{1, \ldots, k\}$ denotes the $l$-th term within the $k$-tuple $(s_1, \ldots, s_k)$ of the subterms of $s$. Analogously, note that components two and five of a transition $t = (\underline{t}, i, e, a, j, ht) \in T$ - namely $i$ and $j$ - of an Or-term $s = [n, (s_1, \ldots, s_k), l, T]$ refer to the $i$-th and $j$-th term of the $k$-tuple $(s_1, \ldots, s_k)$, respectively, but not to the indexes of the states' names in the $k$-tuple.
   For each transition $t = (\underline{t}, i, e, a, j, ht) \in T$, we define $\mathsf{name}(t) =_{\text{df}} \underline{t}$, $\mathsf{sou}(t) =_{\text{df}} s_i$, $\mathsf{ev}(t) =_{\text{df}} e$, $\mathsf{act}(t) =_{\text{df}} a$, $\mathsf{tar}(t) =_{\text{df}} s_j$, and $\mathsf{historyType}(t) =_{\text{df}} ht$. $\mathsf{name}(t)$

[1] Later on, we will classify this kind of transitions as *syntactic* transitions.

is called the *transition name* of $t$, $\mathsf{ev}(t)$ and $\mathsf{act}(t)$ are called the *trigger part* and *action part* of $t$, respectively. $\mathsf{sou}(t)$ and $\mathsf{tar}(t)$ are called the *source* and *target* of $t$, respectively. Furthermore, $\mathsf{historyType}(t)$ is called the *history type* of $t$. Finally, $(e, a)$ is called the *label*[2] of $t$ and is graphically represented as $e/a$ or as $\underline{t} : e/a$.

In both cases (Basic term and Or-term) we refer to $n$ as the *root name* of $s$ and write $\mathsf{root}(s) =_{\text{df}} n$. We assume that all root names and transition names are mutually disjoint, so that terms and transitions within UML-RT Statechart terms are uniquely referred to by their names. For convenience, we sometimes write "state" instead of "term" and abbreviate $(s_1, \ldots, s_k)$ by $(s_{1..k})$.

As can be seen from our UML-RT Statechart term syntax we do not consider the following features of UML-RT Statecharts: entry and exit actions, interlevel transitions, and pseudostates. However, entry and exit actions as well as interlevel transitions had been included in our previous work [17], where we already defined a UML-RT Statechart semantics. Due to lack of space we do not consider these features in this work, where the UML-RT Statechart syntax only constitutes a part of the overall UML-RT capsule syntax.

We exemplify our textual syntax of UML-RT Statecharts graphically by Fig. 1 showing a complete UML-RT capsule which contains a UML-RT Statechart term $S$ shown as a rectangle with rounded corners and with (root) name $n_S$ in the upper part of the figure:

$S = [n_S, (S5, S1), l, \{t_1, t_2\}]$ is a UML-RT Statechart term with $\mathsf{type}(S) = \mathsf{or}$, i.e. $S$ is an Or-term, where

- $n_S$ is the root name of $S$,
- $\{S1, S5\}$ is the set of subterms of $S$, where
  - $S1$ is an Or-term with $S1 = [n_{S1}, (S4, S2, S3), l', \{t_3, t_4, t_5\}]$,
  - $n_{S1}$ is the root name of $S1$,
  - $S5$ is a basic term,
  - $n_{S5}$ is the root name of $S5$,
- $S5$ is the default subterm of $S$,
- $l \in \{1, 2\}$ is the active state index of $S$, (but not shown in Fig. 1)
- $\{t_1, t_2\}$ is the set of transitions between the subterms of $S$ with $t_1 = (\underline{t}_1, 1, e_1, a_1, 2, \mathit{none})$ and $t_2 = (\underline{t}_2, 2, e_2, a_2, 1, \mathit{none})$.

## 2.2 UML-RT Capsules

Let $\mathcal{N}_{ca}, \mathcal{N}_{po}, \mathcal{N}_{co}$ be countable sets of capsule names, port names, and connector names, respectively. Furthermore, let $\mathsf{Prot}$, the set of *protocols* over $\Pi$, be defined as $\mathsf{Prot} =_{\text{df}} \{pr \,|\, pr \subseteq \Pi \times \Pi\}$ and $\mathsf{CO}$, the set of *connectors* over $\Pi$, be defined as $\mathsf{CO} =_{\text{df}} \mathcal{N}_{co} \times \mathcal{N}_{po} \times \mathcal{N}_{po} \times \Pi^*$. Then the set $\mathsf{CAP}$ of *UML-RT capsules* is inductively defined to be the least set satisfying the following conditions:

1. **Basic UML-RT Capsule:**
   If $n \in \mathcal{N}_{ca}, po_1, \ldots, po_l \in \mathcal{N}_{po}$ for $l \geq 0$, $pr_{i_1}, \ldots, pr_{i_l} \in \mathsf{Prot}$, $S \in \mathsf{UML\text{-}SC}$ and $\sigma \in \Pi^*$, then

[2] Later on, we will classify this type of labels as *syntactic* labels.

$$ca = [n, (po_1, \ldots, po_l), (pr_{i_1}, \ldots, pr_{i_l}), S, \sigma]$$

is a UML-RT capsule. More specifically, $ca$ is also called a *basic UML-RT capsule*.

2. **Complex Non-behavioral UML-RT Capsule:**
If $n \in \mathcal{N}_{ca}, po_1, \ldots, po_l \in \mathcal{N}_{po}$ for $l \geq 0$, $pr_{i_1}, \ldots, pr_{i_l} \in \mathsf{Prot}$, $ca_1, \ldots, ca_k$ are UML-RT capsules for $k > 0$, and $co_i \in \mathcal{N}_{co} \times (\{po_1, \ldots, po_l\} \cup \bigcup_{j=1}^{k} \mathsf{Ports}(ca_j))^2 \times \Pi^* \subseteq \mathsf{CO}$ for $1 \leq i \leq m$ for $m \geq 0$, then

$$ca = [n, (po_1, \ldots, po_l), (pr_{i_1}, \ldots, pr_{i_l}), (ca_1, \ldots, ca_k), (co_1, \ldots, co_m)]$$

is a UML-RT capsule. More specifically, $ca$ is also called a *complex non-behavioural UML-RT capsule*.

