MSC'96 and Beyond - a Critical Look

Stefan Loidl, Ekkart Rudolph, Ursula Hinkel[a]

[a]Institut für Informatik, Technische Universität München, D-80290 München
{loidl,rudolphe,hinkel}@informatik.tu-muenchen.de

Deficiencies and inconsistencies within the semantics and graphical syntax of the new MSC'96 concepts are outlined. Problems in the combined use of language constructs, already contained in MSC'92, and the new concepts are discussed. Main goal is the exhibition of open questions and conflicts within the new MSC'96 standard and the presentation of possible solutions. Though already positive experiences have been gained with MSC'96 in the field of protocol specification and prototype implementation, the new concepts need further completion and semantic foundation in order to be successfully applicable in all parts.

1. INTRODUCTION

During the last ITU-study period, 1993 – 1996, Message Sequence Chart has advanced to a considerably powerful and expressive language [6]. Main emphasis has been put on the development of new structural concepts: generalized ordering, new formulation of instance decomposition, inline expression, MSC references, gates and High-level MSCs. In the following, only a brief outline of the new concepts is presented since the paper essentially is addressed to a readership which already is sufficiently familiar with the MSC language. A more extensive description of MSC'96 can be found in [4].

Generalized ordering constructs serve for the definition of general time orderings between MSC events. MSC'92 is restricted to total event ordering on MSC instances (normal case) and to complete unordering of the events contained within coregions. Since MSC instances may refer to higher level entities like, e.g. SDL blocks, language constructs for the specification of more general time orderings within one instance are demanded. The same refers to the event definition on decomposed instances. As a straightforward generalization, the coregion is enhanced by ordering relations graphically represented by special symbols, called connection lines, which denote the generalized time ordering in an intuitive manner. Events between different instances within MSC'92 are ordered merely via messages. However, on an early stage of requirement specification, one often abstracts from the internal message exchange while specifying the external behaviour only. On this level of abstraction, synchronisation constructs are demanded similarly to Time Sequence Diagrams which impose a time ordering between events attached to different instances. This kind of generalized ordering in MSC'96 again is defined by means of ordering relations graphically represented by connection lines between different instances.
Since MSCs can be rather complex, there is a need for a refinement of one instance by a set

of instances defined in another MSC. By means of *instance decomposition*, a refining MSC may be attached to an instance, which describes the events of the decomposed instance on a more detailed level. The refining MSC represents a decomposition of the instance without affecting its observable behaviour, i.e. it must be possible to map the messages leaving or entering the refining MSC to the messages of the decomposed instance. Thus, instance decomposition determines the transition between different levels of abstraction.
By means of *inline expressions*, composition of event structures may be defined inside an MSC. The composition operators *alt*, *par*, *loop*, *exc* and *opt* refer to alternative and parallel composition, iteration, exception and optional region. Graphically, the inline expression is described by a rectangle with dashed horizontal lines as separators. The operator keyword is placed in the left upper corner. Each section of an inline expression in principle again describes one MSC which represents a small trace segment. Thus, inline expressions are ideally suited for the comprehensive description of small variations of system runs.
MSC references are used to refer to other MSCs of the MSC document. The MSC references are objects of the type given by the referenced MSC. Each MSC reference points to another MSC which defines the meaning of the reference, i.e. the reference construct can be seen as a placeholder for an MSC diagram. MSC references may not only refer to a single MSC, but also to MSC reference expressions constructed by means of the operators *alt*, *par*, *seq*, *loop*, *opt*, *exc* and *subst*, and MSC references. By means of the *subst* operator a textual substitution inside the referenced MSC may be defined. Graphically an MSC reference is represented by a rectangle with rounded corners containing the name of the referenced MSC or an MSC expression.
Gates are used to define connection points for messages and order relations with respect to the interior and exterior of MSC references and inline expressions. Gates on inline expressions are merely transit points on the frame of the inline expression. A message gate name can be defined explicitly by a name associated with the gate on the frame or implicitly by the direction of the message through the gate and the message name.
High-level MSCs (HMSCs) provide a means to graphically define how a set of MSCs can be combined. The composition of MSCs specified by HMSCs can be guarded by conditions in the HMSCs. The conditions can be used to indicate global system states. An HMSC is a directed graph where each node is either a start symbol, an end symbol, an MSC reference, a condition, a connection point or a parallel frame. Contrary to plain MSCs, instances and messages are not shown within an MSC. This way, HMSCs can focus completely on the composition aspects. HMSCs are hierarchical in the sense that a reference again may refer to an HMSC.

