Compositional Transformation of Message Sequence Charts into High–level Petri Boxes (Extended Abstract)

Daniel Gurovič und W. Fengler
{gurovic|fengler}@theoinf.tu-ilmenau.de
Department of Computer Science and Automation University of Ilmenau, Germany

Abstract

The concern of this paper consists in utilizing the theoretical findings on the field of formal compositional semantics, in particular those of high level Petri nets, for the specification of communicating embeded systems. Transformation of these approaches, and their evaluation in practice occurs with the aid of one of the formal rotation of the UML namely the *Message Sequence Charts*, particularly the MSC'96. Communication requirements of embedded systems are graphically specified and then transformed into an executable class of high level Petri nets.

Keywords: Petri nets, compositional semantics, specification, MSC

1. Introduction

Message Sequence Charts (MSCs) have established themselves as a standard description notation for the specification of comminicating embedded applications as known for example from the telecomminication area,. Their propagation found even entry into the UML which is becoming more and more popular in the indus-try. Z.120 [IT96] served here as a pattern for so – called *Sequence Charts* defined within the UML. The first effort to standardize the semantics of MSCs by means of a process algebra was undertaken in annex B of Z.120. As for all description notations it also applies to MSCs that a formally defined semantics is necessary in order to make conclusions about the system properties or to put it on the same formal basis as another system design notation like SDL in [FG98], gaining so the possibility to check them against each other, that is to verify whether the designed system meets its specification.

For two reasons we decided for so – called M–nets as our semantic domain. M – nets firstly have an algebraic structure and thus allows a compositional transformation. Secondly there is a possibility of a tool based analysis within the PEPTool [Gra97] and the export into another Petri net format.

We proceed as follows: In section 2 we repeat the most important facts about M – nets from [BFF^{+}95]. Afterwards, in section 3 we describe the transformation process of MSCs into that high – level Petri nets. We begin with the *High – Level Message Sequence Charts* and go ahead with the *Basic Message Sequence Charts* then. The transformation is based on the textuell syntax description in IT96] and is, due to a better readability, described in the same way as in annex B of Z.120 standard.

2. M–Nets

In this section we describe the syntactical and semantical domain of our transformation of MSCs into Petri nets. We use a special class of Petri nets so–called

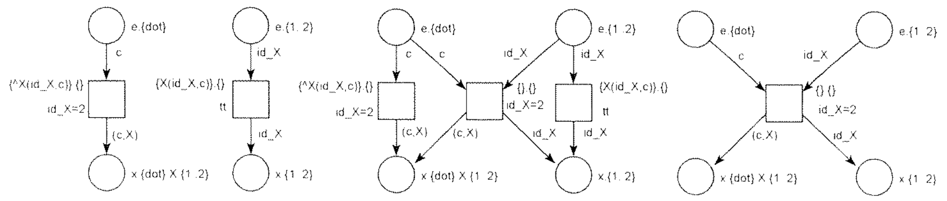


Figure 1 : From the left : $N_1||N_2$, $(N_1||N_2)$ sy X and $(N_1||N_2)$ sy X rs X

M –nets — a version of high–level Petri–Boxes [BDH92]. They were developed with the goal to give a fully compositional semantics for concurrent program–ming languages and system notations. More precise definitions are to be found in [BDH92, BFF^+95, DK95]. First, we give, soma auxiliary definitions and repeat the basic definitions concerning the M–nets.

Let Q be a set, A *multi –set* over Q is a function $\mu : Q \rightarrow N_0$. The set of all finite multi–sets over Q is described as $M_f(Q)$. Further we assume the existence of sets *Val*, *Var*, **A** and **B** of *values, variables, action symbols* and *tie–symbols*, respectively. Action symbols have arbitrary many parameters $ar(A) \in N_0$. There is a bijective function defined over **A** called conjugation $^-$: A $\rightarrow$A satisfying $\forall A \in \mathbf{A} : \overline{A} \neq A \wedge \overline{\overline{A}} = A$ and $ar(\overline{A}) = ar(A)$. A construct $A(i_1,\ldots,i_{ar(A)})$ is an *action term* if A is an action symbol and $\forall j : 1 \leq j \leq ar(A) : i_j \in Val \cup Var$. The set of all action terms is described as AX. A construct $b^{\{+|-\}}(a)$ is a *link symbol* if $b \in B$ and $a \in Val \cup Var$.

Definition 2.1 An M–net is a triple (S,T,ι), such, that S, T is the set of places and transtions respectively and ι is a function called inscription with domain $P \cup T \cup (P \times T) \cup (T \times P)$ such that:

- *$\forall s \in S$, $\iota(s)$ is a pair $\lambda(s).\tau(s)$ with $\lambda(s) \in \{e, i, x\}$ and $\tau(s) \subseteq Val$ the place type.*
- *$\forall t \in T$, $\iota(t) = \lambda(t).\gamma(t)$ with $\lambda(t) = \alpha(t).\beta(t)$ called the label of t, such that*

  $\alpha(t) \in \mathcal{M}_f(AX_h)$ finite multi–set of action terms and $\beta(t) \in \mathcal{M}_f(B_h)$ is the link symbol; $\gamma(t)$ is the guard of t.
- *$\forall (s,t) \in (S,T)$, $\iota((s,t)) \in \mathcal{M}_f(Val \cup Var)$; analogously applies to $(t,s) \in (T,S)$.*

A marking of an M–net (S,T,ι), M : S $\rightarrow$ M$_f(Val)$, assigns to each place s a multi–set of values from $\tau(s)$. The transition firing rule corresponds to the one of colored Petri nets, i.e. transition t is enabled in M if there is an enabling binding σ for variables in the inscription of t such that there are enough tokens of the appropriate type to satisfy the required token amount and the guard $\gamma(t)[\sigma]$ is true. The effect of occurrence of t consists in removing tokens from the input places and adding tokens to its output places according to $\iota((s,t))$ and $\iota((t,s))$ respectively.

M–nets represent a combination of colored and labeled high–level nets. The net elements i.e. places, transitions and arcs axe labeled in addition to types, terms and transition guards with auxiliary information which determines the status of each place (entry, exit or internal) and assigns to each transition a communication resp. refindment interface. These additional inscriptions $\lambda(s)$ and $\lambda(t)$ were introduced in order to define some compositional operators over M–nets [BFF+95]. These are either operators which determine the control flow or the communication with the environment. To the former set belongs the sequential, parallel, alternative and iteractive composition (;, ||, [] resp. [**]). The later group contains operations for synchronous and asynchronous communication, sy bzw. tie. The latter one was introduced first in [KP99].