3. **Complex Behavioral UML-RT Capsule:**
If $n \in \mathcal{N}_{ca}, po_1, \ldots, po_l \in \mathcal{N}_{po}$ for $l \geq 0$, $pr_{i_1}, \ldots, pr_{i_l} \in \mathsf{Prot}$, $S \in \mathsf{UML\text{-}SC}$, $\sigma \in \Pi^*$, $ca_1, \ldots, ca_k$ are UML-RT capsules for $k > 0$, and $co_i \in \mathcal{N}_{co} \times (\{po_1, \ldots, po_l\} \cup \bigcup_{j=1}^{k} \mathsf{Ports}(ca_j))^2 \times \Pi^*$ for $1 \leq i \leq m$ for $m \geq 0$, then

$$ca = [n, (po_1, \ldots, po_l), (pr_{i_1}, \ldots, pr_{i_l}), S, \sigma, (ca_1, \ldots, ca_k), (co_1, \ldots, co_m)]$$

is a UML-RT capsule. More specifically, $ca$ is also called a *complex behavioural UML-RT capsule*.

For the three abovementioned cases the following notions are used:
$n$ is called the *name* of $ca$, $\mathsf{Ports}(ca) =_{\text{df}} \{po_1, \ldots, po_l\}$ is called the *set of ports* of $ca$, $\mathsf{Prot}(po_j) =_{\text{df}} pr_{i_j}$ for $1 \leq j \leq l$ is called the *protocol* of $po_j$, $S$ is called the *Statechart* of $ca$, $\sigma$ is called the *input queue* of $ca$, and $\mathsf{Conn}(ca) =_{\text{df}} \{co_1, \ldots, co_m\}$ is called the *set of (UML-RT) connectors* of $ca$.

Informally, a basic UML-RT capsule does not contain any capsules. In contrast, both types of complex UML-RT capsules recursively contain capsules. Furthermore, a complex non-behavioral UML-RT capsule does not contain a Statechart on its the top level, whereas a complex behavioral UML-RT capsule contains a Statechart on its top level.

The UML-RT capsule syntax defined above does not support the following features: conjugate ports, event priorities, do activities, and dynamic capsules.

In the subsequent sections we need the following definitions. Let $\mathsf{Caps}(po_i) =_{\text{df}} ca$ for $1 \leq i \leq l$ and $\mathsf{SubCaps}(ca) =_{\text{df}} \{ca_1, \ldots, ca_k\}$, where $ca_1, \ldots, ca_k$ are called *subcapsules* of $ca$ and $ca$ is called *parent capsule* of $ca_i$ for $1 \leq i \leq k$ . Furthermore, we use projection functions $\Pi_j$ which are defined by $\Pi_j([x_1, \ldots, x_m]) =_{\text{df}} x_j$ for $1 \leq j \leq m$ for $m \geq 2$. Then, function $\mathsf{type} : \mathcal{N}_{po} \longrightarrow \{\mathsf{relay}, \mathsf{end}\}$ is defined as follows:

$$\mathsf{type}(po) =_{\text{df}} \begin{cases} \mathsf{relay}, \text{ if } \exists ca \in \mathsf{CAP}, co \in \mathsf{CO} \,.\, po \in \mathsf{Ports}(ca) \wedge co \in \mathsf{Conn}(ca) \\ \qquad \wedge (po = \Pi_2(co) \vee po = \Pi_3(co)) \\ \mathsf{end}, \ \text{otherwise} \end{cases}$$

Finally, we exemplify our textual syntax of UML-RT capsules graphically by Fig. 1: $ca = [n_{ca}, (po_1, po_2, po_3), (pr_{i_1}, pr_{i_2}, pr_{i_3}), S, \sigma, (ca_1, ca_2), (co_1, co_2, co_3, co_4)]$ is a complex behavioural UML-RT capsule, where

– $n_{ca}$ is the name of $ca$,
– $\{po_1, po_2, po_3\}$ is the set of ports of $ca$,
– $S$ is the Statechart of $ca$,[3]
– $\sigma$ is the input queue of $ca$,
– $ca_1$ and $ca_2$ are subcapsules of $ca$ (only shown as 'black boxes' with names $n_{ca1}$ and $n_{ca2}$, respectively, i.e. without any interior structure),
– and $\{co_1, \ldots, co_4\}$ is the set of connectors of $ca$.

The protocols $pr_{i_j}$ of $po_j$ for $1 \leq j \leq 3$ are not presented graphically. Furthermore, the ports of $ca_1$ and $ca_2$ are not named.

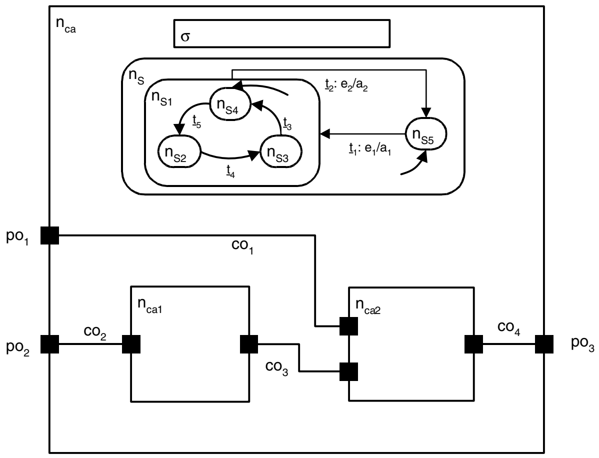

**Fig. 1.** UML-RT Model Example

## 3 Semantics of UML-RT Models

We follow the SOS (Structured Operational Semantics) approach of Plotkin [10]: we take labeled transition systems as semantic domain and use SOS rules to define the semantics of UML-RT models in an operational and modular approach, such that comprehension as well as flexibility (e.g. with respect to subsequent enhancements) are supported - without restricting preciseness.

