A Semantics for MSC based on Petri–Net Components

Stefan Heymer
University of Lübeck, Institute for Telematics
Ratzeburger Allee 160, D–23538 Lübeck, Germany
Tel. +49 451 500 3724, Fax +49 451 500 3722
E–mail `heymer@itm.mu-luebeck.de`

Abstract
In this paper, we develop a non–interleaving semantics based on Petri–net components for MSC'96 and MSC–2000. These semantics are defined using a transformational approach, closely following the lines of [6] apart from giving a direct translation.

Keywords
MSC2000, MSC'96, petri–nets, semantics

1 MOTIVATION

Recently, ITU–T released the revision of [7], MSC–2000, with a number of extensions with respect to MSC '96. With these extensions, among which are the inclusion of data and performance aspects, Message Sequence Charts are a very sophisticated formal description technique for the specification of distributed and reactive systems. Thus, one has a fully developed specification language for a constraint oriented specification of systems.

Several semantics or approaches to a semantics for MSC have been given, e.g. [15, 16, 11, 12, 17, 18, 10, 14, 13, 3, 2, 19], yet there is only a tentative semantics for full MSC'96 given in [14]. This semantics is an interleaving semantics, i. e. the behavior of a system is modeled by sequences of transitions. The intuition behind this approach is that actions are atomic and consume no time, and furthermore that at each moment in time only *one* action can be executed.

Such an interleaving semantics does not model distributed and communicating systems faithfully, because as long as there is no direct communication or synchronization between events at different locations in a distributed system, there are no causal relations between these events. With respect to this, a non–interleaving semantics would do a better job: Causally not related events in a system trace would be modeled by unrelated elements in a partially ordered trace of a system.

This especially holds if one is about to take the transition from untimed systems to timed systems, i. e. systems where actions consume time. Here, using transition systems as a system model usually leads to complicated and unintuitive results. Mostly, instantaneous actions are used together with (time consuming) delays in such models. In reality, we have actions consuming time, providing for interesting situations such as "event a happens while event b is still happening". We are convinced that this transition from untimed to timed systems can be modeled more naturally with non–interleaving models, such as Petri nets or event structures. There are already some approaches to a non-interleaving semantics for MSC, see e.g. [5, 8, 6].

In [6], a translational approach for a semantics of MSC has been given. In this paper, we will use a more direct approach: While retaining most of the translation functions from [6], we give a direct semantics based on Petri net components [9]. We have chosen Petri net components for two reasons, the first one being its intuitive closeness to MSCs, the second being the ease with which data can be incorporated into the semantics.

This paper proceeds as follows: First, we recapitulate the definitions for Petri nets and Petri net components. Next, we will give semantics for subsets of basic MSCs and High–Level MSCs. We close the paper with some conclusions and perspectives for further work.

2 PRELIMINARIES

As our formal model of choice, we will use a modified form of the components presented in [9]. Components are Petri nets with input and output places acting as an interface between the component and its environment. We will use a special variant, labeled Petri nets, which we will introduce first. Lastly, we will define the notions of components and labeled components.

2.1 PETRI NETS

We first present the basic notions from Petri net theory. As we focus on the flow of control in systems, we use place/transition systems [4] as basic formalism. We also restrict ourselves to *T–restricted* nets of *finite synchronization* [1].

We start with the definition of nets, markings and system nets.

Definition 1 *A triple $N = (P, T; F)$ is a net, if P and T are two disjoint sets, the elements of which are called places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$, called the flow relation.*

A marking M of a net N is a mapping $M : P \to \mathbb{N}$.

A system net $\Sigma = (N, M_0)$ is a net N equipped with a marking M_0 of N. M_0 is called the initial marking of Σ. □

As usual, elements of P, T and F are called *places*, *transitions* and *arcs* of the net, respectively. A place or a transition of a net N is called a *node* of N. Sometimes, we will identify the flow relation F of a net with its characteristic function.

Definition 2 *Let $N = (P, T; F)$ be a net and $n \in P \cup T$ be a node of N. We define the preset $\bullet n$ of n to be $\bullet n = \{n' \in P \cup T \mid (n', n) \in F\}$ and the postset $n\bullet$ of n to be $n\bullet = \{n' \in P \cup T \mid (n, n') \in F\}$. For a set of transitions $T' \subseteq T$ we define $\bullet T' = \bigcup_{t \in T'} \bullet t$ and $T'\bullet = \bigcup_{t \in T'} t\bullet$.*

The minimal and maximal elements of N are defined by $Min\ N = \{n \in P \cup T \mid \bullet n = \emptyset\}$ and $Max\ N = \{n \in P \cup T \mid n\bullet = \emptyset\}$, respectively.

A net is T–restricted, if $Min\ N \cup Max\ N \subseteq P$, that is, if no transition has an empty pre- or postset. A net N is of finite synchronization if for each transition $t \in T$ the preset and postset is finite. □

For the dynamics of nets, we have to define the enabling and occurrence of transitions.