These new MSC concepts have been stimulated essentially by two main streams of modern computer science: A major input for MSC'96 was provided by the development of a formal MSC semantics within the last ITU study period, based on process algebra ([10,9]). This has led quite naturally to an enhancement of MSC with composition mechanisms which now play a central role within inline expressions, MSC references with operator expressions and HMSCs. Object-oriented techniques have been equally influential, e.g. for the reuse of MSCs by means of MSC references and substitution ([3]). In this context, also the MSC based formalization of Use Cases, which play a central role within the method Objectory ([7]), has been a major stimulus and has contributed to the development of inline expressions and HMSCs. After this period of rapid and sometimes

rather hectical development of new MSC concepts, naturally a period of consolidation has to follow. Most of the new concepts need further elaboration and, in particular, a precise formal foundation. Certainly, the elaboration of a formal semantics will promote the clarification of these language parts considerably. A corresponding standard document is in preparation which is developed along the same lines as the MSC'92 semantics and therefore again is based on process algebra. An outline of parts of the new semantics can be found in [11]. We do not aim to provide a formal definition for the semantics gaps pointed out within this paper. Main goal of this paper, which is based on a diploma thesis ([8]), carried out at Technical University of Munich and Corporate Research and Development of Siemens AG, is to exhibit the deficiencies discovered in MSC'96 and to discuss possible solutions.

2. GENERAL ORDERING RELATION

General ordering is used to describe the temporal order of two events, in case where this cannot be deduced from the ordering imposed by the instances and messages. In the graphical representation, two general ordering symbols are defined - a line symbol without arrow head (general ordering symbol 1) and a line symbol with an arrow head in the middle (general ordering symbol 2). For the ordering of two events attached to different instances only the general ordering symbol 2 is allowed. This symbol can have any orientation and also be bent.

2.1. Combined general ordering and messages

Messages and general ordering relations may cause confusions in the graphical representation. Beginning and ending points of messages and general ordering relations may lead to ambiguities, in case where a message sending event is connected to a general ordering symbol 2. This situation occurs if the message and/or the general ordering symbol crosses one or more instances (see figure 1). It is not obvious whether the message1 is sent from instance1 or instance2 and this applies also for message2 with regard to instance3 and instance4.

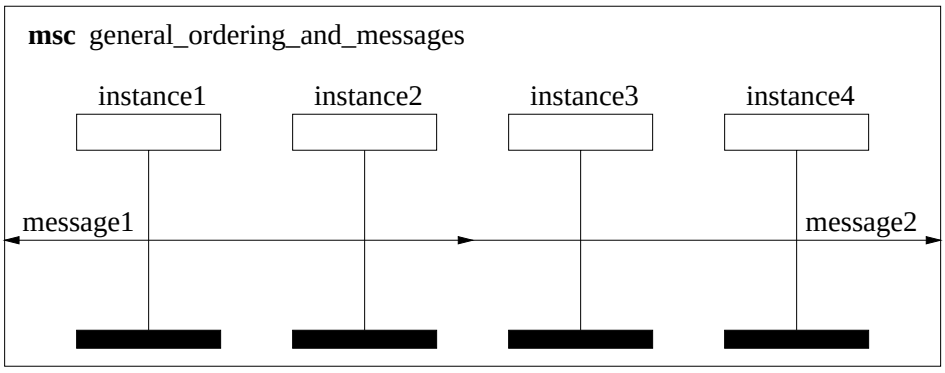

Figure 1. General ordering relation between two messages

Suggested solutions:

- In case, where the general ordering symbol is supposed to cross instances, several arrow heads may be attached to it in order to separate the ordering symbol from message line symbols (see figure 2). This solution does not demand a special arrangement of instances.

- Beginning and end of the general ordering symbol may be indicated by means of special symbols, e.g. semicircles. This solution has the same generality as the second one but needs additional symbols.