As already mentioned above, transitions are labeled with multi–sets of actions. They represent the capability to a synchronous communication With the environ ment.

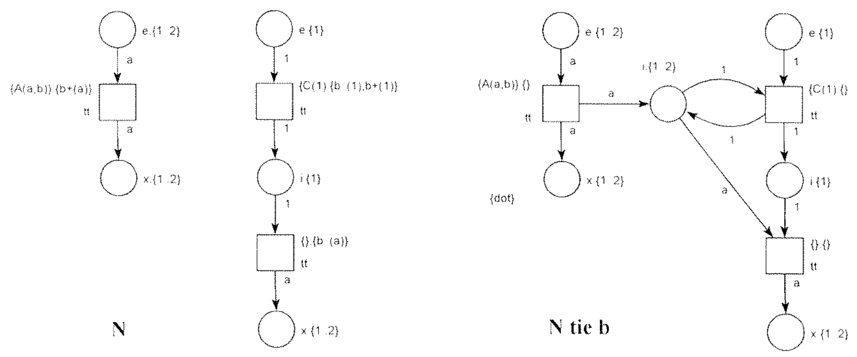


Figure 2: From the left: N and N tie b

Its mechanism is illustrated by means of the figure 1. Nets N_1 and N_2 each contains one transition with X(., .) resp. i.e. both transitions can synchronine, in respect of X. The result is in the middle of the figure. Transition svnchronizaticarr gives rise to a new transition labeled with $\alpha 1 + \alpha 2 - \{X(.,.), \overline{X}(.,.)\}$, i.e. a sum of the both multi–sets minus the synchronization pair. Afterwards, the resulting net is restricted with respect to the action *X* by the *rs*–operator (*restriction*). This restriction consists in deleting all transitions labeled with X from the net.

For the explanation of the *tie*–operator we use a figure 2 from [KP99]. In the left hand part of this figure the *tie*–operator takes an M–net *N* and a link symbol *b* and produces an M–net *tie b* in the right hand part which is like *N* but has an additional internal place s_b of the same type as b, and additional arcs to and from s_b. These arcs connect s_b to every transition *t* with b^+ resp. b^- in such a way that (t, s_b) and (s_b, t) arise. The inscription of the new arcs corresponds to the argument of the link symbol *b*. In other words, the *tie*–operator realizes a kind of intermediate storage for information which occurs somewhere in the net.

Another important operator which we will make use of is a transition refinement $N[X \leftarrow N]$. It is allowed only for hierarchical transitions, i.e. transitions labeled with a variable. Beyond this, input and output places surrounding the transition to be refined, ${}^{\bullet}t \cup t^{\bullet}$, must be of the same type. A precise definition of the refinement operator for high–level Petri–nets is in [DK95].

3. Transformation of MSCs into M–nets

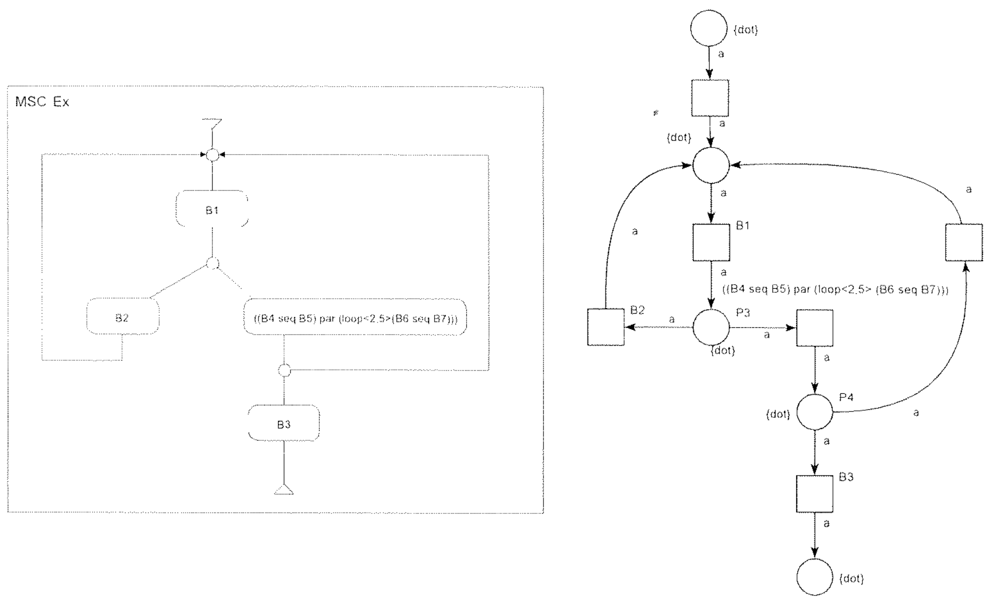
In this section we describe our approach for transformation of MSCs into M–nets. For the illustration of our approach we use some small examples of an *HMSC Ex*, *MSC Driving* and *MSC Ref* in figure 3, 8 resp. 10.

The semantic function *S* translates every MSC into an M–net. In the style of [IT96] the general appearance of *S* is

$$S_{<X>}[\![i]\!] : \mathcal{L}(<\mathrm{X}>) \rightarrow M\text{-}net$$

with <X> a non–terminal from the grammar of MSC'96 [IT96] and *i* the actual argument representing (part of) an MSC. For example, let $a_1, a_2 \in L$ (<*msc ref expr*>) i.e. two sequences of the language derived from the non–terminal symbol <*msc ref expr*> then a_1 and a_2 **alt** a_2 are valid argument, the latter one representing a sequence constructed from a_1 , the terminal symbol **alt** and a_2.

We start with the HMSCs and proceed then in a top–down manner with the BMSCs, instances, events and coregions until the references and inlines. Further, we assume that a maximum number *max_i* of possibly concurrent instances[1] of one and the same BMSC is given, $max\_i \in$ N and that places without any type inscription carry implictly as their type theset{dot}.