Definition 3 *A transition t is enabled under a marking M, if $\forall p \in P.F(p,t) \leq M(p)$. If t is enabled under the marking M, it can occur; its occurrence leads to a new marking M' (denoted $M\ [t\rangle\ M'$), which is defined for each place p as*

$$M'(p) = M(p) - F(p,t) + F(t,p).$$

Enabling and occurrence of transitions can be generalized to sets of transitions. A set of transitions $T' \subseteq T$ is enabled under the marking M, if $\forall p \in P. \sum_{t \in T'} F(p,t) \leq M(p)$. The occurrence of a step $T' \subseteq T$ enabled under a marking M then leads to the marking M' (denoted $M\ [T'\rangle\ M'$) defined for each place $p \in P$ by

$$M'(p) = M(p) - \sum_{t \in T'} F(p,t) + \sum_{t \in T'} F(t,p).$$

$\square$

Having presented these basic notions, we now go on to give a definition of processes of Petri nets.

2.2 COMPONENTS

Components are system nets with a special subset of the places of the net serving as the interface to the environment. Thus, we introduce two sets I and O of *inputs* and *outputs*, respectively. Departing from [9], we allow inputs and outputs to overlap, and also drop quiescent transitions.

Definition 4 *Let $\Sigma = (N, M_0)$ be a system net with $N = (P,T;F)$ and let $I, O \subseteq P$ be two sets. We further demand $\bullet T \cap (O \setminus I) = \emptyset$ and $T\bullet \cap (I \setminus O) = \emptyset$. We call the triple $\mathcal{C} = (\Sigma, I, O)$ a component. A component is closed, if $I = O = \emptyset$.* $\square$

While in [9] composition of components is done by merging same places marked as input in one and output places in another net, we choose a different method of composition. We annotate transitions and places with labels, merging components at places with different direction, but the same label. For this extension, we first define labeled Petri nets.

Definition 5 *Let $N = (P,T;F)$ be a Petri net, and let $\lambda : P \cup T \to \Lambda$ be a partial labeling function for some given set of labels Λ. We call the tuple $N' = (P,T;F,\lambda)$ a labeled Petri net.*

A labeled system net is a tuple $\Sigma = (N, M_0)$, where N is a labeled Petri net.

A labeled component is a tuple $\mathcal{C} = (\Sigma, I, O)$, where Σ is a labeled system net. $\square$

The composition of labeled components is done by identifying inputs and outputs with the same label, the unified places becoming internal places of the composed system.

Definition 6 *Let $\mathcal{C}_i = (\Sigma_i, I_i, O_i)$ for $i = 1,2$ with $\Sigma_i = ((P_i, T_i; F_i, \lambda_i), M_i)$ be two labeled components. The composition $\mathcal{C}_1 \circ \mathcal{C}_2$ is defined as*

$$\mathcal{C}_1 \circ \mathcal{C}_2 = (\Sigma, I, O)$$

with $\Sigma = ((P, T; F, \lambda), M)$*, where:*

$$\begin{array}{rcl}
P & = & (P_1 \setminus (I_1 \cup O_1)) \cup (P_2 \setminus (I_2 \cup O_2)) \\
 & & \cup \{(i, o) \in P_1 \times P_2 \mid i \in I_1 \wedge o \in O_2 \wedge \lambda_1(i) = \lambda_2(o)\} \\
 & & \cup \{(o, i) \in P_1 \times P_2 \mid o \in O_1 \wedge i \in I_2 \wedge \lambda_1(o) = \lambda_2(i)\} \\
T & = & (T_1 \times \{*\}) \cup (\{*\} \times T_2) \\
F & = & \{(p, t) \in P \times T \mid (\pi_1(p), \pi_1(t)) \in F_1 \vee (\pi_2(p), \pi_2(t)) \in F_2\} \\
 & & \cup \{(t, p) \in T \times P \mid (\pi_1(t), \pi_1(p)) \in F_1 \vee (\pi_2(t), \pi_2(p)) \in F_2\} \\
\lambda & : & \left\{ \begin{array}{lcl} (p, *) & \mapsto & \lambda_1(p) \\ (t, *) & \mapsto & \lambda_1(t) \\ (p_1, p_2) & \mapsto & \lambda_1(p_1) \\ (*, p) & \mapsto & \lambda_2(p) \\ (*, t) & \mapsto & \lambda_2(t) \end{array} \right. \\
M & : & \left\{ \begin{array}{lcl} (p, *) & \mapsto & M_1(p) \\ (p_1, p_2) & \mapsto & 0 \\ (*, p) & \mapsto & M_2(p) \end{array} \right. \\
I & = & (P \cap (I_1 \times \{*\})) \cup (P \cap (\{*\} \times I_2)) \\
O & = & (P \cap (O_1 \times \{*\})) \cup (P \cap (\{*\} \times O_2))
\end{array}$$

The functions π_1 *and* π_2 *used above are the projection functions projecting a tuple onto its first and second component, respectively.* □

It is easily checked that the composition of two components is a component again.

3 A PETRI NET SEMANTICS FOR MSC

After having recapitulated the definitions from literature and having defined the notion of labeled components, we now proceed to give a semantics to a subset of MSC. We start out with a semantics for a subset of Basic MSCs, and will then extend this to a semantics for High–Level MSCs.

