Towards a Petri Net Based Semantics Definition for Message Sequence Charts

Peter Graubmann
Siemens AG ZFE BT SE 4, Otto-Hahn-Ring 6, D-8000 München 83
gr@ztivax.zfe.siemens.de

Ekkart Rudolph
Siemens AG ZFE BT SE 5, Otto-Hahn-Ring 6, D-8000 München 83
rudolph@ztivax.zfe.siemens.de

Jens Grabowski
Universität Bern, Länggassstraße 51, CH-3012 Bern
grabowsk@iam.unibe.ch

Abstract: Within the study period 1988 - 1992 of the CCITT the work on Message Sequence Chart (MSC) within the CCITT Study Group X focussed on the definition of the graphical and textual syntax of MSC [3] and on the development of a corresponding informal semantics definition to explain the meaning of the standardized MSC constructs. Within Study Group X it has been decided that the new study period should be used to provide a formal semantics definition for the existing MSC recommendation. The discussion about a suitable model for MSC semantics started at a CCITT interims meeting in Geneva, November 1992. The semantics models for MSC discussed at this meeting are based on Process Algebra, Automata Theory, and Petri Nets. In this paper the Petri Net based approach is presented.

1 Introduction

Since the new MSC recommendation Z.120 [3] has been approved quite recently (May 1992) only an informal semantics description has been developed corresponding to the syntax definition. For noncritical situations this may be often sufficient. However, experience with related specification languages can give hints about the usefulness and even necessity of a formal semantics definition.

For formal description techniques, a formal semantics is defined since they are bound to describe the functional properties (i.e. the behaviour) of a communication system without any ambiguities. In particular, the formal semantics definition serves as the basis for the development of simulation, analysis, or validation tools. Additionally, it ensures an unambiguous communication between partners. Therefore, for complex specification languages like SDL, Estelle, or LOTOS, a formal semantics definition is necessary.

In contrast, for TTCN ('Tree and Tabular Combined Notation', a standardized test case description language of the ISO/IEC [10]) no formal semantics is defined, yet the TTCN standard is claimed to have an unambiguous meaning.

For MSCs the situation lies somehow in-between. In many cases, they are used simply as informal illustrations and these applications do not require a formalization. Furthermore, the current MSC recommendation only includes a few and mainly simple language constructs and some people argue that these constructs can be understood easily and therefore no formal semantics would be necessary. But this is only part of the story. Different variants of MSCs, such as Arrow Diagrams, Time Sequence Diagrams, Interworkings, frequently are used at universities, standardization bodies, and within companies [8]. Depending on the application areas, they assume different communication mechanisms and include different language constructs any exchange between organizations or tools proves to be problematic. Besides the standardized syntax definition [3], a formal semantics definition may be helpful to harmonize the use of MSCs. Beyond that, it is no longer true that the MSC recommendation only includes simple constructs. This may be substantiated by means of three examples:

- At an experts meeting in Munich, March 1992, it was recognized that the submsc-construct may lead to ambiguities since it is not clear whether the refinement of a decomposed instance axis should maintain the message ordering along the refined axis, or whether the message ordering along the refined instance axis should only be possible within the submsc.
- At the last CCITT interim meeting in Geneva, November 1992, problems concerning the interpretation of Conditions were discussed. Some people argued that a Condition can be interpreted as a synchronization point, whereas other denied this.
- At the same meeting a high level construct for the description of more general time orderings along an instance axis was required.

In order to clarify and unambiguously define the meaning of such constructs, a formal semantics definition for MSC is inevitable.

Since the MSC recommendation Z.120 has been produced just recently it seems to be the right point of time for the development of a formal semantics. Further extensions of Z.120 may then evolve from this sound basis.

This paper is organized in the following way: Within the second chapter, we give a brief rationale for our selection of Petri nets as a basis for the MSC semantics and relate it to two further approaches which came up recently. In chapter 3, a special class of Petri Nets, the so-called labelled occurrence nets, is introduced. Within the fourth, fifth and sixth chapter, the translation of MSCs into labelled occurrence nets is defined. In the seventh chapter, a semantics for MSC composition and decomposition is provided. Finally, an outlook is given.

2 Comparison with other approaches

Until now, two further approaches towards a formal MSC semantics have been proposed in addition to the Petri Net based semantics definition described in this paper. They can be considered the starting points from which the MSC semantics discussion within CCITT SG X will proceed. For completeness and comparison also the other approaches shall be sketched briefly in the following.

One approach which has been worked out at University of Berne uses an interleaving model and is based on finite automata [6,12]. Formally, a single MSC can be interpreted as a graph with two sorts of edges. The nodes represent communication events, e.g. message sending and message consumption. This graph can be interpreted as a global state transition graph, containing all possible global states specified by the MSC. It corresponds to an automaton without explicitly defined end states. Unfortunately, with MSC composition problems may occur with unique end-state determination. To find proper end states a termination criterion from ω-automata theory, due to Büchi [14], is used. The main advantage of the sketched semantics approach and the hereupon based MSC semantics is its flexibility. According to the chosen set of end states it is possible to analyse MSCs under various points of view, e.g. with respect to safety and liveness properties of the system.