MSC Ex
B1
B2
((B4 seq B5) par (loop<2,5>(B6 seq B7)))
B3
{dot}
a
B1
B2
P3
P4
B3
((B4 seq B5) par (loop<2,5> (B6 seq B7)))

Figure 3: Ein *HMSC Ex*–Scenario und das zugeh" origes M–Netz

3.1 MSC–Document

1 Not to confuse with process instances a BMSC consist of.

In dealing with MSCs one MSC–document is a combination of High–level Message Sequence Charts (HMSCs) and Basic Message Sequence Charts (BMSCs). The HMSCs determine the control flow within the scenario and represent a means of abstraction. Therefore the meaning of an MSC–document results froth the parallel composition of all referenced BMSCs and the M–net of the actual HMSC. All BMSCs are translated only once. Different references of one and the sane BMSC are translated as a sequence of a reference call and termination similar to a procedure call in an imperative programming language [FG97]. Thereby it is possible to keep the M–net size small.

Definition 3.1 *Let* $m \in \mathcal{L}(<msc\ doc>)$, $i \in \mathcal{L}(<bmsc>)$ *and* $j \in \mathcal{L}(<hmsc>)$

such that

$$S_{<msc\ doc>}[\![m]\!] =$$
$$[sync : (\|_{i \in AllBmsc(m)} S_{<bmsc>}[\![i]\!]) \parallel S_{<hmsc>}[\![ExtrHmsc(j)]\!]]$$

where sync $= \{M_c, M_r \mid c, r \in AllRefName(m)\}$ *is the set of all MSC referrcnce Calls and terminations of all referenced BMSCs in m referenzierten BMSCs and where analogous AllBmsc() is the set of all BMSC and ExtrHmsc() the set of all HMSC definitions.*

Expression [a : E] is a shortened spelling for (E sy a rs a).

3.2 High–level MSC

High–level MSCs describe how a set of BMSCs is combined within a single dia–gram and therefore they describe the control flow between several BMSCs. An HMSC is a directed graph *G*= (*N, V*) whose nodes are either a start symbol, an end symbol, an MSC reference, a connector, a condition or a parallel frame,

i.e. $N = N_S \cup N_T$ with $N_S = N_{startsymbol} \cup N_{endsymbol} \cup N_{connector} \cup N_{condition}$ and $N_T = N_{reference} \cup N_{frame}$.

Definition 3.2 *Let* $R_a \in \mathcal{L}(<node>)$ *then the semantics of an HMSCs* H *is defined as a refinement* $S_{<hmsc>}[\![H]\!] = N[X_a \leftarrow S_{<node>}[\![R_a]\!] \mid \forall a \in N_T]$ *where* $N = (S, T, \iota)$ *with,*

$$S = N_S \times \{*\} \cup \{(x,y) | x, y \in N_T \wedge (x,y) \in V\}$$

$$T = N_T \times \{*\} \cup \{(x,y) | x, y \in N_S \wedge (x,y) \in V\}$$

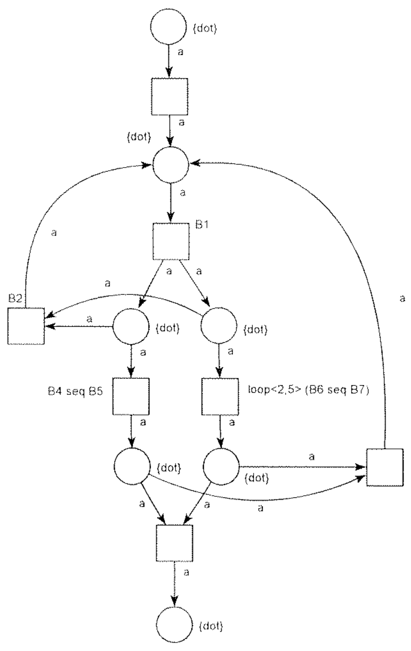
$$W((x,*),(y,*)) = V(x,y), \text{if}(x,y) \in N_S \times N_T \cup N_T \times N_S$$

Figure 4: $S_{\text{<hmsc>}}[\![\text{Ex}]\!] = \ldots\ S_{\text{<node>}}[\![\ \ldots\ \text{par}\ \ldots\ ]\!]\ \ldots$

$$W((x,*),(x,y)) = V(x,y), \text{if}(x,y) \in N_S \times N_S \cup N_T \times N_T$$

$$W((x,y),(y,*)) = V(x,y), \text{if}(x,y) \in N_S \times N_S \cup N_T \times N_T$$

$$\iota((a,b)) = \begin{cases} \{e\}.\{dot\}, & if \quad a \in N_{startsymbol} \wedge b = * \\ \{x\}.\{dot\}, & if \quad a \in N_{Endsymbol} \wedge b = * \\ \emptyset.\{dot\}, & wenn \quad a \in N_S \backslash N_{startsymbol} \cup N_{endsymbol} \quad or \\ & a \neq * \neq b \wedge (a,b) \in N_S \times N_S \cup N_T \times N_T \\ \{X\}.\emptyset.\emptyset, & wenn \quad a \in N_T \wedge X_a \in Var \end{cases}$$



where $S_{<node>}$ is the *semantic function for the translation of an* HMSC node.

This means that a start, an end, a connector, a condition symbol and edges between any two symbols from N_T become places. All MSC reference, a parallel frame and any two edges between symbols from N_S become transitions. The resulting M–net is then refined with the meaning of the MSC references.

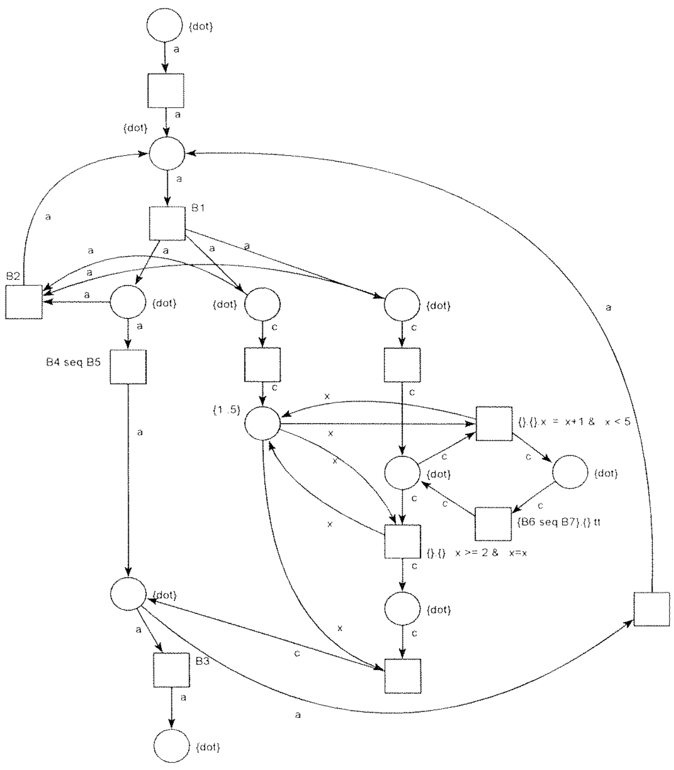


Figure 5: $S_{<\text{hmsc}>}[\![\text{Ex}]\!]= \dots\ S_{<\text{node}>}[\![\text{loop} \dots\ ]\!] \dots$