3.1 BASIC MSCS

To give a semantics to MSC, we start with a fragment of the language describing Basic MSCs (BMSCs). The textual syntax of this fragment is shown in Figure 1 and is a simplified form of the textual syntax for MSC'96 presented in [7]. To encode BMSCs into labeled Petri nets, we first have to fix the set of labels we will use. Letting $InstId$ be the set of instance identifiers with $env \in InstId$ being the identifier for the environment, $MscId$ be the set of Message Sequence Chart identifiers, $ActId$ be the set of action identifiers and $MsgId$ be the set of message identifiers, we define the set of labels to be

$$\begin{array}{rcl}
\Lambda & = & \{in(i, m)@j \mid i, j \in InstId, m \in MsgId\} \\
 & & \cup \{out(i, m)@j \mid i, j \in InstId, m \in MsgId\} \\
 & & \cup \{act(a)@i \mid i \in InstId, a \in ActId\} \\
 & & \cup \{msg(i, j, m) \mid i, j \in InstId, m \in MsgId\} \\
 & & \cup \{start@i \mid i \in InstId\}
\end{array}$$

$$
\begin{aligned}
&\cup \{end@i \mid i \in \mathit{InstId}\} \\
&\cup \{start, end\} \\
&\cup \{mid@i \mid i \in \mathit{InstId}\}.
\end{aligned}
$$

In this set of labels, $in(i, m)@j$ denotes the receiving of message m from instance i by instance j, $out(i, m)@j$ denotes the sending of message m from instance j to instance i, and $act(a)@i$ denotes the execution of action a by instance i. These labels will be used to label transitions of a net, i. e. they are labels for actions. The label $msg(i, j, m)$ will be used to label places containing a message m from instance i to instance j, and it will be used to label both input and output places. The two special labels $start@i$ and $end@i$ denote the start and end of instance i, respectively. We use these as labels for input and output places, an will use them to join subnets in our definition of the semantics. The label $mid@i$ will also be used in the composition of components, but will only be used as an intermediate label. The labels $start$ and end will be used for special transitions starting and terminating the components for a MSC.

```
<msc>         ::=  msc <msc name>; <msc body> endmsc;
<msc body>    ::=  <> | <inst def> <msc body>
<inst def>    ::=  instance <inst name>; <inst body> endinstance;
<inst body>   ::=  <> | <event> <inst body>
<event>       ::=  <out> | <in> | <action> | <coregion>
<out>         ::=  out <msg name> to <address>
<in>          ::=  in <msg name> from <address>
<action>      ::=  action <at>;
<address>     ::=  <inst name> | env
<coregion>    ::=  concurrent <coevents> endconcurrent;
coevents      ::=  <> | <action> <coevents>
                      | <in> <coevents> | <out> <coevents>
```

Figure 1: Simplified syntax for BMSC

Our idea is to give components for the atomic actions of input, output and performing an action in an instance. From these components, we will proceed to build the translations for higher constructs like coregions, instance bodies and MSC bodies by composing ("gluing") Petri net components.

As in [6], we proceed in a bottom–up manner, defining translations $\mathbb{T}_{\mathtt{i}}$ for each nonterminal <i> of the syntax given in Figure 1. Starting out with inputs, outputs and actions, we define the translation functions. The translation for an input action is given as

$$\mathbb{T}_{\text{in}}(i)(\texttt{in}\ m\ \texttt{from <address>}) = (\Sigma_{\text{in}}, I_{in}, O_{in})$$

with

$$\Sigma_{\text{in}} = ((S_{in}, T_{in}; F_{in}, \lambda_{in}), M_{in})$$

and

$$
\begin{aligned}
S_{in} &= \{p, p', p''\}, \\
T_{in} &= \{t\}, \\
F_{in} &= \{(p, t), (p', t), (t, p'')\},
\end{aligned}
$$

$$
\begin{array}{rcl}
M_{in} & = & \emptyset, \\
I_{in} & = & \{p, p'\} \quad \text{and} \\
O_{in} & = & \{p''\}.
\end{array}
$$

The labeling function for this component is given as

$$
\begin{aligned}
\lambda_{in} = \{ & p \mapsto start@i, p' \mapsto msg(\mathbb{T}_{address}(\texttt{<address>}), i, m), p'' \mapsto end@i, \\
& t \mapsto in(\mathbb{T}_{address}(\texttt{<address>}), m)@i\}.
\end{aligned}
$$

This component is depicted in Figure 2.

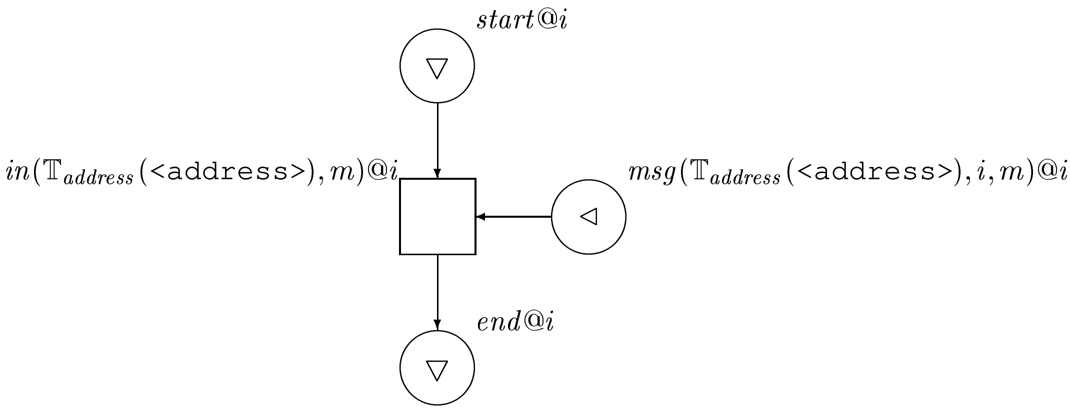

Figure 2: Translation of an input action

Analogously, the translation for an output action is given as

$$
\mathbb{T}_{\text{out}}(i)(\texttt{out}\ m\ \texttt{to <address>}) = (\Sigma_{out}, I_{out}, O_{out})
$$