A second approach again applies an interleaving model and is based on a process algebra [13]. The semantics is used to process, analyse, and combine so called Interworkings which are closely related to MSCs. Within an Interworking a message interaction between two entities can be split into two different events: output and input of the message. But contrary to MSCs, within Interworkings communication is meant to be synchronous. A formal semantics for Interworkings is defined with the use of the algebraic concurrency theory Basic Process Algebra. To that end, it is extended by two operators: merging and sequencing of Interworkings.

The approach towards a formal MSC semantics chosen in this paper is based on Petri net theory [5,9]. As the idea of partial ordering of signaling and instance events was one of the leading principles during the definition of MSCs, occurrence nets - which are the Petri net way of presenting partial orderings - seem to be particularly well suited for a basis of MSC semantics. Within this framework in particular a clear semantics may be attached to composition rules for MSCs.

In comparison with the other semantics definitions the correspondence between syntax definition and formal semantics description seems to be more evident in the Petri net approach. In particular, occurrence nets immediately reflect the partial ordering of events since the relations co (concurrency) and li (line, i.e. sequentiality) may be derived directly from the net structure. By employing occurrence nets for a semantics description of MSCs the well elaborated mathematical theory within net theory can be utilized.

3 Occurrence nets

As basis of our considerations we use occurrence nets. In fact, for the semantics we are going to provide, we aim at something similar to Petri net processes. Yet, in our case, there is no original Petri net from which a Petri net processes can be assumed to have evolved, and thus, we are forced to formulate the essentials of the Petri net processes without relying on such a related net.

Occurrence nets [1,2] are cycle free and conflict free nets which therefore suitably describe individual traces within distributed systems. As mentioned above they evidently present the partial ordering of events being contained.

Definition:
An occurrence net is a net (B, E, F) with

(1) arbitrary elements, places B and the transitions E with $B \cap E = \emptyset$ and $B \cup E \neq \emptyset$,

(2) a flow relation F with $F \subseteq (B \times E) \cup (E \times B)$,

such that:

(3) no place has more than one predecessor and one successor in F
$(\forall\, b \in B \,.\, \mathrm{card}(\bullet b) \leq 1 \wedge \mathrm{card}(b \bullet) \leq 1)$,

(4) the flow relation F is cycle free
$(\forall\, x,y \in B \cup E \,.\, (x,y) \in F^+ \Rightarrow (y,x) \notin F^+)$.

A labelled occurrence net is obtained from an occurrence net by assigning a labelling function to it.

The labeling is a carefully arranged string (cf. the conversion schemes) and it is used to relate the elements of the occurrence net to instance axis and the MSC elements thereon. Further it serves to identify which places of the occurrence net correspond to which messages and timer constructs in the MSC. Thus the labeling establishes the actual semantic link.

Definition:
A labeled occurrence net (B, E, F; λ) is an occurrence net (B, E, F) with labeling function $\lambda : B \cup E \rightarrow$ LABEL.

Thereby, LABEL is an arbitrary set of labels. Indeed, we consider LABEL as the set of arbitrary strings. During occurrence net composition, labeling strings are analysed and indicate thus which occurrence net elements are to be identified.

We further need the notion of initial and final elements of a (labeled) occurrence net: Final elements of an occurrence net are the elements with no output place or transition, initial elements are those with no input places or transitions. Thus we define for the occurrence net (B, E, F):

Definition:
The element $x \in B \cup E$ is an initial element of (B, E, F) iff $\bullet x = \emptyset$. It is a final element of (B, E, F) iff $x \bullet = \emptyset$.

4 MSC nodes: the basis for the mapping of MSCs onto Occurrence Nets

Any MSC document merely contains an arbitrary collection of MSCs. Our considerations with respect to formal semantics therefore rather start on the level of MSCs contained in one MSC document. In order to describe appropriately the formal mapping of MSCs onto occurrence nets we have to identify further structures within MSCs and their instances. At this, the first step is to consider an MSC as a collection of event structures that are represented by the MSC's instance axes:

(A) $\mathrm{msc}_{\text{message sequence chart name}} = \{\, \mathrm{instance}_{\text{instance name 1}}, \ldots, \mathrm{instance}_{\text{instance name n}} \,\}$

This view of MSCs is based on the alphanumeric definition of MSCs by the concrete textual grammar [3] (we refer the MSC syntax as condensed as possible to keep arguments brief):

<message sequence chart> ::=

  msc <message sequence chart name> ... <instance definition>* **endmsc** <end>

<instance definition> ::= **instance** <instance name> ... <instance body> **endinstance** <end>

The iteration of <instance definition> produces the various numbered event structures ($\text{instance}_{\text{instance name } i}$) in equation (A). The rendering of the MSC syntax definition as given here excludes the <global condition> which we do not tackle in this paper. Reason for this is that the construct itself is up to now under controversial discussion and requires a somewhat clumsy albeit not problematic semantics definitions.