An example of all HMSC and corresponding M–net are depicted ill figure 3. It describes the order of individual BMSCs, namely at first an MSC reference *B1* occurs and then a connector node comes where the flow branches out either to the reference *B2* or to another reference consisting of a complex expression.

$$((B4\ \mathbf{seq}\ B5)\ \mathbf{par}\ (\mathbf{loop}\ B6\ \mathbf{seq}\ B7)))$$

Definition 3.3 *Let name* $\in \mathcal{L}(<\mathit{msc\ name}>)$, *par* $\in \mathcal{L}(<\mathit{par\ expr\ list}>)$, *exp*

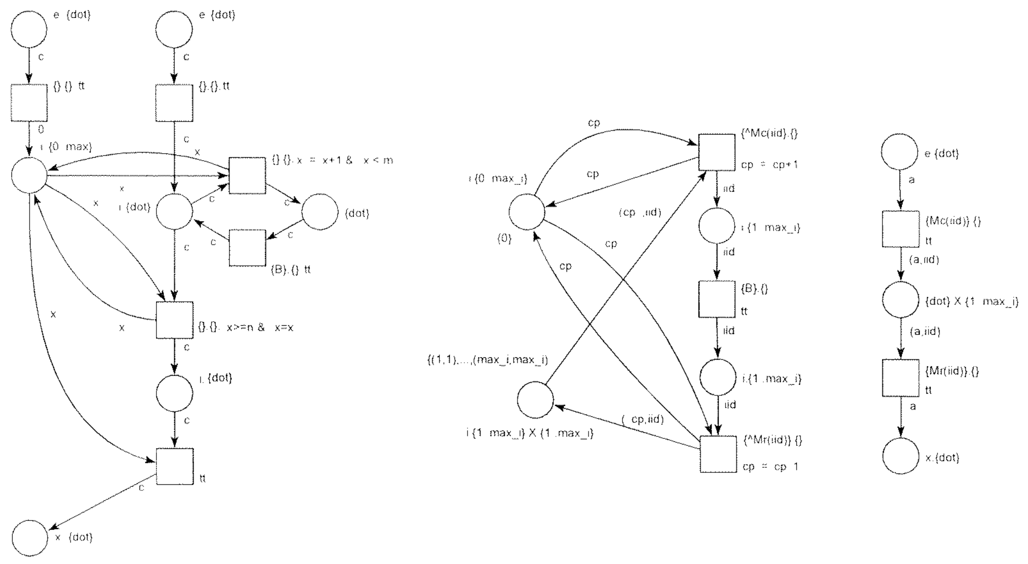


Figure 6: from the left: $N^{<n,m>}_{Loop}$, N_{MSCDed} and $Nref$

∈ λ L(<msc ref expr> and H be an HMSC, then $S_{<node>}$ is defined as

$$
\begin{aligned}
S_{<node>}[\![par]\!] &= \|_{i \in AllParHmsc(par)}\ S_{<node>}[\![i]\!] \\
S_{<node>}[\![exp]\!] &= S_{<msc\ ref\ expr>}[\![exp]\!] \\
S_{<node>}[\![name]\!] &= \begin{cases} N_{ref}, & \textit{if name is an BMSC} \\ S_{<hmsc>}[\![H]\!], & \textit{if name is an HMSC H} \end{cases}
\end{aligned}
$$

where AllParHmsc(par) is the set of all HMSCs within a parallel frame par.

This definition says that a parallele frame is translated by means of the parallel operator ||. A reference expression combines a number of MSCs to one and is consequently translated according to the following definition (see figures 4 and 5). Finally, an BMSC–reference is translated as a procedure call. The net N_{ref} is in figure 6.

Definition 3.4 *Let $a_1, a_2, exp \in \mathcal{L}(<msc\ ref\ expr>)$, $p_1, p_2 \in \mathcal{L}(<msc\ ref\ par\ expr>)$, $s_1, s_2 \in \mathcal{L}(<msc\ ref\ seq\ expr>)$, $lp \in \mathcal{L}(<loop\ boundary>)$ and $name \in$*

$\mathcal{L}(<msc\ name>)$ then $S_{<msc\ ref\ exp>}$ is defined as

$$
\begin{array}{ll}
S_{<msc\ ref\ expr>}[\![a_1]\!] & = S_{<msc\ ref\ par\ expr>}[\![a_1]\!] \\
S_{<msc\ ref\ expr>}[\![a_1\ \textbf{alt}\ a_2]\!] & = S_{<msc\ ref\ par\ expr>}[\![a_1]\!]\ []\ \\
 & \quad S_{<msc\ ref\ par\ expr>}[\![a_2]\!] \\
S_{<msc\ ref\ par\ expr>}[\![p_1]\!] & = S_{<msc\ ref\ seq\ expr>}[\![p_1]\!] \\
S_{<msc\ ref\ par\ expr>}[\![p_1\ \textbf{par}\ p_2]\!] & = S_{<msc\ ref\ seq\ expr>}[\![p_1]\!]\ \|\ \\
 & \quad S_{<msc\ ref\ seq\ expr>}[\![p_2]\!] \\
S_{<msc\ ref\ seq\ expr>}[\![s_1]\!] & = S_{<msc\ ref\ loop\ expr>}[\![s_1]\!] \\
S_{<msc\ ref\ seq\ expr>}[\![s_1\ \textbf{seq}\ s_2]\!] & = S_{<msc\ ref\ loop\ expr>}[\![s_1]\!]\ ;\ \\
 & \quad S_{<msc\ ref\ loop\ expr>}[\![s_2]\!]
\end{array}
$$

$$S_{<msc\ ref\ loop\ expr>}[\![\textbf{loop}\ <n,m>\ exp]\!] = N_{loop}^{<n,m>}[B \leftarrow S_{<msc\ ref\ expr>}[\![exp]\!]]$$

$$S_{<msc\ ref\ loop\ expr>}[\![name]\!] \quad = S_{<node>}[\![name]\!]$$

where $S_{<msc\ ref\ par\ expr>}$, $S_{<msc\ ref\ seq\ expr>}$ and $S_{<msc\ ref\ loop\ expr>}$ are semantic functions for the translation of the respective reference operations, i. e. **par**, **seq** *and* **loop**.