with

$$
\Sigma_{out} = ((S_{out}, T_{out}; F_{out}, \lambda_{out}), M_{out})
$$

and

$$
\begin{array}{rcl}
S_{out} & = & \{p, p', p''\}, \\
T_{out} & = & \{t\}, \\
F_{out} & = & \{(p, t), (t, p'), (t, p'')\}, \\
M_{out} & = & \emptyset, \\
I_{out} & = & \{p\} \quad \text{and} \\
O_{out} & = & \{p', p''\}.
\end{array}
$$

The labeling function for this component is given as

$$
\begin{aligned}
\lambda_{out} = \{ & p \mapsto start@i, p' \mapsto msg(i, \mathbb{T}_{address}(\texttt{<address>}), m), p'' \mapsto end@i, \\
& t \mapsto out(\mathbb{T}_{address}(\texttt{<address>}), m)@i\}.
\end{aligned}
$$

This component is depicted in Figure 3.

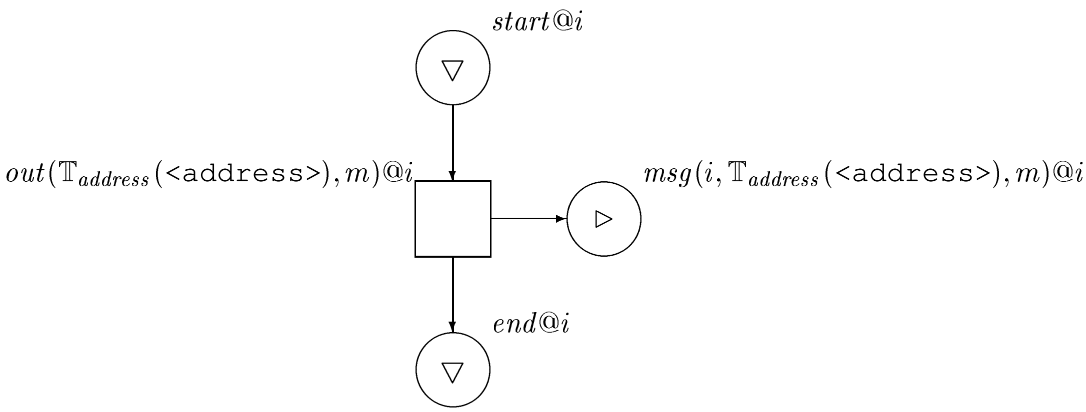

Figure 3: Translation of an output action

Lastly, the translation of an action is given by

$$\mathbb{T}_{\text{act}}(i)(\texttt{action}\, a) = (\Sigma_{act}, I_{act}, O_{act})$$

with

$$\Sigma_{act} = ((S_{act}, T_{act}; F_{act}, \lambda_{act}), M_{act})$$

and

$$\begin{aligned} S_{act} &= \{p, p'\}, \\ T_{act} &= \{t\}, \\ F_{act} &= \{(p,t),(t,p')\}, \\ M_{act} &= \emptyset, \\ I_{act} &= \{p\} \quad \text{and} \\ O_{act} &= \{p'\}. \end{aligned}$$

The labeling function for this component is given as

$$\lambda_{act} = \{p \mapsto start@i, p' \mapsto end@i, t \mapsto act(a)@i\}.$$

This component is depicted in Figure 4.

In all these translations, the places p, p', p'' and the transition t are assumed to be "fresh", i. e. the sets of places and transitions of components are assumed to be disjoint.

The translations are parameterized with the instance identifier. The motivation for this will become clear as soon as we define a translation function for the instance body. First, we define the translation $\mathbb{T}_{\texttt{address}}$ used in the translations above:

$$\begin{aligned} \mathbb{T}_{\text{address}}(i)(j) &= j, \qquad j \in \mathit{InstId} \\ \mathbb{T}_{\text{address}}(i)(\texttt{env}) &= \mathit{env} \end{aligned}$$

Coregions are quite simple to translate. We combine the components for all the events in a coregion. Yet, we cannot use the composition operator for components to do this, as input and output places of subcomponents for events in a coregion must not be glued together. For this, we define a parallel composition operator.