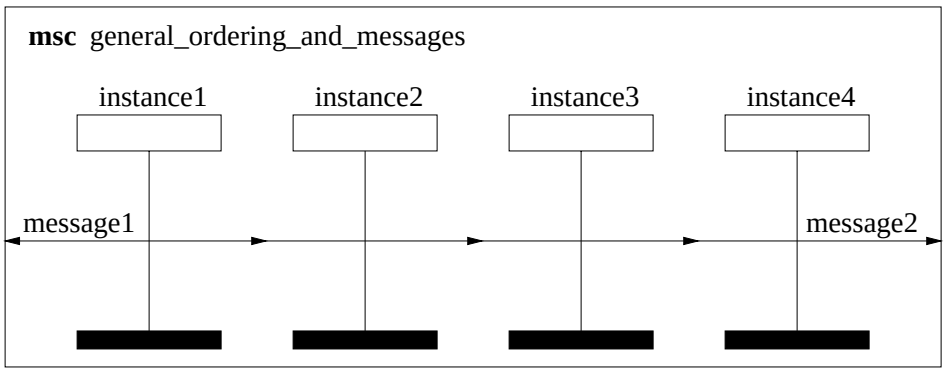

Figure 2. Representation of the general ordering relation with many arrow heads

2.2. Graphical representation of general ordering in combination with other orderable events

Apart from message events, MSC'96 contains several other orderable events: incomplete messages (lost and found messages), create, timer statements and action. The graphical grammar for the combination of general ordering symbols with orderable events appears to be not sufficiently precise, e.g. general ordering symbols may be attached to any point of timer symbols. This may easily lead to rather complicated pictures or even to misinterpretations. Additional drawing rules would be very advantageous, e.g. a rule for timer symbols which states that generalized ordering symbols may only be attached to the connection points of timer symbols with the instance axis.

3. INSTANCE DECOMPOSITION

Within MSC'96, a refining MSC may be attached to an instance. The external behaviour of the refining MSC defined by the messages entering and leaving the environment is formally related to the messages sent and consumed by the decomposed instance. No formal mapping is prescribed for other language constructs like conditions and actions.

3.1. Generalized ordering and instance decomposition

Generalized ordering relations between events on decomposed instances and events on other instances or gates are not excluded within MSC'96. Therefore, it seems to be natural to define a formal mapping between decomposed instances and refining MSCs also for general ordering relations in analogy to messages. However, the mapping is unclear in case where more than one general ordering relation are attached to one decomposed instance since no names are associated with general ordering relations. Since gates are not defined on decomposed instances also a mapping via gates is not possible (see figure 3).