The individual stages of an HMSC translation can be soon in figures 3 to 5. Figure 4 shows the M–net after translating the *par* expression; figure 5 is the M–net after resolving the *loop* expression.

3.3 Basic Message Sequence Charts

As mentioned above an BMSC–reference is translated like a procedure call. On the other hand an BMSC resembles again the body of a procedure.

Definition 3.5 *Let $p \in \mathcal{L}(<bmsc>)$ then $S_{<bmsc>}[\![p]\!]$ is defined as*

$$S_{<bmsc>}[\![p]\!] = N_{MSCDecl}[B \leftarrow S_{<msc\ body>}[\![p]\!]]$$

Definition 3.6 *Let* $p \in \mathcal{L}(<msc\ body>)$, $i \in \mathcal{L}(<inst\ def>)$, $j \in \mathcal{L}(<msc\ ref\ expr>)$ *and* $k \in \mathcal{L}(<inline\ expr>)$ *then* $S_{<msc\ body>}$ *is defined as*

where $S_{<msc\ body>}$ *is the semantic function for an MSC body.*

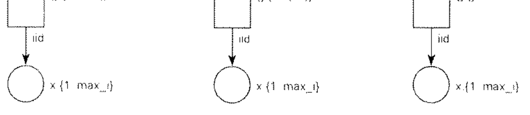
The M–net N_{MSCDed} is defined in the middle part of figure 6. A new instance of the referenced BMSC is created or destroyed through the action names M_c and M_r once the respective reference is entered. In order to achieve reusability of instance identifiers a stack similar mechanism consisting of places $p2$ and $p1$ is modeled. Once an MSC reference is entered by synchronizing with *Mc* the next free position is taken from the pool $p2$ of free stack positions and the number of currently existing instances in $p1$ is incremented. If an BMSC has finished, identifier is put back onto $p2$ by synchronizing with *Mr*. and $p1$ is decremented.

The semantics of a body of a Basic Message Sequence Chart is obtained by placing all process instances in parallel together with all MSC references and inlines. Process instances communicate with each other by exchanging asyn
chronous messages.

$$
\begin{aligned}
& S_{<msc\ body>}[\![p]\!] = \\
& \quad [Ref \cup Inl : (\|_{i \in AllInst(p)}\ S_{<inst\ def>}[\![i]\!])\ tie\ AllMsg(p) \\
& \qquad \| \ (\|_{j \in AllRef(p)}\ S_{<msc\ reference>}[\![j]\!]) \\
& \qquad \| \ (\|_{k \in AllInl(p)}\ S_{<inline\ expr>}[\![k]\!])]
\end{aligned}
$$

where AllInst(p) is the set of all instance definitions in p. AllRef(p) is the set of all MSC references in p and AllInl(p) is the set of allinlines in p; analogously

$S_{<inst\ def>}$ *,* $S_{<msc\ reference>}$ *and* $S_{<inline\ expr>}$ *are the semantic functions for one instance, reference and inline definition. AllMsg(p) is the set of all*



$$\begin{aligned} S_{<inst\ def>}[\![e]\!] &= S_{<event>}[\![e]\!] \\ S_{<inst\ def>}[\![e\ r]\!] &= S_{<event>}[\![e]\!]; S_{<inst\ def>}[\![r]\!] \end{aligned}$$

Figure 7: From the left: N_m^+, N_m^- and N_{action}

message and timer names in p. Further, it applies to the sets Ref and Inl that

Ref = $\{\overline{i_r^{call}} \mid i \in AllInstNames(p) \wedge r \in AllRefNames(p)\}$ *and* *Inl* = $\{\overline{i_r^{call}} \mid i \in AllInstNames(p) \wedge r \in AllInlNames(p)\}$

Tile precise meaning of functions $S_{<msc\ reference>}$ and $S_{<\ inline\ expr>}$ is explained by means of an example in section 3.7.

3.4 Instance

The semantics of an instance is a sequence of a number of events which occur from the top to the bottom along the instance. The order in which they appear ill the textual representation is the order in which they are to be executed. We assume that each message event has a unique name.

Definition 3.7 Let e$\in$ L (< event>) and r $\in$ L(< inst def >) then $S_{<\ inst\ def>}$ is defined as

$$S_{<insedef>}\|e\| = S_{<event>}\|e\|$$

$$S_{<insedef>}\|\ e\ r\ \| = S_{<event>}\|e\|\,; S_{<event>}\|r\|$$

where $S_{<event>}$ is the semantic function for instance events.

Definition 3.8 *Let $bl \in \mathcal{L}(<event\ name\ list>)$, $en \in \mathcal{L}(<event\ name>)$, $m \in \mathcal{L}(<msg\ name>)$ and $i \in \mathcal{L}(<instance\ name>)$*

$$\begin{aligned} S_{<event>}[\![en\ \textbf{out}\ m\ \textbf{to}\ i\ bl]\!] &= N_{m^+} \\ S_{<event>}[\![en\ \textbf{in}\ m\ \textbf{from}\ i\ bl]\!] &= N_{m^-} \\ S_{<event>}[\![\textbf{action}\ at]\!] &= N_{action} \end{aligned}$$

3.5 Communication, Action and Timer

Asynchronous communication is modeled by the application of the tie–operation in definition 3.6. Each send event of message *m* is translated as an M–net N_{m+} in figure 7; each receive event as an M–net N_{m-}.