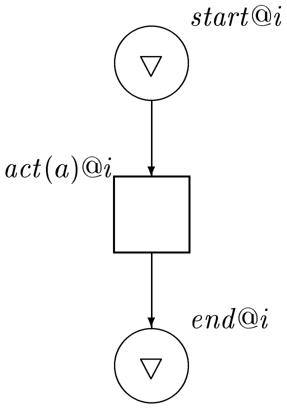

Figure 4: Translation of an action

Definition 7 *Let* $\mathcal{C}_1 = (\Sigma_1, I_1, O_1)$ *and* $\mathcal{C}_2 = (\Sigma_1, I_2, O_2)$ *be labeled components with* $\Sigma_i = ((S_i, T_i; F_i, \lambda_i), M_i)$ *for* $i = 1, 2$ *and* $(S_1 \cup T_1) \cap (S_2 \cup T_2) = \emptyset$*. The parallel composition* $\mathcal{C}_1 \parallel \mathcal{C}_2$ *of* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *is defined as*

$$\mathcal{C}_1 \parallel \mathcal{C}_2 = (\Sigma, I_1 \cup I_2, O_1 \cup O_2)$$

with

$$\Sigma = ((S_1 \cup S_2, T_1 \cup T_2; F_1 \cup F_2, \lambda_1 \cup \lambda_2), M_1 \cup M_2).$$

□

With this operation, we are able to define the translation for coregions as

$$\begin{aligned} &\mathbb{T}_{\text{coregion}}(i)(\texttt{concurrent <coevents> endconcurrent;}) \\ &\quad = \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}), \end{aligned}$$

with the translation for coevents being given by the equations

$$\begin{aligned} &\mathbb{T}_{\text{coevents}}(i)(\texttt{<>}) = \mathcal{E} \\ &\mathbb{T}_{\text{coevents}}(i)(\texttt{<action> <coevents>}) \\ &\qquad\qquad = \mathbb{T}_{\text{action}}(i)(\texttt{<in>}) \parallel \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}) \\ &\mathbb{T}_{\text{coevents}}(i)(\texttt{<in> <coevents>}) \\ &\qquad\qquad = \mathbb{T}_{\text{in}}(i)(\texttt{<in>}) \parallel \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}) \\ &\mathbb{T}_{\text{coevents}}(i)(\texttt{<out> <coevents>}) \\ &\qquad\qquad = \mathbb{T}_{\text{out}}(i)(\texttt{<out>}) \parallel \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}), \end{aligned}$$

the component $\mathcal{E}$ being the empty component

$$\mathcal{E} = (((\emptyset, \emptyset; \emptyset, \emptyset), \emptyset), \emptyset, \emptyset).$$

To translate the bodies of instances, we combine two components by "gluing" the output places labeled $end@i$ of the first component to those labeled $start@i$ of the second one. As the composition of labeled components glues places with the same label in the interfaces, we will have to relabel both components before composing them, and relabel them a second time to hide away the label of the then internal place. For this, we define a relabeling operation on components.

Definition 8 *Let $\mathcal{C} = (\Sigma, I, O)$ be a labeled component with $\Sigma = ((S,T;F,\lambda), M)$. Further let $\Xi : \Lambda \to \Lambda$ be a partial function on labels with the restriction that Ξ is total on $\{l \in \Lambda \mid \exists p \in I \cup O.\lambda(p) = l\}$. We define the relabeling $\mathcal{C}\{\Xi\}$ of $\mathcal{C}$ by Ξ to be*

$$\mathcal{C}\{\Xi\} = (\Sigma', I, O)$$

with $\Sigma' = (N', M)$ and $N' = (S,T;F,\Xi \circ \lambda)$. □

Now, we are able to join two components with the operation

$$\mathcal{C}_1 \bowtie \mathcal{C}_2 := (\mathcal{C}_1\{\Xi_1\} \circ \mathcal{C}_2\{\Xi_2\})\{\Xi_3\},$$

the relabeling functions being given as

$$\begin{aligned}
\Xi_1(l) &= \begin{cases} \mathit{mid}@i, & x = \mathit{end}@i \\ x & \text{otherwise}, \end{cases} \\
\Xi_2(l) &= \begin{cases} \mathit{mid}@i, & x = \mathit{start}@i \\ x & \text{otherwise, and} \end{cases} \\
\Xi_3(l) &= \begin{cases} \text{undefined}, & x = \mathit{end}@i \\ x & \text{otherwise.} \end{cases}
\end{aligned}$$

With this operation, we are able to define the translation of an instance body as

$$\begin{aligned}
&\mathbb{T}_{\text{inst body}}(i)(\texttt{<>}) = \mathcal{E} \\
&\mathbb{T}_{\text{inst body}}(i)(\texttt{<event> <inst body>}) \\
&\quad = \mathbb{T}_{\text{event}}(i)(\texttt{<event>}) \bowtie \mathbb{T}_{\text{inst body}}(i)(\texttt{<inst body>}),
\end{aligned}$$

with $\mathbb{T}_{\text{event}}$ being defined as

$$\begin{aligned}
\mathbb{T}_{\text{event}}(i)(\texttt{<in>}) &= \mathbb{T}_{\text{in}}(i)(\texttt{<in>}) \\
\mathbb{T}_{\text{event}}(i)(\texttt{<out>}) &= \mathbb{T}_{\text{out}}(i)(\texttt{<out>}) \\
\mathbb{T}_{\text{event}}(i)(\texttt{<action>}) &= \mathbb{T}_{\text{action}}(i)(\texttt{<action>}) \\
\mathbb{T}_{\text{event}}(i)(\texttt{<coregion>}) &= \mathbb{T}_{\text{coregion}}(i)(\texttt{<coregion>}).
\end{aligned}$$