In order to support a modular semantics definition we do not only follow Plotkin's SOS approach, but we also split up the overall semantics definition into three steps:

1. UML-RT Statechart semantics (section 3.1)
2. UML-RT connector semantics (section 3.2)
3. UML-RT capsule semantics = UML-RT model semantics (section 3.3)

In the first and in the second step the semantics of UML-RT Statecharts and UML-RT connectors are defined independently from each other. Then, in the third step, we use these semantics definitions to define the UML-RT capsule semantics.

[3] The structure of $S$ was already described at the end of Section 2.1.

## 3.1 UML-RT Statechart Semantics

The following formal semantics of UML-RT Statecharts is based on our earlier work [17].

To define the UML-RT Statechart semantics, we proceed as follows: In a first step we define how the state resulting from transition execution is computed. We use the solution in a second step to formally define the semantics of UML-RT Statecharts.

**Computing the Next State.** We define function $\mathsf{next}$ which computes the state which results from a transition execution. This function will be used in the SOS rule which handles transition execution (in an OR-state).

Given a UML-RT Statechart transition $t$ with history type $ht = \mathsf{historyType}(t)$ and target $s$, function $\mathsf{next} : \mathsf{HT} \times \mathsf{UML\text{-}SC} \longrightarrow \mathsf{UML\text{-}SC}$ computes the UML-RT Statechart term $s' = \mathsf{next}(ht, s)$ which results after execution of transition $t$. In order to simplify the presentation of $\mathsf{next}$ as well as the presentation of several subsequent definitions, we use the substitution notation $\cdot_{[\cdot/\cdot]}$ as follows: If $t$ is a term, then $t_{[a/b]}$ is the term which results from replacing all occurrences of $a$ in $t$ by $b$. Furthermore, for $l \in \{1, \dots, k\}$ we abbreviate $(s_1, \dots, s_{l-1}, s'_l, s_{l+1}, \dots, s_k)$ by $(s_{1..k})_{[s_l/s'_l]}$.

$$\mathsf{next}(ht, [n]) =_{\mathrm{df}} [n]$$

$$\mathsf{next}(ht, [n, (s_{1..k}), l, T]) =_{\mathrm{df}} \begin{cases} [n, (s_{1..k}), l, T] & \text{if } ht = \mathsf{deep} \\ [n, (s_{1..k})_{[s_1/\mathsf{default}(s_1)]}, 1, T] & \text{if } ht = \mathsf{none} \end{cases}$$

The definition of $\mathsf{next}$ uses function $\mathsf{default} : \mathsf{UML\text{-}SC} \longrightarrow \mathsf{UML\text{-}SC}$ which especially defines for an Or-state that its currently active substate is given by its default substate.

$$\mathsf{default}([n]) =_{\mathrm{df}} [n]$$

$$\mathsf{default}([n, (s_{1..k}), l, T]) =_{\mathrm{df}} [n, (s_{1..k})_{[s_1/\mathsf{default}(s_1)]}, 1, T]$$

**UML-RT Statechart Semantics Definition.** The UML-RT Statechart semantics will be defined for the textual UML-RT Statechart syntax as given by the set $\mathsf{UML\text{-}SC}$ of UML-RT Statechart terms.

We define the semantics by function $[\![.]\!] : \mathsf{UML\text{-}SC} \longrightarrow \mathsf{LTS}$, where $\mathsf{LTS}$ is the set of *labeled transition systems* and where the (semantic) transitions[4] work on single input events $e \in \Pi$. The semantics $[\![s]\!]$ of a UML-RT Statechart term $s \in \mathsf{UML\text{-}SC}$ is given by the labeled transition system $(\mathsf{UML\text{-}SC}, L, \longrightarrow, s) \in \mathsf{LTS}$, where

- $\mathsf{UML\text{-}SC}$ is the set of states,[5]
- $L = \Pi \times (\Pi \cup \{\epsilon\}) \times \{0, 1\}$ is the set of (semantic) labels[6].
- $\longrightarrow \subseteq \mathsf{UML\text{-}SC} \times L \times \mathsf{UML\text{-}SC}$ is the transition relation, and
- $s$ is the start state.

[4] We use the term "semantic transition" in order to distinguish transitions in the semantics of UML-RT Statecharts from the already defined (syntactic) transitions (cf. Section 2.1) in the syntax of UML-RT Statecharts, more precisely in UML-RT Statechart terms of type Or.

[5] This implies that each state of the transition system is given by a UML-RT Statechart term.

[6] Analogously to the distinction between syntactic and semantic transitions we also distinguish between syntactic and semantic labels. Syntactic labels have been defined in Section 2.1.

For the sake of simplicity, we write $s \xrightarrow[a]{e}_f s'$ instead of $(s,(e,a,f),s') \in \longrightarrow$ and $s \overset{e}{\not\to}_f$ instead of $\nexists s', a \,.\; s \xrightarrow[a]{e}_f s'$, where $s$ and $s'$ are called the *source* and the *target* of these (semantic) transitions, respectively, $e$ and $a$ are called the *input* and *output*, respectively, and $f$ is called the *flag*. We say that term $s$ may perform a (semantic) transition with input $e$, output $a$, and flag $f$ (or for short: with (semantic) label $(e,a,f)$) to term $s'$. If appropriate, we do no mention the input, output, and/or target of the transition. Intuitively, flag $f$ states whether a semantic transition is performed,

- either because at least one (syntactic) UML-RT Statechart transition is taken (in this case we have $f = 1$, denoted as *positive flag*)
- or without taking any (syntactic) UML-RT Statechart transition (in this case we have $f = 0$, denoted as *negative flag*). In this case only the input is "consumed", whereas source and target are identical. This is usually denoted as a *stuttering step*.