Each instance axis, or better, the event structure represented by it, is viewed as a list of consecutively numbered nodes,

(B) $\text{instance}_{\text{instance name}} = (\text{node}_{\text{instance name}, 0}, \ldots, \text{node}_{\text{instance name}, m}),$

which again are basically drawn from the MSC grammar:

<instance definition> ::=

  instance <instance name> ... [**decomposed**] <end> <instance event list> [<stop>]

  endinstance <end>

<instance event list> ::=

  { <message input> | <message output> | <create> | <set> | <re-set> | <timeout> |

  <coregion> | <action> | <condition> }*

The linear numbering scheme of the nodes reflects the order in which nodes are written down in a concrete instance definition. First node is the Instance-init-node, last one is the Instance-end-node.

Additionally to the instance events, a node is needed to express initialization of an instance (this node corresponds to the instance definition) and another to indicate the stop event. Thus, we derive a list of nodes which again are related to certain expressions of the concrete textual grammar. Node parameters collect the information specific to the instance event, i.e. names or lists of names defined by the syntactical representation of the instance event. Each node has the name of its instance axis as a first parameter. In the following MSC grammar expressions, references to other instance axes are preceded by qualifiers set in *italics*.

(1) Instance-init-node (instance name)

  maps onto the grammatical expression "**instance** <instance name>" beginning the <instance definition>

(2) Message-input-node (instance name, message identification, sender instance name)

  <message input> ::=

    in <message identification> **from** { <*sender* instance name> | **env** } <end>

(3) Message-output-node (instance name, message identification, receiver instance name)

  <message output> ::=

    out <message identification> **to** { <*receiver* instance name> | **env** } <end>

(4) Create-node (instance name, created instance name)

  <create> ::= **create** <*created* instance name> ... <end>

(5) Set-node (instance name, timer name, timer instance name)

<set> ::= **set** <timer name> [, <timer instance name>] ... <end>

(6) Reset-node (instance name, timer name, timer instance name)
<reset> ::= **reset** <timer name> [, <timer instance name>] <end>

(7) Timeout-node (instance name, timer name, timer instance name)
<timeout> ::= **timeout** <timer name> [, <timer instance name>] <end>

(8) Coregion-node (instance name)
<coregion> ::=
concurrent { <message output> + | <message input> + } **endconcurrent** <end>

(9) Action-node (instance name)
<action> ::= **action** <action text> <end>

(10) Condition-node (instance name, condition name, shared instance list)
<condition> ::=
condition <condition name> [**shared** { <shared instance list> | **all** }] <end>

(11) Stop-node (instance name)
<create> ::= **stop** <end>

(12) Instance-end-node
maps onto the keyword "endinstance" occurring in <instance definition>. The Instance-end-node has no corresponding occurrence net representation.

Default rules for syntactic options are fixed: timer instance names are considered an empty string if not explicitly stated; in case, a message comes from or goes to the environment, receiver or sender instance name is given as "env", respectively.

So far, we systematically have described a fragmentation procedure for MSCs into certain nodes which is solely based on matching the respective syntactic MSC elements. The construction of the corresponding occurrence nets inverts this procedure (cf. section 6).

In case a submsc definition was used in one of the transformed MSC's <instance definitions>, i.e. if one of them has the form

<instance definition> ::= **instance** <instance name> ... **decomposed** <end> ... **endinstance** <end>

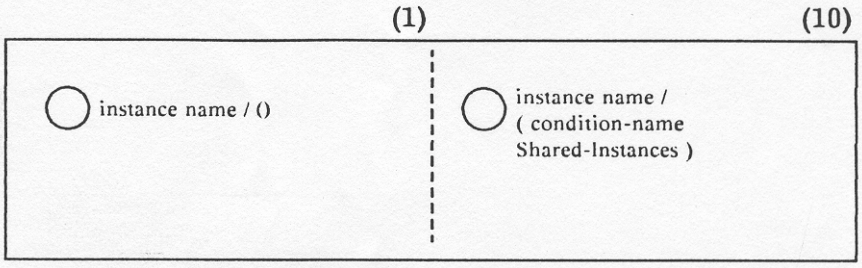
a final step becomes necessary. The MSC's occurrence net representation has to be merged specifically (see section 6) with the one, derived from the submsc. The relevant submsc is determined by its name (the submscs <message sequence chart name> is equal to <instance name> of the decomposed instance).

5 Occurrence net fragments, corresponding to the MSC nodes

A considerable part of the information contained in an instance event goes into the labeling of the occurrence net elements. The parameters of the respective nodes have collected this information. Basically, the occurrence net fragments we use here consist of a transition with one input and one output place. After the composition process, this construct establishes the "line" corresponding to the instance axis, along which the instance events are ordered. Interaction with other instance events is provided by an additional place linked with the transition (Message- and Timer-nodes). For Instance-init-node and Condition-node the construct degenerates to a simple place. The Coregion-node is a special case, for it has to provide for the causal independency of its messages and itself is the result of a composition process (see below).