For the definition of instances, the instance identifier in the definition is used as an additional parameter for the translation function for instance bodies, giving the definition

$$\begin{aligned}
&\mathbb{T}_{\text{inst def}}(\texttt{instance}\, i\texttt{; <inst body> endinstance;}) \\
&\quad = \mathbb{T}_{\text{inst body}}(i)(\texttt{<inst body>}), \qquad\qquad i \in \mathit{InstId}.
\end{aligned}$$

With these translations at hand, the translation of a Message Sequence Chart is no problem: the instances in its body are translated, and the components provided by these translations are glued together to join the output and input of messages between instances. Thus, the definition for $\mathbb{T}_{\text{msc body}}$ reads

$$\begin{aligned}
&\mathbb{T}_{\text{msc body}}(\texttt{<>}) = \mathcal{E} \\
&\mathbb{T}_{\text{msc body}}(\texttt{<inst def> <msc body>}) \\
&\quad = \mathbb{T}_{\text{inst def}}(\texttt{<inst def>}) \circ \mathbb{T}_{\text{msc body}}(\texttt{<msc body>}).
\end{aligned}$$

This way, the need to explicitly encode communication between instances leading to the formulation of a channel process in [6] is taken away, this is being done here by joining the interface places labeled $msg(i,j,m)$ of components.

For the translation of a basic MSC, we translate the MSC body to a component. Furthermore, we parameterize the translation function with an environment ρ, which will be used in the translation of High–Level MSCs in the next Subsection. The result of the translation function will be an environment, binding the name of the MSC to the component yielded by the translation. Hence, the translation function for MSCs is defined as

$$\begin{aligned}&\mathbb{T}_{\text{msc}}(\rho)(\texttt{msc}\, m\texttt{; <msc body> endmsc;})\\ &\qquad = \{m \mapsto \mathbb{T}_{\text{msc body}}(\texttt{<msc body>})\}.\end{aligned}$$

3.2 HIGH–LEVEL MSCS

Up to now, the definitions for the translation functions were quite similar to the ones given in [6]. For MSC expressions, we will have to deviate from this path. This is the case as the semantics of alternatives in MSC expressions demands a *delayed* choice between alternatives — the choice between two alternative MSCs is taken in the moment the two MSCs start to differ. To model this intended semantics, we will have to leave the non–interleaving semantics for the moment of choice.

But first, let us consider the extensions of the syntax to allow the specification of High–Level MSCs. These additional rules are shown in Figure 5. These rules can also be found in [7]. Here, we restrict the `<msc name>` to be names of Basic MSCs.

```
<msc>            ::=  msc <msc name>; expr <msc expr> endmsc;
<msc expr>       ::=  <start>; <node exprs>
<start>          ::=  <label> <alternatives>
<alternatives>   ::=  <> | alt <label> <alternatives>
<node exprs>     ::=  <> | <node expr> <node exprs>);
<node expr>      ::=  <label>:  <node> seq (<label> <alternatives>);
                      | <label>:  end
<node>           ::=  empty | <msc name> | <par expr> | connect
<par expr>       ::=  expr <msc expr> endexpr <par exprs>
<par exrs>       ::=  <> | par expr <msc expr> endexpr <par exprs>
```

Figure 5: Simplified syntax for HMSC

Let *Labels* be a set of node labels for a High–Level MSC. We will give the semantics for High–Level MSCs starting from node expressions, following this by an explanation how the components yielded by the translation of node expressions are aggregated into components for MSC expressions.

Again, we will give translation functions for the nonterminals. We will tag the start and end places of the components with additional sets of labels for nodes of an HMSC. That is, we extend the labeling set to:

$$\begin{aligned}\Lambda \quad = \quad & \{in(i,m)@j \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\}\\ & \cup \{out(i,m)@j \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\}\\ & \cup \{act(a)@i \mid i \in \mathit{InstId}, a \in \mathit{ActId}\}\\ & \cup \{msg(i,j,m) \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\}\\ & \cup \{start@i \mid i \in \mathit{InstId}\}\\ & \cup \{end@i \mid i \in \mathit{InstId}\}\\ & \cup \{mid@i \mid i \in \mathit{InstId}\}\end{aligned}$$

$$\cup \{L : start@i \mid i \in InstId, L \subseteq Labels\}$$
$$\cup \{L : end@i \mid i \in InstId, L \subseteq Labels\}.$$

For node expressions, each node expression returns a binding of the label to a component and a set of node labels. These sets determine the set of alternatives which may be attached to the output part of the component. These bindings are accumulated recursively from the translation of single node expressions.

$$\begin{aligned}
&\mathbb{T}_{\text{node exprs}}(\rho)(\texttt{<>}) = \{\} \\
&\mathbb{T}_{\text{node exprs}}(\rho)(\texttt{<node expr> <node exprs>}) \\
&\quad = \mathbb{T}_{\text{node expr}}(\rho)(\texttt{<node expr>}) \cup \mathbb{T}_{\text{node exprs}}(\rho)(\texttt{<node exprs>})
\end{aligned}$$

The simplest node expression is an end node, which is translated to an empty component, allowing no further attachments of components:

$$\mathbb{T}_{\text{node expr}}(l\texttt{:}\quad \texttt{end}) = \{l \mapsto (\mathcal{E}, \emptyset)\}$$