Definition 3.8 Let *bl* $\in$ *L (<event name list>), en* $\in$ *L (<event name>), m* $\in$ *L (<msg name>), i* $\in$ *L (<event name>)*

$$\begin{aligned} S_{<event>}[\![en \textbf{ out } m \textbf{ to } i\ bl]\!] &= N_{m^+} \\ S_{<event>}[\![en \textbf{ in } m \textbf{ from } i\ bl]\!] &= N_{m^-} \\ S_{<event>}[\![\textbf{action } at]\!] &= N_{action} \end{aligned}$$

where both the sets S and R are defined as $S = \{ L^{+}_{en}(BeforeList(bl) \mid \Theta iid \}$ *and* $R = \{ L^{-}_{name}(iid) \mid name \in IsBefore(m) \}$*. Funcions BeforeList(bl), IsBefore(m)* $\subset$ *L (<event name>) define the set of event names whose meaning is explained in definition 3.9.*

The semantics of a timer *T* i.e. the setting of a timer set_T , a subsequent reset $reset_T$ or timeout $timeout_T$ are treated in the same way. We assume that for each timer setting in an BMSC or inline there is a corresponding timeout or reset event. All timers have unique names.

3.6 Coregion and General Ordering

The purpose of a coregion is to specify unordered events on a process instance that, is within a coregion all send or receive events may occur in any order even parallel. Only send and receive events are allowed inside a coregion. According to this explanation the semantics of a coregion is the parallel composition of all the events defined within the coregion.

Definition 3.9 *Let cr* $\in$ *Lf(<coregion>)*

$$S_{<event>}[\![cr]\!] = (\|_{e \in CoEvents(cr)} S_{<event>}[\![e]\!]) \ tie \ Ordering(cr)$$

where CoEvents(cr) is the set of all message events inside cr and Ordering(cr) is the set of all message event names which have a before–clause assigned to it.

In definition 3.8 there are the sets *S* and *R* defined. Their meaning and that of definition 3.9 are explained in figure 8 and 9. In the upper part of figure 9 there is the M–net corresponding to the coregion on the *AS_System* process instance bevor the tie–operation is applied. The receive event *wheel* has a before–list $bl = \{L_{brake}, L_{\text{angle}}\}$ attached to it, i.e. this message event has to occur before any of the events in *bl*. The same effect is achieved in the M–net through the link symbol $S = \{Lwheel^{+(}((x\,\Theta\, iid))$ with $x = |BeforeList(bl)|$, i.e. *Lwheel* puts as many instance identifier tokens onto the place S_{Lwheel} as the cardinality of the *BeforeList* set. Θ is used to relate an element of a multi–set *A* to the number of its occurrences in a multi–set over A. On the other side all message events which appear in any *BeforeList* are assigned a link symbol L^{-}_{name} (iid).

In the MSC of figure 8 this corresponds to the arrows inside the coregion. This is what is called *general ordering* since causal ordering among otherwise causal unrelated events of a coregion can be enforced. It applies to the process instance *AS_System* that *Wheel* occurs before *Brake* and *Angle*, but *Brake* and *Angle* are unrelated among each other (analogously for *Accel* and *Strength*).

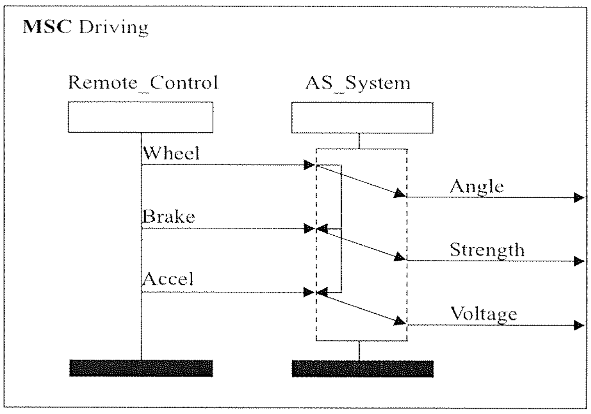


Figure 8: *MSC Driving* with a coregion

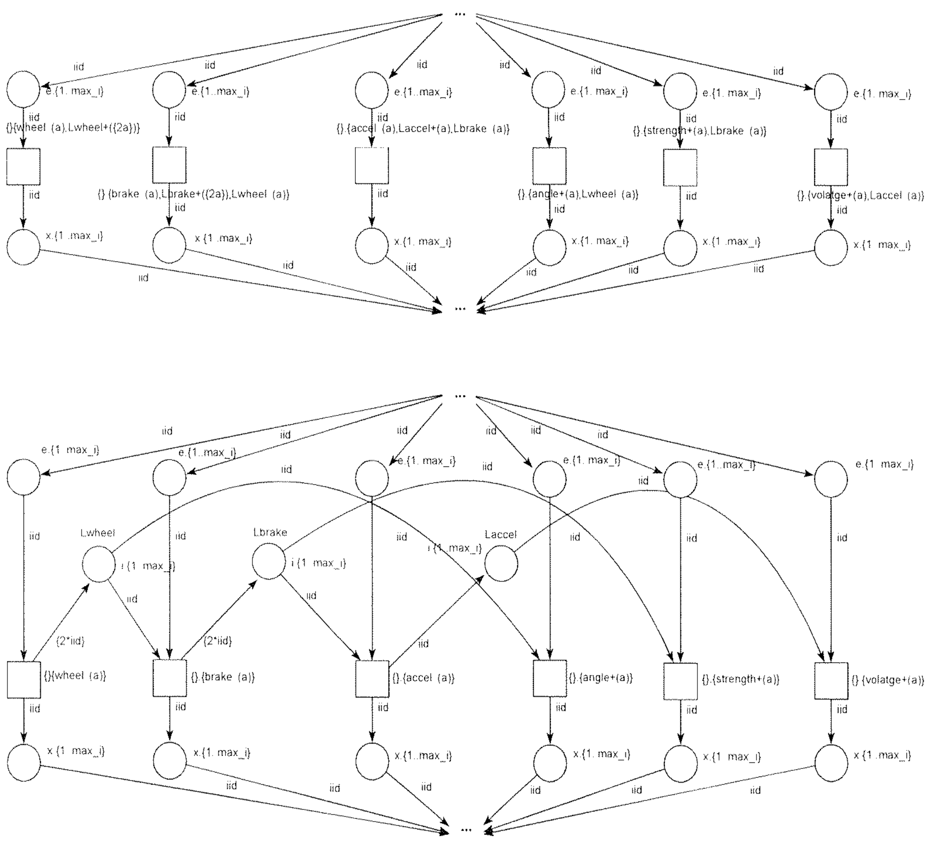

...
iid
e.{1..max_i}
{}{wheel (a),Lwheel+({2a})}
{}.{brake (a),Lbrake+({2a}),Lwheel (a)}
{}.{accel (a),Laccel+(a),Lbrake (a)}
{}.{angle+(a),Lwheel (a)}
{}.{strength+(a),Lbrake (a)}
{}.{volatge+(a),Laccel (a)}
x.{1..max_i}
Lwheel
Lbrake
Laccel
i.{1..max_i}
{2*iid}
{}{wheel (a)}
{}.{brake (a)}
{}.{accel (a)}
{}.{angle+(a)}
{}.{strength+(a)}
{}.{volatge+(a)}