Instance-init-node
(instance name)

Condition-node
(instance name,
condition name,
shared instance list)

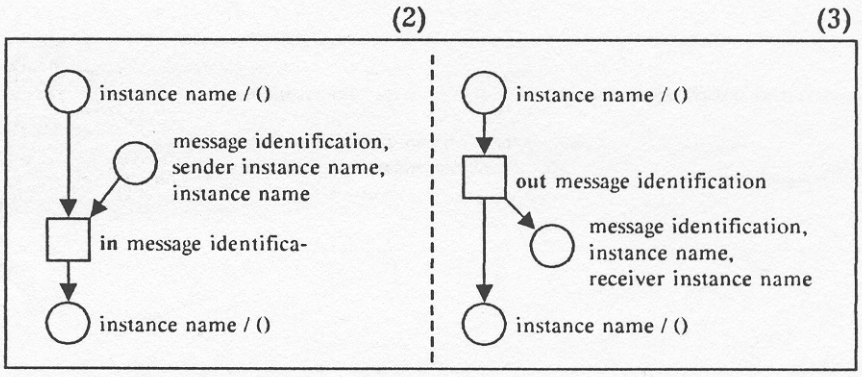


The Instance-init-node indicates the beginning of an instance axis. Like all the places referring to the "line" of an instance axis, it is labeled with the instance name. Additionally, a pair of brackets »()« is separated from the instance name by a slash »/«. »()« indicates an empty list. The Condition-node also maps to one place labeled with the instance name and, separated by the slash »/«, it follows a one element list. The only list element is the condition name together with the set "Shared-Instances", which is built from the instance names, occurring in the shared instance list. Shared-Instances is denoted as set of elements enclosed in curled brackets:

"{ *shared* instance name$_1$, .., *shared* instance name$_m$ }".

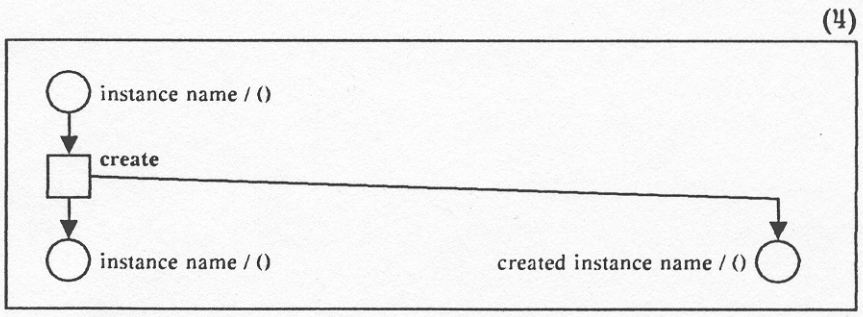
Message-input-node
(instance name,
message identification,
sender instance name)

Message-output-node
(instance name,
message identification,
receiver instance name)



The place label "message identifier, sender instance name, instance name" in the Message-input-node is identical to the place label "message identifier, instance name, receiver instance name" in the Message-output-node, if the message is exchanged between sender and receiver instance.

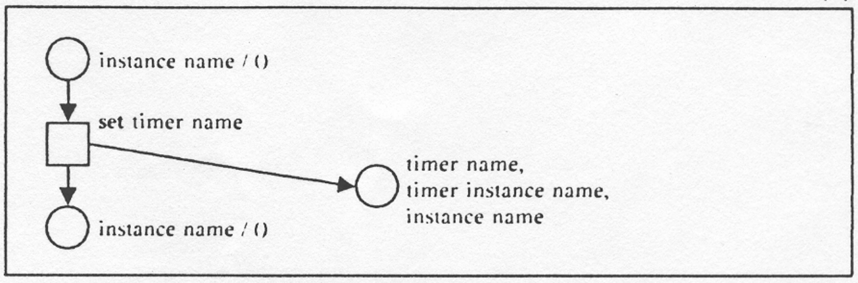
Create-node
(instance name,
created instance name)



The place labeled "created instance name" identifies with the Instance-init-node of the created instance.

(5)

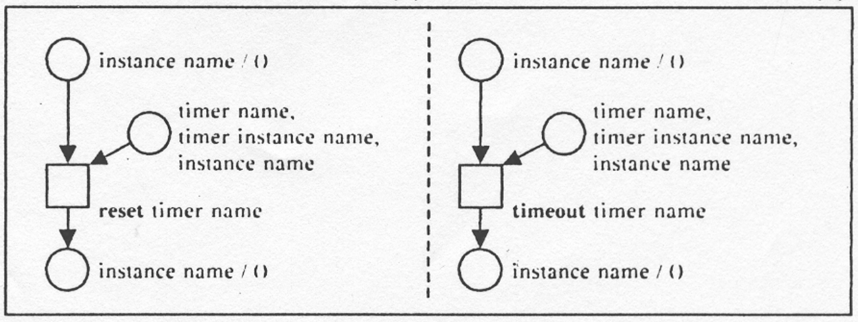
Set-node
(instance name,
timer name,
timer instance name)



(6) (7)

Reset-node
(instance name,
timer name,
timer instance name)