For all other node expressions, we translate the node, allowing as attachments the set of alternatives for that node. Hence, we get the translation:

$$\begin{aligned}
&\mathbb{T}_{\text{node expr}}(\rho)(l_1\texttt{:}\quad \texttt{<node> seq (}l_2\,\texttt{<alternatives>);}) \\
&\quad = \{l_1 \mapsto (\mathbb{T}_{\text{node}}(\rho)(\texttt{<node>}), \{l_2\} \cup \mathbb{T}_{\text{alternatives}}(\texttt{<alternatives>})\}
\end{aligned}$$

The alternatives are just accumulated as a set of labels:

$$\begin{aligned}
\mathbb{T}_{\text{alternatives}}(\texttt{<>}) &= \{\} \\
\mathbb{T}_{\text{alternatives}}(\texttt{alt}\ l\ \texttt{<alternatives>}) &= \{l\} \cup \mathbb{T}_{\text{alternatives}}(\texttt{<alternatives>})
\end{aligned}$$

For the translation of a node, empty nodes and connectors are mapped to empty components $\mathcal{E}$. This will be caught in the translation of MSC expressions, the set of alternatives following the empty node or connector will be filled in in the translation of the node expression.

For MSC references, the component for the referenced basic MSC is inserted, the same holding for parallel expressions. Generally, this restriction could be relaxed when using a fixpoint construction for the semantics of MSC documents.

$$\begin{aligned}
\mathbb{T}_{\text{node}}(\rho)(\texttt{empty}) &= \mathcal{E} \\
\mathbb{T}_{\text{node}}(\rho)(m) &= \rho(m), \qquad m \in MscId \\
\mathbb{T}_{\text{node}}(\rho)(\texttt{<par expr>}) &= \mathbb{T}_{parexpr}(\rho)(\texttt{<par expr>}) \\
\mathbb{T}_{\text{node}}(\rho)(\texttt{connect}) &= \mathcal{E}
\end{aligned}$$

For parallel expressions, the translation is simply the parallel composition of the components translated from the MSC expressions in the parallel expression. Hence, we define

$$\begin{aligned}
&\mathbb{T}_{\text{par expr}}(\rho)(\texttt{expr <msc expr> endexpr <par exprs>}) \\
&\quad = \mathbb{T}_{\text{msc expr}}(\rho)(\texttt{<msc expr>})\mathbb{T}_{\text{par exprs}}(\rho)(\texttt{<par exprs>})
\end{aligned}$$

and

$$\begin{aligned}
&\mathbb{T}_{\text{par exprs}}(\rho)(\texttt{<>}) = \mathcal{E} \\
&\mathbb{T}_{\text{par exprs}}(\rho)(\texttt{par expr <msc expr> endexpr <par exprs>})\ , \\
&\quad = \mathbb{T}_{\text{msc expr}}(\rho)(\texttt{<msc expr>})\mathbb{T}_{\text{par exprs}}(\rho)(\texttt{<par exprs> }).
\end{aligned}$$

with the translation of MSC expressions defined in the following paragraphs.

To translate a MSC containing an MSC expression, we produce a binding of the name of the MSC to the translation of the expression.

$$\begin{aligned}&\mathbb{T}_{\text{msc}}(\rho)(\texttt{msc}\,m\texttt{; expr <msc expr> endmsc;})\\&\qquad = \{m \mapsto \mathbb{T}_{\text{msc expr}}(\rho)(\texttt{<msc expr>})\}\end{aligned}$$

The environment of bindings ρ used in this definition binds the names of the basic MSCs which can be used as HMSC nodes.

Translating a MSC expression is more complicated. We will give the definition first, and will explain it afterwards:

$$\begin{aligned}&\mathbb{T}_{\text{msc expr}}(\rho)(\texttt{<start>; <node exprs>})\\&\qquad = \mathit{unroll}(\mathcal{S}(\mathbb{T}_{\text{start}}(\texttt{<start>})), \mathbb{T}_{\text{node exprs}}(\rho)(\texttt{<node exprs>}))\end{aligned}$$

The translation of the alternatives given as the start node returns just a set of labels which will be used as the tags of the input places of the component into which the HMSC will be translated:

$$\mathbb{T}_{\text{start}}(l\ \texttt{<alternatives>}) = \{l\} \cup \mathbb{T}_{\text{alternatives}}(\texttt{<alternatives>})$$

This deviates from the path taken in [6], as the delayed choice operator cannot be given abstractly just on the names of components or nets. Instead, we tag the input or output places of the component for the node with the set of labels of the alternatives.

In the definition for the translation of MSC expressions, we start out with a starting component, whose output places are tagged with the set of alternatives, as it is shown in Figure 6. Starting from this component, the *unroll* function (which we will not formally define here) will start to add transitions from the alternatives, building a kind of branching process of the components translated from the node expressions. As soon as a transition is found that is not common to the components, a transition is inserted that takes the tokens from *all* maximal places of the common prefix, is labeled as the transition not included in the common prefix, and that is attached to the remainder of the (unrolling of) the components that share this transition. Places with labels that occurred earlier in the unrolling process are not replicated in this process, but back arrows are introduced instead.