Figure 9: From top: coregion before and after the application of the tie–operation

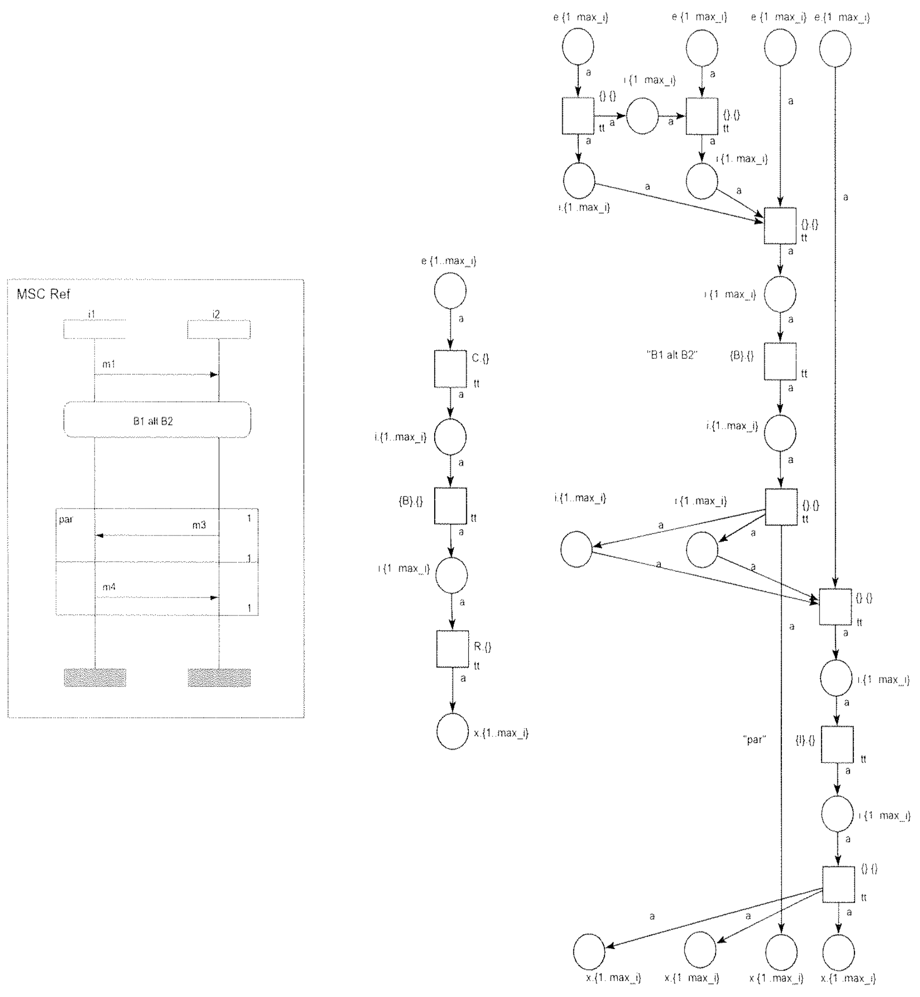


Figure 10: From the left: *MSC Ref*, N_{mul} and M–net skeleton for *MSC Ref*

3.7 Inline and BMSC–reference

for reasons of a more comfortable modeling MSC'96 is enriched with so–called *inlines* by which means composition of small and clear event structures may be defined. They serve as a substitute for otherwise small sized BMSC–references. In figure 10 there is an BMSC example with one BMSC–reference expression and inline operator expression. They both are translated in the same way that is all BMSC–reference and an inline operator expression become processes running concurrent side by side with the BMSC body. They have the form of N_{mul} in figure 10.

Definition 3.10 Let i $\in L$ (<*msc ref expr*>), $n \in L$ (<inline expr name>) and iname $\in L$ (<name>)

$$
\begin{aligned}
S_{<event>}[\![iname\ \textbf{reference}\ n\ i]\!] &= N_{mref} \\
S_{<event>}[\![iname\ \textbf{par begin}\ n]\!] &= N_{mref} \\
S_{<event>}[\![iname\ \textbf{loop begin}\ n]\!] &= N_{mref} \\
S_{<event>}[\![iname\ \textbf{alt begin}\ n]\!] &= N_{mref}
\end{aligned}
$$

where N_{mref} structuraly equals to N_{ref} in figure 6 except for the action symbols Mc(iid) and Mr(iid) which are replaced by replaced by $iname_n^{call}$ and $iname_n^{ret}$ respectivey.

The individual process instances synchronize mutually, similar to a procedure call, all entry into the MSC reference or inline expression so that they simultaneously enter the body. The same comes about while termination.

Definition 3.11 Let p ∈ L (<msc ref expr>) the $S_{<\text{msc reference}>}$ is defined as

$$S_{<msc\ reference>}[\![p]\!] = N_{mul}[B \leftarrow S_{<msc\ ref\ expr>}[\![p]\!]]$$

where $S_{<\text{msc ref' expr}>}$ is defined in section 3.2.

The M–net N_{mul} synchronizes with the corresponding process instances via the reference or inline names. For this purpose N_{mul} is assigned the action set

$$C = \{\overline{i1^{call}_{(B1altB2)}}, \overline{i2^{call}_{(B1altB2)}}\}$$

and

$$R = \{\overline{i1^{ret}_{(B1altB2)}}, \overline{i2^{ret}_{(B1altB2)}}\}$$

The transition in the middle of the sequence labeled with B is refined with the contents of MSC reference or inline expression. This can be similar to an HMSC node, a simple BMSC reference or a reference expression containing **alt, par** or ***seq*** combined sub–expressions.