The flag is needed to assure that stuttering steps can only occur, if no non-stuttering step is possible. This assures a lower-first priority mechanism for transition execution in our UML-RT Statechart semantics.

Transition relation $\longrightarrow$ is defined by the SOS rules of Table 1 using rule format:

$$\textit{name}\ \frac{\textit{premise}}{\textit{conclusion}}$$

*Explanation of SOS rules of UML-RT Statechart semantics*

- BAS (stuttering)
  A basic state may perform a semantic transition with arbitrary input event $e$, empty output $\epsilon$, and negative flag such that the state does not change, i.e. that the input is just consumed.
- OR-1 (progress)
  If $t$ is a UML-Statechart transition of an Or-state $s$ with trigger part $e$, then $s$ can perform a semantic transition with input $e$ and positive flag if $s_l$ cannot perform a semantic transition with input $e$ and positive flag ($s_l \overset{e}{\not\to}_1$). The condition assures the lower-first priority of UML-RT Statecharts.
  The target of the semantic transition differs from its source by changing the currently active substate from $s_l$ to $s_i$, because $s_l$ and $s_i$ are the source and target of the UML-Statechart transition $t$, respectively. Furthermore, the dynamic information of $s_i$ is updated according to the history type $ht$ of $t$ using function next. This update is performed by the substitution $(s_{1..k})_{[s_i/\mathsf{next}(ht,s_i)]}$.
- OR-2 (propagation of progress)
  If a substate of an Or-state may perform a semantic transition with a label containing a positive flag, then the Or-state may perform a semantic transition with the same label.
- OR-3 (propagation of stuttering)
  If a substate of an Or-state may perform a semantic transition with a label containing a negative flag (i.e. no UML-RT Statechart transition can be taken within the Or-state) and if the Or-state cannot perform a semantic transition with positive flag, then the Or-state may also perform a semantic transition with the same label (in particular with negative flag).

The rules define that for every input event $e \in \Pi$ and for every state $s \in \mathsf{UML\text{-}SC}$

- either a semantic transition $s \xrightarrow[a]{e}_1 s'$ with output $a \in \Pi \cup \{\epsilon\}$ and state $s' \in \mathsf{UML\text{-}SC}$
- or a semantic transition $s \xrightarrow[\epsilon]{e}_0 s$ with empty output $\epsilon$ and without state change exists.

**Table 1.** SOS rules of the UML-RT Statechart semantics

$$\mathsf{BAS}\quad \frac{\mathsf{true}}{[n] \xrightarrow[\epsilon]{e}_0 [n]}$$

$$\mathsf{OR\text{-}1}\quad \frac{(\_, l, e, a, i, ht) \in T,\ s_l \not\xrightarrow{e}_1}{[n, (s_{1..k}), l, T] \xrightarrow[a]{e}_1 [n, (s_{1..k})_{[s_i/\mathsf{next}(ht, s_i)]}, i, T]}$$

$$\mathsf{OR\text{-}2}\quad \frac{s_l \xrightarrow[a]{e}_1 s_l'}{[n, (s_{1..k}), l, T] \xrightarrow[a]{e}_1 [n, (s_{1..k})_{[s_l/s_l']}, l, T]}$$

$$\mathsf{OR\text{-}3}\quad \frac{s_l \xrightarrow[\epsilon]{e}_0 s_l,\ [n, (s_{1..k}), l, T] \not\xrightarrow{e}_1}{[n, (s_{1..k}), l, T] \xrightarrow[\epsilon]{e}_0 [n, (s_{1..k}), l, T]}$$

## 3.2 UML-RT Connector Semantics

Connectors of UML-RT support the modeling of asynchronous communication between UML-RT capsules. In general, their behaviour is not precisely defined, but constitutes a semantic variation point e.g. to allow modeling of unreliable communication channels. However, we define UML-RT connectors as unbounded FIFO (First-In First-Out) queues.

The semantics $[\![co]\!]_c$ of a UML-RT connector $co \in \mathsf{CO}$ is given by the labeled transition system $(\mathsf{CO}, L', \rightarrow_c, co) \in \mathsf{LTS}$, where

- $\mathsf{CO}$ is the set of states,
- $L' = \{\tau\} \cup \{\mathsf{in}(sig)\,\mathsf{via}\,po \mid sig \in \Pi, po \in \mathcal{N}_{po}\}$
  $\cup\ \{\mathsf{out}(sig)\,\mathsf{via}\,po \mid sig \in \Pi, po \in \mathcal{N}_{po}\}$ is the set of labels (with $\tau \notin \Pi$),
- $\rightarrow_c\ \subseteq \mathsf{CO} \times L' \times \mathsf{CO}$ is the transition relation, and
- $co$ is the start state.

We distinguish whether a capsule or a connector uses a signal $sig$ as an input or as an output by writing $\mathsf{in}(sig)$ or $\mathsf{out}(sig)$, respectively. This distinction is necessary for the definition of synchronization between a capsule $ca_i$ and a connector $co_j$. This synchronization occurs as an internal communication of the parent capsule $ca$ of $ca_i$, where $co_j$ is contained in the set of ports of $ca$. Synchronization is hidden from the environment of $ca$, only an internal action $\tau$ can be observed outside $ca$. (See e.g. [9]).