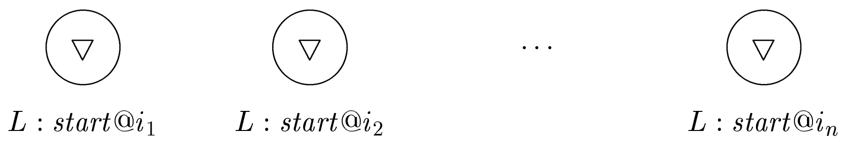

Figure 6: Parameterized starting component $\mathcal{S}(L)$

This way, the delayed choice from [7] is translated to Petri net components. Yet, this approach does break the intended non–interleavingness of the semantics in the moment of choice between alternatives, as in this very moment, only *one* transition can be fired, although the alternative components might enable more than one transition.

This concludes our presentation of a semantics for High–Level MSCs based on Petri net components.

4 CONCLUSIONS AND FUTURE WORK

What have we achieved so far? We have shown how to construct a non–interleaving semantics for MSC based on Petri net components. In our construction we follow the path shown in [6], although giving a direct translation instead of a two–step one.

We have not considered the inclusion of data in the scope of this paper, but Petri nets lend themselves to this — it is simple to go over from place/transition nets to colored nets, retaining the structure of the components, while switching from tokens to data objects. The inclusion of the performance related aspects of MSC–2000 into the semantics is not that simple, but these performance requirements could be expressed on processes of the components.

For our future work, we will concentrate on the introduction of timing aspects into the semantics shown here, working towards a semantics for a timed variant of MSC'96 and MSC–2000, respectively.

5 ACKNOWLEDGEMENTS

I would like to thank the anonymous referees for their helpful and demanding comments on my proposal. Thanks also have to go to Michael Ebner and Michael Schmitt for their comments on the final version of this paper.

REFERENCES

[1] E. Best and C. Fernandez. *Non–sequential Processes*. Number 13 in EATCS Monographs on Theoretical Computer Science. Springer–Verlag, 1988.

[2] W. Damm and D. Harel. Lscs: Breathing life into message sequence charts. Technical Report Reseach Report CS98-09, Weizmann Institute, 1998.

[3] Thomas Gehrke, Michaela Huhn, Arend Rensink, and Heike Wehrheim. An algebraic semantics for message sequence charts documents. In *Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE/PSTV '98)*, pages 3–18. Kluwer Academics Publisher, 1998.

[4] U. Goltz and W. Reisig. The non–sequential behaviour of Petri Nets. *Information and Control*, (57):125–147, 1983.

[5] J. Grabowski, P. Graubmann, and E. Rudolph. Towards a Petri net based semantics definition for Message Sequence Charts. In O. Færgemand and A. Sarma, editors, *SDL'93: Using Objects*, Proceedings of the Sixth SDL Forum, pages 179–190, Darmstadt, 1993. Amsterdam, North-Holland.

[6] St. Heymer. A Non–Interleaving Semantics for MSC. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC — SAM'98, Informatik–Berichte Humboldt–Universität zu Berlin*, June 1998.

[7] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1996.

[8] J.-P. Katoen and L. Lambert. Pomsets for Message Sequence Charts. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC — SAM'98, Informatik–Berichte Humboldt–Universität zu Berlin*, June 1998.

[9] E. Kindler. A Compositional Partial Order Semantics for Petri Net Components. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997, Proceedings*, volume 1248 of *LNCS*. Springer–Verlag, 1997.

[10] P. Kosiuczenko. Formalizing msc'96: Inline expressions. Technical Report Technical Report Nr. 9705, Ludwig–Maximilians–Universität München, Institut für Informatik, January 1997.

[11] P. B. Ladkin and S. Leue. What do Message Sequence Charts mean? In R. L. Tenney, P. D. Amer, and M. Ü. Uyar, editors, *Formal Description Techniques VI*, IFIP Transactions C, Proceedings Sixth International Conference on Formal Description Techniques, pages 301–316, Boston, 1994. North-Holland.

[12] P.B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7(5):473–509, 1995.

[13] S. Mauw and M. A. Reniers. High-Level Message Sequence Charts. In A. Cavalli and A. Sarma, editors, *SDL'97: Time for Testing — SDDL, MSC and Trends*, Proceedings of the Eigth SDL Forum, pages 291–306, Evry, France, 1997. Institut national des tèlècommunications, North-Holland.

[14] S. Mauw and M. A. Reniers. Operational Semantics for MSC'96. In A. Cavalli and D. Vincent, editors, *SDL'97: Time for Testing — SDL, MSC and Trends*, Tutorials of the Eigth SDL Forum, pages 135–152, Evry, France, 1997. Institut national des tèlècommunications, North-Holland.

[15] S. Mauw and M.A. Reniers. An algebraic semantics of Basic Message Sequence Charts. Technical Report CSN 94/17, Eindhoven University of Technology, Department of Computing Science, Eindhoven, 1994.

[16] S. Mauw and M.A. Reniers. An algebraic semantics of Message Sequence Charts. Technical Report CSN 94/23, Eindhoven University of Technology, Department of Computing Science, Eindhoven, 1994.

[17] M. A. Reniers. Syntax requirements of Message Sequence Charts. In R. Bræk and A. Sarma, editors, *SDL'95 with MSC in CASE*, Proceedings of the Seventh SDL Forum, pages 63–74, Oslo, 1995. Amsterdam, North-Holland.

[18] M. A. Reniers. Static semantics of Message Sequence Charts. Technical Report CSR 96-19, Eindhoven University of Technology, Department of Computing Science, 1996.

[19] M. A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1998.