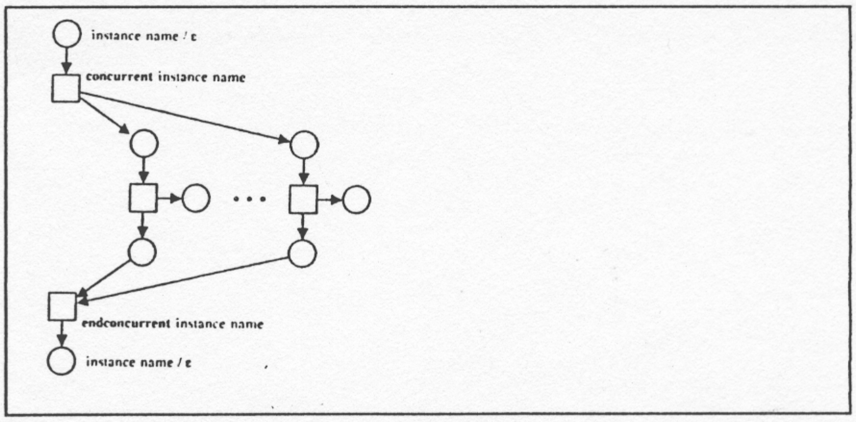
Timeout-node
(instance name,
timer name,
timer instance name)



The additional places in the Set-, Reset-, and Timeout-node of one timer instance are uniquely labeled "timer name, timer instance name, instance name". They indicate concurrency of a running timer with the events and conditions between fork (Set-node) and join (Reset- or Timeout-node).

(8)

Coregion-node
(instance name)



The Coregion-node has to be composed separately. Like the instance axis itself it is considered a sequence of nodes

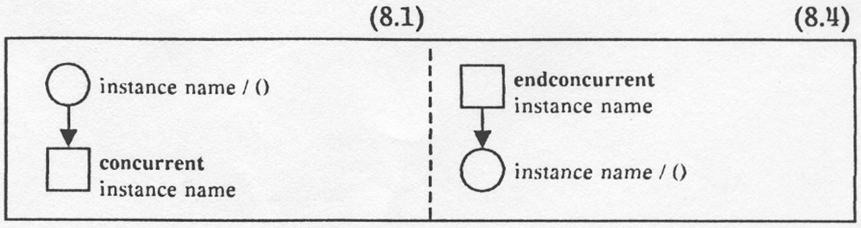
(C) Coregion-node =
(Co-init-node, Co-msg-...-node$_1$, ..., Co-msg-...-node$_k$, Co-end-node),

along which the occurrence net composition will be done.

Co-msg-input- and -output-nodes correspond to the <message input> and <message output> definitions that occur within the coregion. They contain Message-input- and -output-nodes, which are expanded by transitions to allow a fork and join respectively. Co-init- and -end-node map onto the keywords "concurrent" and "endconcurrent", respectively.

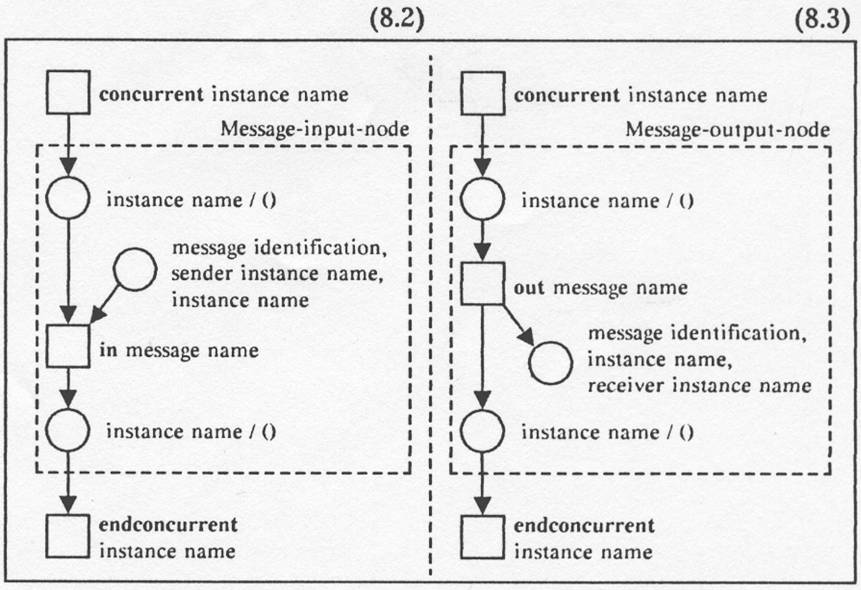
Co-init-node
(instance name)

Co-end-node
(instance name)



Co-msg-input-node
(instance name,
message identification,
sender instance name)

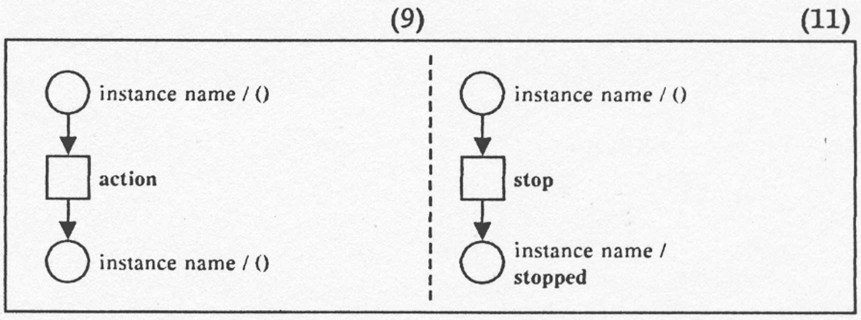
Co-msg-output-node
(instance name,
message identification,
receiver instance name)



Action-node and Stop-node do not contribute to the communication between instances. They, however, appear as separate events in the occurrence nets in order to reflect the event structure of an instance axis completely.