We write $co \xrightarrow{l}_c co'$ instead of $(co, l, co') \in\ \rightarrow_c$ and say that connector $co$ may perform a transition with label $l$ to $co'$. Transition relation $\rightarrow_c$ is defined by SOS rules $\mathsf{co1}$ and $\mathsf{co2}$ shown in Table 2 using three relations $\hat{=}, >, le \subseteq \mathcal{N}_{po} \times \mathcal{N}_{po}$ defined by:

$$
\begin{aligned}
po \hat{=} po' \;&:\Longleftrightarrow \exists ca \in \mathsf{CAP} : \\
&\qquad (\mathsf{Caps}(po) \in \mathsf{SubCaps}(ca) \wedge \mathsf{Caps}(po') \in \mathsf{SubCaps}(ca)) \\
po > po' \;&:\Longleftrightarrow \mathsf{Caps}(po') \in \mathsf{SubCaps}(\mathsf{Caps}(po)) \\
po \leq po' \;&:\Longleftrightarrow po \hat{=} po' \vee po' > po
\end{aligned}
$$

Relations $>$ and $\leq$ are used in co1 and co2 to compare the hierarchy level of ports.

**Table 2.** SOS rules of UML-RT connector semantics

$$
\mathsf{co1}\ \frac{\mathsf{true}}{[n, po, po', \sigma] \xrightarrow{\mathsf{in}(sig)\mathsf{via}po}_c [n, po, po', \langle sig \rangle :: \sigma]} \quad \begin{pmatrix} (po > po' \wedge sig \in \mathsf{In}(\mathsf{Prot}(po))) \\ \vee \\ (po \leq po' \wedge sig \in \mathsf{Out}(\mathsf{Prot}(po))) \end{pmatrix}
$$

$$
\mathsf{co2}\ \frac{\mathsf{true}}{[n, po', po, \sigma :: \langle sig \rangle] \xrightarrow{\mathsf{out}(sig)\mathsf{via}po}_c [n, po', po, \sigma]} \quad \begin{pmatrix} (po \leq po' \wedge sig \in \mathsf{In}(\mathsf{Prot}(po))) \\ \vee \\ (po > po' \wedge sig \in \mathsf{Out}(\mathsf{Prot}(po))) \end{pmatrix}
$$

*Explanation of SOS rules of UML-RT connector semantics*

- co1 (input event for connector)
  Informally, a connector can read an input event from a port $po$, if $po$ is a port of this connector and if the event fulfils the protocol of the port.
- co2 (output event from connector)
  Informally, a connector can write an output event to a port $po$, it $po$ is a port of this connector and if the event fulfils the protocol of the port.

## 3.3 UML-RT Capsule Semantics

In the following we distinguish two cases to define the semantics of UML-RT capsules:

- In the 'general case' we use the UML-RT Statechart semantics of section 3.1 as well as the UML-RT connector semantics of section 3.2 to define the semantics of a UML-RT capsule generally, i.e. not restricted to one of the capsule's ports.
- In the 'port-specific case' we use the 'general case semantics' to define the semantics of a UML-RT capsule restricted to one of its ports.

**General Case.** The semantics $[\![ca]\!]'$ of a UML-RT capsule $ca \in \mathsf{CAP}$ is given by the labeled transition system $(\mathsf{CAP}, L', \twoheadrightarrow, ca) \in \mathsf{LTS}$, where

- CAP is the set of states,
- $L'$ is defined as before (in the semantics of UML-RT connectors),
- $\twoheadrightarrow \subseteq \mathsf{CAP} \times L' \times \mathsf{CAP}$ is the transition relation, and
- $ca$ is the start state.

Similar to the case of UML-RT Statechart semantics we write $ca \xrightarrow{l} ca'$ instead of $(ca, l, ca') \in \;\rightarrow$. We say that capsule $ca$ may perform a (semantic) transition with label $l$ to capsule $ca'$. For $l = \tau$ we say that $ca$ may perform a *silent* transition to $ca'$.

Transition relation $\rightarrow$ is defined by a set of SOS rules presented in Table 3 using the same rule format as in the case of UML-RT Statecharts as well as the rule format

$$\textit{name}\ \frac{\textit{premise}}{\begin{array}{c}\textit{conclusion 1}\\ \textit{conclusion 2}\end{array}}\ \textit{(condition)}$$

being an abbreviation for two rules with identical premises and identical conditions:

$$\textit{name}\ \frac{\textit{premise}}{\textit{conclusion 1}}\ \textit{(condition)} \qquad \text{and} \qquad \textit{name}\ \frac{\textit{premise}}{\textit{conclusion 2}}\ \textit{(condition)}$$

We abbreviate tuples $(x_1, \ldots, x_l)$ by $\bar{x}$ and we use functions $\mathsf{In}, \mathsf{Out} : \mathsf{Prot} \longrightarrow \Pi$ defined by $\mathsf{In}(pr) =_{\mathrm{df}} \Pi_1(pr)$ and $\mathsf{Out}(pr) =_{\mathrm{df}} \Pi_2(pr)$, respectively. In addition, function $\mathsf{Set}$ is defined by $\mathsf{Set}([x_1, \ldots, x_n]) =_{\mathrm{df}} \{x_1, \ldots, x_n\}$ transforming a tuple of elements to a set of these elements. The operator :: concatenates two lists to a single list. The list operator $\langle\rangle$ applied to an argument $sig$ produces a list which contains $sig$. For the case $sig = \epsilon$ we have $\langle sig \rangle = \langle \epsilon \rangle \overset{\mathrm{def}}{=} \langle\rangle$, i.e. the empty list.

Note that the premises of rules R2 and R3 use (semantic) transitions of the UML-RT Statecharts semantics, whereas the premises of rules R5-R8 use transitions of the UML-RT connector semantics.