Definition 3.12 *Let* $n, m \in \mathbb{N}$, $ba \in \mathcal{L}(<msc\ body> \{\textbf{alt}\ <msc\ body>\}^*)$, *further* $bp \in \mathcal{L}(<msc\ body> \{\textbf{par}\ <msc\ body>\}^*)$, $b \in \mathcal{L}(<msc\ body>)$

$$S_{<inline\ expr>}[\![\textbf{loop} < n, m > \textbf{begin}\ b\ \textbf{loop end}]\!] = N_{loop}^{<n,m>}[B \leftarrow S_{<msc\ body>}[\![b]\!]]$$

$$S_{<inline\ expr>}[\![\textbf{alt begin}\ ba\ \textbf{alt end}]\!] = []_{i \in AllAlt(ba)}\ S_{<msc\ body>}[\![i]\!]$$

$$S_{<inline\ expr>}[\![\textbf{par begin}\ bp\ \textbf{par end}]\!] = \|_{i \in AllPar(bp)}\ S_{<msc\ body>}[\![i]\!]$$

where AllAlt(ba) is the set of all alternative event definitions and AllPar(bp) is the set of all parallel event definitions.

This definition has a similar structure as definition 3.4.

4 Comparison to other Work

There has already been a few works attempting to give a formal semantics to MSCs e.g. [LL93, MR94, Klu97, IT96]. The main determination feature of all these works is less the semantic domain but much more the MSC model elements considered. The older works are concerned with the translation of BMSCs only, the recent one [Klu97] with HMSCs. Both [MR94, KIu97] and annex B of the MSC'96 standard [IT96] translate into a process algebra.

In our work we use a. high–level colored Petri net class— M–nets [BFF$^+$95]. We fully utilize both features of M–nets: On the one hand it is their algebraic structure and on the other hand the color of tokens. The former allows us a relatively simple translation of both BMSCs and HMSCs, latter allows us to keep the net size small. In contrast to other works we have defined translation for MSC reference, inline operation expressions, coregion and general ordering. We have achieved similar results for the translation of HMSCs comparing to [Klu97] especially both for simple MSC references, MSC reference expresions and parallel frames by one–step application of the refinement operation.

5 Conclusion

We have showed in this article how to translate MSCs into an executable formal notation based on M–nets. BMSCs are translated by the composition of small and easy to understand M–nets to larger M–nets by the application of some few operations. Due to the possibility of drawing any arbitrary structured HMSCs a compositional approach is not feasible here. Instead of this, we assign directly to any HMSC a structure similar M–net whose nodes are refined with subnets in the next step. In order to keep the net size small each referenced BMSC is translated only once. Different instances of one and the same BMSC are distinguished from each other by different integer identifiers. Necessity of a colored high–level Petri net class results out of this.

An extension of our approach with temporal restrictions according to [Hau] is straightforward. Timing restrictions can be assigned to messages, a whole scenario or pairs of events on a process instance. Restrictions have the form of all interval [sft, lft]. Such intervals specify the lower and upper bounds on the delay of message delivery, execution duration of whole MSCs or the delay between two consecutive events. Timing information can be used to deduce additionall causal information or to rule out possible system traces. Since communication is asynchronous timing restrictions are related always to two events, i.e. two transitions in the M–net. However, most Petri net classes assign timing information only to one net element class that is to transitions, arcs or tokens. Two possibilities result from it: Either the restrictions can be added to the respective net as a set of linear inequalities or the restrictions are split up onto two net elements. In the former case one gets a new Petri net class, i.e. a new theoretical framework is necessary in order to be able to combine it with other net classes. In the latter case there is a risk of inconsistent MSC specification [AHP96] arises. However, this can be analyzed and therefore avoided.

References

[AHP96] R. Alur, D. J. Holzmann, and D. Peled. An analyzer for message sequence charts. In *Alle Angaben “Uberpr” ufen*, number 1155 in Lecture Notes in Computer Memo pages 35 48. Springer Verlag, 1996.

[BDH92] E. Best, R. Devillers, and J. G. Half. The box calculus: A new causal algebra with multi–label communication. In *Advances in Petri Nets*, number 609 in Lecture Notes in Computer Science, pages 21–69. 1992.

[BFF+95] E. Best, F. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel. and E. Pelz. A class of composable high level petri nets. In *Proc. of ATPN’95*, number 935 in Lecture Notes in Computer Science, 1995.

[DK95] R. Devillers and H. Klaudel. Refinement and recursion in high level petri box calculus. In *Structures in Concurrency Theory*, number "Uberpr”ufen in Lecture Notes in Computer Science, page’s 144 159. Springer Wag, 1995.

[FG97] Hans Fleischhack and Bernd Grahlmann. A Petri Net Semantics for B(PN)2 with Procedures. In P*roceedings of PDSE’97 (Parallel and Distributed Software*

Engineering), Boston MA, pages 15–27. IEEE Computer Society, May 1997.

[FG98] Hails Fleischhack and Bernd Grahlmann. A Compositional Petri Net Semantics for SDI. In *Proceedings of ATPN'98* (*Application and Theory of Petrii Nets*), number 1420 in Lecture Notes in Computer Science. pages 144–161. Springer–Verlag, June 1998.

[Gra97] B. Grahlmann. The PEP Tool. In *Organ Gramberg, editor, Proceedings of CAV'97* (*Computer Aided Verification*), volume 1254 of *Lecture Notes in Computer Science*, pages 440–443. Springer–Verlag, .June 1997.

[Hau] Øystein Haugen. Msc–2000 highlights. http://www.sdl–forum.org/–MSC2000/MSC2000_files/frame.htm.

[IT96] ITU–T. Message Sequence Chart (MSC). Series Z: Programming Languages, International Telecommunication Union, October 1996.

[Klu97] O. Kluge. A normal form for composition in msc'96. Master's thesis, Friedrich–Alexander Unicersität Erlangen–Nürnberg, September 1997.

[KP99] H. Klaudel and F. Pommereau. Asynchronous links in the pbc and M–nets. In *Advances in Computing Science* ASIAN'99, number 1742 in Lecture Notes in Computer Science, pages 190–200, 1999.

[LL93] P. B. Ladkin and S. Leue. On the semantics of message sequence charts. In *Formale Methoden f"ur Verteilte Systeme*. Hartmuth K"onig, 1993.

[MR.94] S. Mau and M. A. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal,* 37(4), 1994.