Action-node
(instance name)

Stop-node
(instance name)



6 Construction of the occurrence net corresponding to an MSC

The construction of the corresponding occurrence net inverts the fragmentation process, described by the equations (A) and (B) in section 4. Hence, the first step is to compose equivalent occurrence nets for the instance axes of the MSC.

Let us denote the MSC under consideration with *msc* and its corresponding occurrence net with *occ (msc)*. Each instance axis *inst* of *msc* has to be converted into a labelled occurrence net *occ (inst)*. Following the numbering scheme given in (B), we construct successively the occurrence nets *occ (inst,i)* which are meant to represent the instance axis up to its ith node.

To start with, we identify *occ (inst,0)* with the Instance-init-node (here, we do not distinguish between a node as constituent of the instance axis and its occurrence net

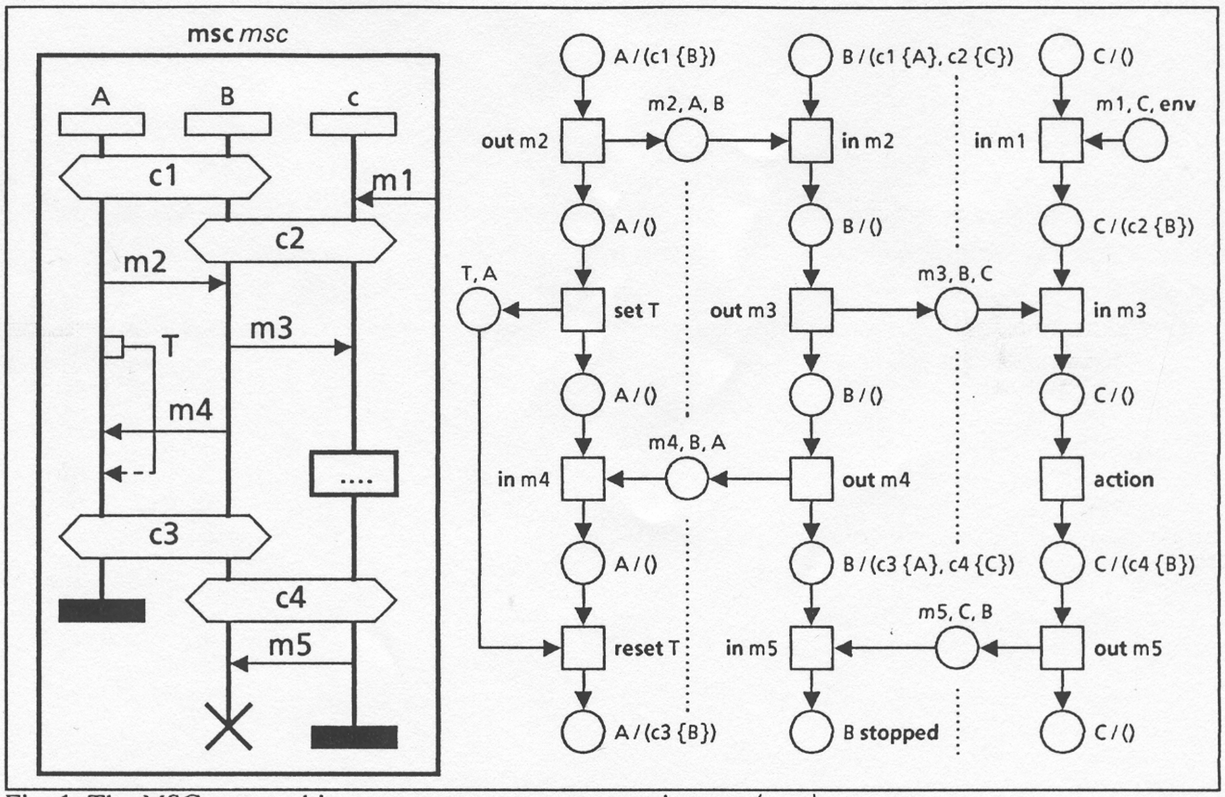


Fig. 1: The MSC *msc* and its occurrence net representation *occ (msc)*

representation). If we have already constructed *occ (inst,i)*, we get *occ (inst,i+1)* by identifying final elements of *occ (inst,i)* with initial elements of the (i+1)th node, provided they are equally labeled.

For all labels containing a slash »/«, the label is split into the "proper label" (consisting of the part before the slash) and the "associated list" (following the slash). To be 'equally labeled' means in this case for two elements, that their proper labels are identical. Identifying element e_1 with element e_2 additionally means to concatenate the associated lists, such that the label of the identified elements look like

»proper label / associated list of e_1 + associated list of e_2«.

In case, a Coregion-node has to be added, an analogous construction process is performed to yield the Coregion-node's occurrence net representation, following the coregion's presentation as list in (C).