*Explanation of SOS rules of UML-RT capsule semantics (general case)*

- R1 (storing an input event in input queue)
  A capsule can read event $\mathsf{in}(sig)$ at port $po$ and can store it as event $sig$ in its input queue.
- R2 (processing and storing input queue events )
  If Statechart term $S$ may perform a transition with input $sig$, output $sig'$, and flag $f$ to term $S'$, then a capsule with Statechart $S$ can read event $sig$ from its input queue, can produce event $sig'$, and can store event $sig'$ in its input queue.
- R3 (processing an input queue event and producing an output event)
  If Statechart term $S$ may perform a transition with input $sig$, output $sig'$, and flag $f$ to term $S'$, then a capsule with Statechart $S$ can read event $sig$ from its input queue and can produce event $\mathsf{out}(sig')$ which is offered at port $po$.
- R4 (propagation of internal communication)
  If a capsule $ca$ can perform a silent transition to capsule $ca'$, then a parent capsule of $ca$ can also perform a silent transition.
- R5 (communication from capsule to connector)
  If capsule $ca$ may perform a transition with label $\mathsf{out}(sig)\mathsf{via}po$ to $ca'$ and if connector $co$ may perform a transition with label $\mathsf{in}(sig)\mathsf{via}po$ to $co'$, then a parent capsule of $ca$ and $co$ may perform a silent transition to a parent capsule of $ca'$ and $co'$, if $po$ is a port of $ca$. Informally, capsule $ca$ offers event $sig$ at its port $po$ and connector $co$ reads event $sig$ at this port.
- R6 (communication from connector to capsule)
  If connector $co$ may perform a transition with label $\mathsf{out}(sig)\mathsf{via}po$ to $co'$ and if

**Table 3.** SOS rules of UML-RT capsule semantics (general case)

| | | |
|---|---|---|
| R1 | $\dfrac{\mathsf{true}}{\begin{array}{c}[n,\bar{po},\bar{pr},S,\sigma] \xrightarrow{\mathsf{in}(sig)\mathsf{via}po} [n,\bar{po},\bar{pr},S,\langle sig\rangle :: \sigma] \\ [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}] \xrightarrow{\mathsf{in}(sig)\mathsf{via}po} [n,\bar{po},\bar{pr},S,\langle sig\rangle :: \sigma,\bar{ca},\bar{co}]\end{array}}$ | $\left(\begin{array}{c}\exists j : [\Pi_j(\bar{po}) = po \\ \wedge \\ sig \in \mathsf{In}(\Pi_j(\bar{pr}))] \\ \wedge \\ \mathsf{type}(po) = \mathsf{end}\end{array}\right)$ |
| R2 | $\dfrac{S \xrightarrow[sig']{sig}_f S'}{\begin{array}{c}[n,\bar{po},\bar{pr},S,\sigma :: \langle sig\rangle] \xrightarrow{\tau} [n,\bar{po},\bar{pr},S',\langle sig'\rangle :: \sigma] \\ [n,\bar{po},\bar{pr},S,\sigma :: \langle sig\rangle,\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},S',\langle sig'\rangle :: \sigma,\bar{ca},\bar{co}]\end{array}}$ | |
| R3 | $\dfrac{S \xrightarrow[sig']{sig}_f S'}{\begin{array}{c}[n,\bar{po},\bar{pr},S,\sigma :: \langle sig\rangle] \xrightarrow{\mathsf{out}(sig')\mathsf{via}po} [n,\bar{po},\bar{pr},S',\sigma] \\ [n,\bar{po},\bar{pr},S,\sigma :: \langle sig\rangle,\bar{ca},\bar{co}] \xrightarrow{\mathsf{out}(sig')\mathsf{via}po} [n,\bar{po},\bar{pr},S',\sigma,\bar{ca},\bar{co}]\end{array}}$ | $\left(\begin{array}{c}\exists j : [\Pi_j(\bar{po}) = po \\ \wedge \\ sig' \in \mathsf{Out}(\Pi_j(\bar{pr}))] \\ \wedge \\ \mathsf{type}(po) = \mathsf{end}\end{array}\right)$ |
| R4 | $\dfrac{ca \xrightarrow{\tau} ca'}{\begin{array}{c}[n,\bar{po},\bar{pr},\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},\bar{ca}_{[ca/ca']},\bar{co}] \\ [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},S,\sigma,\bar{ca}_{[ca/ca']},\bar{co}]\end{array}}$ | $(ca \in \mathsf{Set}(\bar{ca}))$ |
| R5 | $\dfrac{ca \xrightarrow{\mathsf{out}(sig)\mathsf{via}po} ca' \quad co \xrightarrow{\mathsf{in}(sig)\mathsf{via}po}_c co'}{\begin{array}{c}[n,\bar{po},\bar{pr},\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},\bar{ca}_{[ca/ca']},\bar{co}_{[co/co']}] \\ [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},S,\sigma,\bar{ca}_{[ca/ca']},\bar{co}_{[co/co']}]\end{array}}$ | $\left(\begin{array}{c}ca \in \mathsf{Set}(\bar{ca}) \\ \wedge \\ co \in \mathsf{Set}(\bar{co}) \\ \wedge \\ po \in \mathsf{Ports}(ca)\end{array}\right)$ |
| R6 | $\dfrac{co \xrightarrow{\mathsf{out}(sig)\mathsf{via}po}_c co' \quad ca \xrightarrow{\mathsf{in}(sig)\mathsf{via}po} ca'}{\begin{array}{c}[n,\bar{po},\bar{pr},\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},\bar{ca}_{[ca/ca']},\bar{co}_{[co/co']}] \\ [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}] \xrightarrow{\tau} [n,\bar{po},\bar{pr},S,\sigma,\bar{ca}_{[ca/ca']},\bar{co}_{[co/co']}]\end{array}}$ | $\left(\begin{array}{c}ca \in \mathsf{Set}(\bar{ca}) \\ \wedge \\ co \in \mathsf{Set}(\bar{co}) \\ \wedge \\ po \in \mathsf{Ports}(ca)\end{array}\right)$ |
| R7 | $\dfrac{co \xrightarrow{\mathsf{in}(sig)\mathsf{via}po}_c co'}{\begin{array}{c}[n,\bar{po},\bar{pr},\bar{ca},\bar{co}] \xrightarrow{\mathsf{in}(sig)\mathsf{via}po} [n,\bar{po},\bar{pr},\bar{ca},\bar{co}_{[co/co']}] \\ [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}] \xrightarrow{\mathsf{in}(sig)\mathsf{via}po} [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}_{[co/co']}]\end{array}}$ | $\left(\begin{array}{c}co \in \mathsf{Set}(\bar{co}) \\ \wedge \\ po \in \mathsf{Set}(\bar{po})\end{array}\right)$ |
| R8 | $\dfrac{co \xrightarrow{\mathsf{out}(sig)\mathsf{via}po}_c co'}{\begin{array}{c}[n,\bar{po},\bar{pr},\bar{ca},\bar{co}] \xrightarrow{\mathsf{out}(sig)\mathsf{via}po} [n,\bar{po},\bar{pr},\bar{ca},\bar{co}_{[co/co']}] \\ [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}] \xrightarrow{\mathsf{out}(sig)\mathsf{via}po} [n,\bar{po},\bar{pr},S,\sigma,\bar{ca},\bar{co}_{[co/co']}]\end{array}}$ | $\left(\begin{array}{c}co \in \mathsf{Set}(\bar{co}) \\ \wedge \\ po \in \mathsf{Set}(\bar{po})\end{array}\right)$ |

capsule $ca$ may perform a transition with label $\mathsf{in}(sig)\mathsf{via}po$ to $ca'$, then a parent capsule of $ca$ and $co$ may perform a silent transition to a parent capsule of $ca'$ and $co'$, if $po$ is a port of $ca$. Informally, connector $co$ offers event $sig$ at its port $po$ and capsule $ca$ reads event $sig$ at this port.

- $\mathsf{R7}$ (propagation of external input communication)
  If connector $co$ may perform a transition with label $\mathsf{in}(sig)\mathsf{via}po$ to $co'$, then a parent capsule of $co$ may perform a transition with the same label to a parent capsule of $co'$, if $po$ is a port of the parent capsule.
- $\mathsf{R8}$ (propagation of external output communication)
  If connector $co$ may perform a transition with label $\mathsf{out}(sig)\mathsf{via}po$ to $co'$, then a parent capsule of $co$ may perform a transition with the same label to a parent capsule of $co'$, if $po$ is a port of the parent capsule.

Note that due to the modularity of the UML-RT capsule syntax and semantics definition, the syntax and semantics can be easily enhanced. For example, in order to consider UML-RT Statecharts with interlevel transitions and with entry and exit actions, we could use the (enhanced) UML-RT Statechart terms $\mathsf{UML\text{-}SC}'$ and the transition relation $\longrightarrow'$ of our earlier work [17]. Then we would only have to replace

- the set of UML-RT Statechart terms $\mathsf{UML\text{-}SC}$ in Section 2.2 in the definitions of a basic UML-RT capsule and of a complex behavioural UML-RT capsule by the (enhanced) UML-RT Statechart terms $\mathsf{UML\text{-}SC}'$ and
- the transition relation in the premise of the SOS rules $\mathsf{R2}$ and $\mathsf{R3}$ in the same section by transition relation $\longrightarrow'$.

**Port-specific Case.** As already mentioned at the end of Section 1, process-algebraic equivalence and refinement notions could be used for systematic analysis of UML-RT models. Engels et al. [3] follow this approach to define UML-RT consistency notions. As a precondition, it is neccessary to define equivalence and refinement notions on the semantics of UML-RT. However, in some cases such a notion should not be defined on the "overall" UML-RT capsule semantics, but on a UML-RT capsule semantics "restricted to" a port of the considered capsule. Therefore, we now define the port-specific semantics of a UML-RT capsule for a given port of the capsule using our already defined general case UML-RT capsule semantics.

The port-specific semantics $[\![ca]\!]_{po}$ of a UML-RT capsule $ca \in \mathsf{CAP}$ for port $po$ (with $po \in \mathsf{Ports}(ca)$) is given by the labeled transition system $(\mathsf{CAP}, L'', \twoheadrightarrow_{po}, ca) \in \mathsf{LTS}$, where

- $\mathsf{CAP}$ is the set of states,
- $L'' = \{\tau\} \cup \Pi$ is the set of labels,
- $\twoheadrightarrow_{po} \subseteq \mathsf{CAP} \times L'' \times \mathsf{CAP}$ is the transition relation defined by the three SOS rules[7] presented in Table 4, and
- $ca$ is the start state.

*Explanation of SOS rules of UML-RT capsule semantics (port-specific case)*
Informally, the port-specific semantics of a UML-RT capsule $ca$ for a port $po$ of this capsule constitutes a restriction of the general case semantics of $ca$, as follows:

[7] Note that the premises of the rules use the transition relation of the general case UML-RT capsule semantics.

**Table 4.** SOS rules of UML-RT capsule semantics (port-specific case)

$$\mathsf{P1}\quad \frac{ca \xrightarrow{\mathsf{dir}(sig)\mathsf{via}po} ca'}{ca \xrightarrow{sig}_{po} ca'} \quad (\mathsf{dir} \in \{\mathsf{in}, \mathsf{out}\})$$

$$\mathsf{P2}\quad \frac{ca \xrightarrow{\mathsf{dir}(sig)\mathsf{via}po'} ca'}{ca \xrightarrow{\tau}_{po} ca'} \quad \left(\begin{array}{c}\mathsf{dir} \in \{\mathsf{in}, \mathsf{out}\} \\ \wedge \\ po \neq po'\end{array}\right)$$

$$\mathsf{P3}\quad \frac{ca \xrightarrow{\tau} ca'}{ca \xrightarrow{\tau}_{po} ca'}$$

- P1
  A signal $sig$ occuring at port $po$ is communicated - however without any annotations like 'in', 'out' and 'via $po$'.
- P2
  No signal occuring at another port $po'$ is communicated. In this case only the internal action $\tau$ is communicated.
- P3
  If a silent transition can occur in the general case semantics, then a silent transition can also occur in the port-specific semantics.