The composition process described above terminates, if the mth node is the Instance-end-node (which is without occurrence net representation). Then, the already provided *occ (inst,m-1)* is the desired *occ (inst)*. The gained occurrence net representation *occ (inst)* of the instance axis *inst* is again a occurrence net due to the construction process.

If an instance axis *inst* is decomposed into an submsc *submsc*, then *occ (inst,m-1)* is not yet the final occurrence net representation of *inst*. The submsc representation *occ (submsc)* has to be merged with *occ (inst,m-1)*. The partial ordering of message events on *inst* is necessarily equal to the partial ordering of environmental messages in *submsc*, due to the MSC syntax [3] (i.e. *inst* and *submsc* are partial order equivalent).

The instance representation *occ (inst)* is yielded by identifying the related Message-input- and -output-nodes of both the occurrence nets:

For each initial place p_1 of *occ (inst,m-1)*, that is labeled according to the scheme "message identifier, sender instance name, receiver instance name", the corresponding initial place p_2 of *occ (submsc)* has to be labeled "message identifier, **env**, submsc instance name" (i.e. the first components match). The places are identified and the label of the place p_1 in *occ (inst,m-1)* prevails. Subsequently, the two transitions t_1 and t_2 for which the just identified places are input places are also identified ($p_1\bullet=\{t_1\}$, $p_2\bullet=\{t_2\}$).

It may happen that more than one place in *occ (submsc)* correspond with p_1. In that case, it has to be checked whether the place selected for identification does not violate the partial order equivalence of *inst* and *submsc*. It can be done in the following way: let p_1' and p_2' be two other identified places of *occ (inst,m-1)* and *occ (submsc)*, respectively. In case, p_1' and p_2' are initial places, let be t_1' and t_2' the transitions with $p_1'\bullet=\{t_1'\}$ and $p_2'\bullet=\{t_2'\}$, if they are final places, then their respective predecessors have to be chosen ($\bullet p_1'=\{t_1'\}$, $\bullet p_2'=\{t_2'\}$). Select an arbitrary path in *occ (inst,m-1)*, that contains both t_1 and t_1' and one in *occ (submsc)*, that contains t_2 and t_2': The correspondence between p_1 and p_2 is correct, if and only if the two pairs of transitions occur in same order in each path. In case, no path containing both p_1 and p_1' can be found in *occ (inst,m-1)*, there must not be a path of *occ (submsc)*, containing the other pair.

An analogous identification process is performed for final places of *occ (inst,m-1)* and *occ (submsc)*. This identification process exactly reflects that messages from and to the environment of *submsc* correspond to messages sent and received by the decomposed instance axis (for an example see fig. 2).

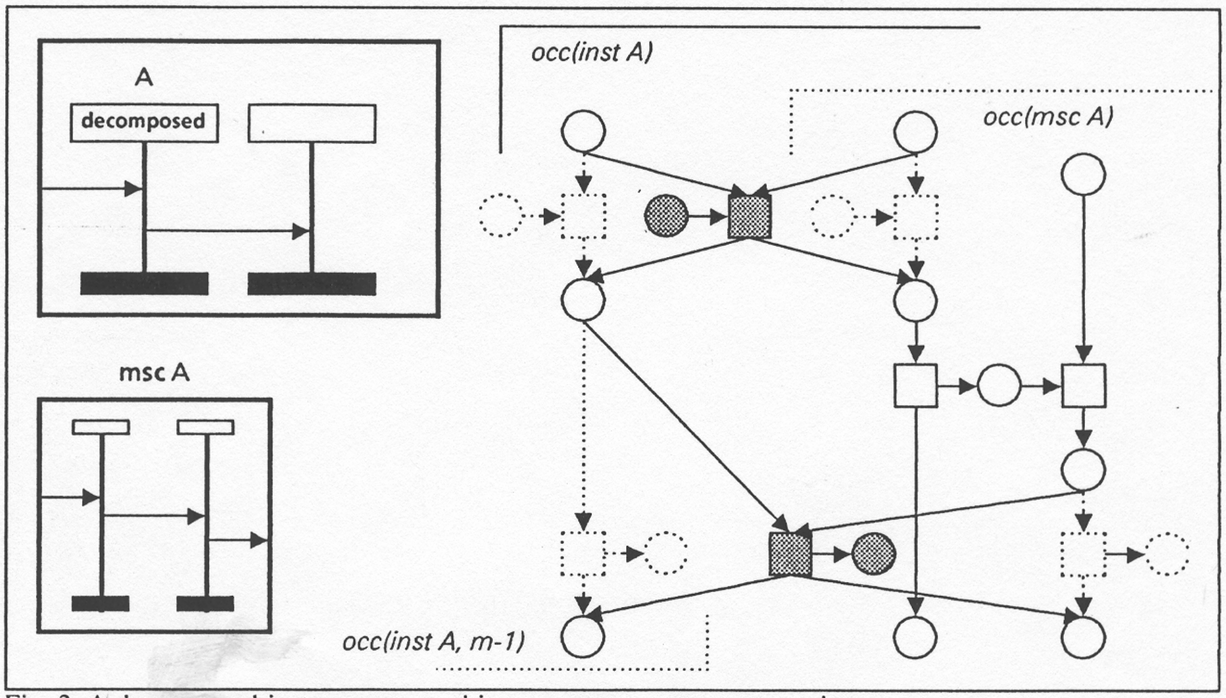


Fig. 2: A decomposed instance axes and its occurrence net representation

As second step, all occurrence nets $occ\,(inst_i)$ resulting from the transformation of the instance axes of msc have to be concatenated to yield $occ\,(msc)$. This is done by identifying each initial place of one of the $occ\,(inst_i)$ with an equally labeled final place, if it exists. In fact, by this procedure, places representing messages between instance axes are identified, as well as places, indicating instance creation with the respective Instance-init-node.

That $occ\,(msc)$ indeed is a labelled occurrence net can be proven out of the composition procedures and the MSC properties (an example is given in fig. 1).

7 Composition of occurrence nets

Two MSCs msc_a and msc_b can be composed by means of conditions, if for each instance, which both MSCs have in common, the first ends and the second begins with a corresponding condition. 'Corresponding' in this context means that both conditions refer to the same subset of instances and both conditions agree with respect to name identification.

Based on this considerations, there is the most important concatenation: the concatenation of occurrence nets that represent MSCs. This concatenation however is not, like the concatenations discussed above, a mere identification of equally labelled elements. We have also to take into account that this concatenation is governed by the matching rule for MSC conditions. We call this process 'occurrence net composition' in accordance with the notion 'MSC composition' and define it as follows: Let $occ\,(msc_a)$ and $occ\,(msc_b)$ be the two occurrence nets, representing the MSCs msc_a and msc_b. We identify a final element x_a of $occ\,(msc_a)$ with the equally labeled initial element x_b of $occ\,(msc_b)$, if - and only if - these elements represent the same condition. This is provided if the last element of the associated list in the label of x_a is identical with the first element of the associated list in the label of x_b.

Under that assumption, $occ\,(msc_a) \oplus occ\,(msc_b)$ provides the corresponding occurrence net to the MSC composition of msc_a and msc_b.

8 Conclusion

Because of their close relationship to Message Sequence Charts, occurrence nets prove to be an appropriate means for the formalization of basic MSCs. Within this framework, in particular a clear semantics may be attached to composition rules for MSCs. A generalization of transformation rules for basic constructs to cover structural concepts like coregion and submsc has been shown to be straightforward.

References

[1] E. Best, C. Fernández: Notations and Terminology on Petri Net Theory. Gesellschaft für Mathematik und Datenverarbeitung, Arbeitspapiere der GMD 195, GMD 1986

[2] W. Brauer, W. Reisig, G. Rozenberg: Advances in Petri Nets 1986, Lecture Notes in Computer Science Vol. 254, 255, Springer 1987

[3] CCITT Recommendation Z.120: Message Sequence Chart (MSC), Geneva, 1992

[4] CCITT SDL Methodology Guidelines, Appendix I to Z.100, Geneva 1992

[5] J. Grabowski: Statische und dynamische Analysen für SDL-Prozessdiagramme auf der Basis von Petri-Netzen und Sequence Charts. University of Hamburg, Diploma Thesis, April 1990

[6] J. Grabowski, D. Hogrefe, P. Ladkin, S. Leue, R. Nahm: Conformance Testing - A Tool for the Generation of Test Cases. Interim Report of the F&E Project Contract No. 233, funded by Swiss PTT, University of Berne, May 1992

[7] J. Grabowski, P. Graubmann, E. Rudolph: Towards an SDL-Design-Methodology Using Sequence Chart Segments. SDL'91 Evolving Methods. O. Faergemand and R. Reed (editors), North-Holland 1991

[8] J. Grabowski, P. Graubmann, E. Rudolph: The Standardization of Message Sequence Charts. Submitted to SESS'93, 1993

[9] P. Graubmann, E. Rudolph: Comments on a Petri Net Based Message Sequence Chart Semantics. CCITT Interims-Meeting, Geneva, 1992

[10] ISO/IEC JTC 1/SC 21: Information Technology - Open Systems Interconnection - Conformance Testing Methology and Framework - Part 3: The Tree and Tabular Combined Notation. Intern. Standard 9646-3, ISO 1991

[11] P. B. Ladkin, S. Leue: An Automaton Interpretation of Message Sequence Charts. Technical Report IAM 92-012, University of Berne, Institute of Informatics and Applied Mathematics, 1992

[12] P. Ladkin, S. Leue: An Analysis of Message Sequence Charts. Technical Report IAM-92-013, University of Berne, 1992

[13] S. Mauw, M. van Wjik, T. Winter: Syntax and Semantics of Synchronous Interworkings, Formal and Informal Semantics. CCITT Interims Meeting, Geneva, 1992

[14] W. Thomas et al.: Automata on Infinite Objects. Handbook of Theoretical Computer Science, pp. 132-191, Elsvier Science Publisher 1990

[15] P. A. J. Tilanus: A Formalization of Message Sequence Charts. SDL'91 Evolving Methods. O. Faergemand and R. Reed (editors), North Holland 1991