## 4 Related Work

Our work was motivated by results from several areas: a diversity of formal semantics definitions of Statecharts (e.g. [8,15,6,17,7,16]) and the formal semantics definition of SDL from Godskesen [4].

In addition, our work was influenced by the work of Engels et al. [3]. In contrast to them, we do not restrict to atomic UML-RT models, but consider complex, hierarchical ones. Furthermore, we select labeled transition systems as semantic domain, whereas Engels et al. use CSP processes [5]. Finally, we explicitly distinguish between internal and external communication in our UML-RT semantics definition.

A lot of work exists which deals with formal semantics definition in the context of UML:

Reggio et al. [11] consider classes associated with state machines. They define a formal semantics for flat UML state machines in terms of transition systems.

Rumpe [12] defines a formal semantics for flat automata (i.e. not for hierarchical systems) based on traces.

Damm et al. [2] define the syntax and formal semantics for a subset *krtUML* of UML encompassing - among others - asynchronous signal based communication as well as synchronous communication using operation calls. Symbolic transition systems are chosen as semantic domain. krtUML models do not support hierarchical statemachines, whereas *rtUML*-models - a superset of krtUML models - do provide this support. However, a translation from a rtUML model to a krtUML model is (only) sketched. In addition, Damm et al. provide quite a detailed and well-classified overview of related work concerning formal UML semantics definitions.

Shankar and Asa [14] also deal with formal semantics definition of real-time UML behaviour, namely concurrently interacting statecharts and sequence diagrams, however they do not cover hierarchical models and use propositional linear temporal logic for defining a compositional semantics.

Arons et al. [1] present a formal semantics for a subset of UML encompassing class diagrams and state machine diagrams. They use transition systems as semantic domain. However, the considered UML subset is restricted to flat models.

## 5 Conclusions and Further Work

We presented a precise and modular syntax and semantics definition of a sublanguage of UML-RT. We followed Plotkin's style of Structured Operational Semantics (SOS) based on labeled transition systems as semantic domain. To the best of our knowledge this is the first formal semantics definition for hierarchical UML-RT models.

In future we will consider the semantics of UML 2.0 instead of UML-RT. In addition, we want to examine and adapt existing equivalence and refinement notions to be used for systematic analysis of UML-RT and UML 2.0 models.

## References

1. T. Arons, J. Hooman, H. Kugler, A. Pnueli, and M. van der Zwaag. Deductive verification of uml models in tlpvs. In T. Baar, A. Strohmeier, A. M. D. Moreira, and S. J. Mellor, editors, *UML*, volume 3273 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2004.
2. W. Damm, B. Josko, A. Pnueli, and A. Votintseva. Understanding uml: A formal semantics of concurrency and communication in real-time uml. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *FMCO*, volume 2852 of *Lecture Notes in Computer Science*, pages 71–98. Springer, 2002.
3. G. Engels, R. Heckel, J. Kuester, and L. Groenewegen. Consistency-preserving model evolution through transformations. In J.-M. Jezequel, H. Hussmann, and S. Cook, editors, *UML 2002 - The Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 2002.
4. J. Godskesen. An operational semantic model for basic sdl. Technical Report TFL RR 1991-2, Telecommunications Research Laboratory (TFL), Horsholm, 1991.
5. C. Hoare. *Communicating Sequential Processes*. Prentice Hall, London, UK, 1985.
6. D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML Statechart diagrams. In *Formal Methods for Open Object-based Distributed Systems*. Chapman & Hall, 1999.
7. G. Lüttgen, M. von der Beeck, and R. Cleaveland. A Compositional Approach to Statecharts Semantics. In *Proc. of ACM SIGSOFT Eighth Int. Symp. on the Foundations of Software Engineering (FSE-8)*, pages 120–129. ACM, 2000.
8. A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In U. Montanari and V. Sassone, editors, *CONCUR '96 (Concurrency Theory)*, volume 1119 of *Lecture Notes in Computer Science*, pages 687–702, Pisa, Italy, August 1996. Springer-Verlag.
9. R. Milner. *Communication and Concurrency*. Prentice Hall, London, UK, 1989.

10. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Department, Aarhus University, Denmark, 1981.
11. G. Reggio, E. Astesiano, C. Choppy, and H. Hussmann. Analysing UML Active Classes and Associated State Machines – A Lightwight Formal Approach. In *Fundamental Approaches to Software Engineering*, number 1783 in LNCS, pages 127–146. Springer, 2000.
12. B. Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. PhD thesis, Institut für Informatik, Technische Universität München, 1996.
13. B. Selic, G. Gullekson, and P. Ward. *Real-time Object Oriented Modeling and Design*. J. Wiley, 1994.
14. S. Shankar and S. Asa. Formal semantics of uml with real-time constructs. In P. Stevens, J. Whittle, and G. Booch, editors, *UML*, volume 2863 of *Lecture Notes in Computer Science*, pages 60–75. Springer, 2003.
15. A. Uselton and S. Smolka. A compositional semantics for Statecharts using labeled transition systems. In B. Jonsson and J. Parrow, editors, *CONCUR '94 (Concurrency Theory)*, volume 836 of *Lecture Notes in Computer Science*, pages 2–17, Uppsala, Sweden, August 1994. Springer-Verlag.
16. M. von der Beeck. A Concise Compositional Statecharts Semantics Definition. In *Proc. of FORTE/PSTV 2000*, pages 335–350. Kluwer, 2000.
17. M. von der Beeck. A structured operational semantics for UML-statecharts. *Software and Systems Modeling*, 1(2):130–141, 2002.