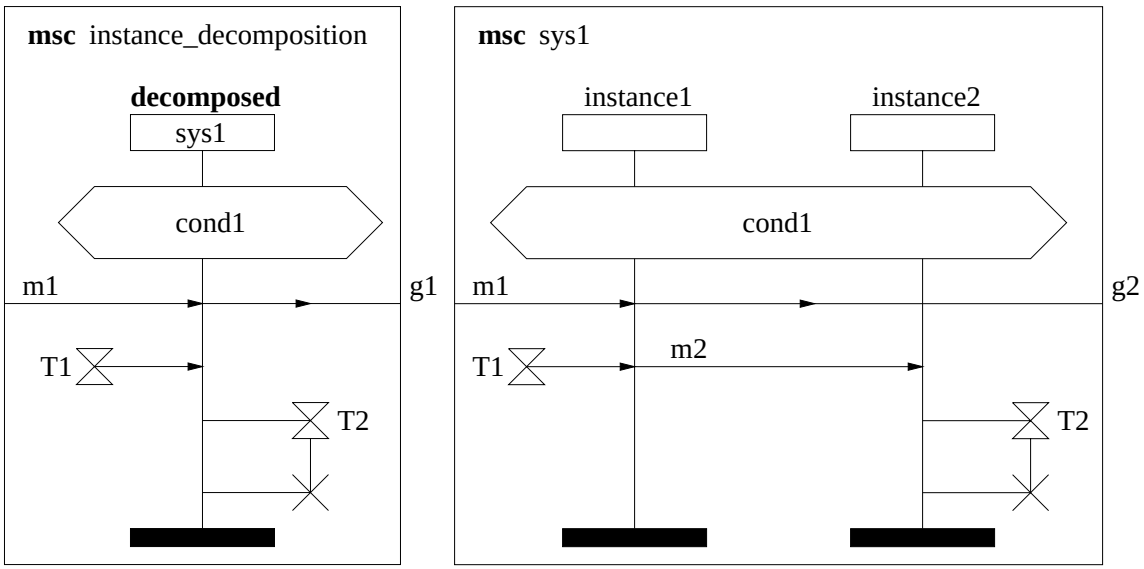

Figure 3. Instance decomposition with general ordering, timer and conditions

3.2. Timer and instance decomposition

Within MSC'96, no rules concerning the mapping of timer elements of a decomposed instance to timer elements in the refining MSC are defined. Timers, however, are essential constituents of the MSC'96 standard. One possible solution is to map each timer construct of a decomposed instance to a corresponding timer construct on one of the instances of the refining MSC. Together with the additional rule that correlated timer constructs, like timer setting and timeout of the same timer, have to be attached to the same instance within the refining MSC, a consistent mapping of timers for decomposed instances can be achieved.

3.3. Instance creation and instance stop in combination with instance decomposition

Within the MSC'96 standard, an instance may be created by a decomposed instance, however, this instance creation cannot be mapped consistently onto the refining MSC. In order to achieve such a formal mapping the standard has to be extended by including

gates for instance creation events in addition to messages and general ordering relations. Otherwise, instance creation should be disallowed on decomposed instances.

Equally, it is not defined how the termination of decomposed instances by an instance stop - which is possible within MSC'96 - may be carried over to the refining MSC. Two solutions seem to be possible: Either all instances within the refining MSC have to stop or instance stop is disallowed on decomposed instances. A similar reasoning holds for the creation of decomposed instances.

3.4. Conditions on decomposed instances

Within MSC'96, there is also no formal mapping defined for conditions between decomposed instance and refining MSC. This is problematic, in particular, in case where the condition attached to the decomposed instance also refers to other instances (non-local condition) or even to all instances in the MSC (global condition). The case of global initial and final conditions deserves special attention since such global conditions play an important role for MSC composition. Therefore, at least for global initial and final conditions, a formal mapping between decomposed instance and refining MSC should be defined. It seems to be reasonable that initial and final conditions in an MSC containing a decomposed instance also appear as initial and final condition in the refining MSC with the same condition name (see figure 3). This would perfectly fit with the interpretation that global conditions always refer to all instances contained in the MSC document. However, this interpretation needs further discussion, too. The alternative, to introduce gates also for conditions, appears to be not very attractive. It should be noted that the role of decomposed instances within MSC composition needs further elaboration.

3.5. Inline expressions and references on decomposed instances

Within MSC'96, inline expressions and references must not be attached to decomposed instances. In practice, such a rule seems to be too restrictive. In particular, the use of MSC references without operator expressions in combination with decomposed instances is requested. E.g., related to SDL, it should be possible to attach references to instances of type block which may be further decomposed into refining MSCs containing instances of type process. Since decomposed instances refer to a vertical (de)composition and references or inline expressions to a horizontal (de)composition the combination seems to produce nontrivial problems. In full generality, this appears to be a challenging task for MSC2000.

4. INLINE EXPRESSIONS

Inline expressions introduced in MSC'96 contain several conflicting points: The textual and graphical grammar contain some inconsistencies and the semantics of some operators (*alt*, *opt*, *exc*, *loop*) is not sufficiently defined.

4.1. Inconsistencies of textual and graphical grammar

The textual grammar allows the use of instance creation, instance stop and the representation of instance beginning and instance end within inline expressions. Within the graphical grammar, instance stop and the beginning and end of instances are excluded inside an inline expression. A create event is not explicitly ruled out, however, this does

not make sense without a new instance beginning. Apart from these inconsistencies, the use of instance creation and stop in connection with operator expressions leads to several semantic problems.

From that we conclude that for both, textual and graphical grammar, creation and stop as well as beginning and end of instances should be disallowed within inline expressions.

4.2. Inline expressions with gates

Inline expressions with message gates or general ordering gates can lead to problems. As an example (see figure 4), we take an optional region with a message 'm1' crossing an inline gate and being sent to the instance 'instance2'. Due to the optionality, it is possible that the message will not be sent from the inline expression and therefore also not be received by 'instance2' which causes a deadlock on 'instance2'. For an illustration, the semantics of the MSC 'opt_inline_expression' is modelled by a corresponding Petri net (condition event system, see [2]).

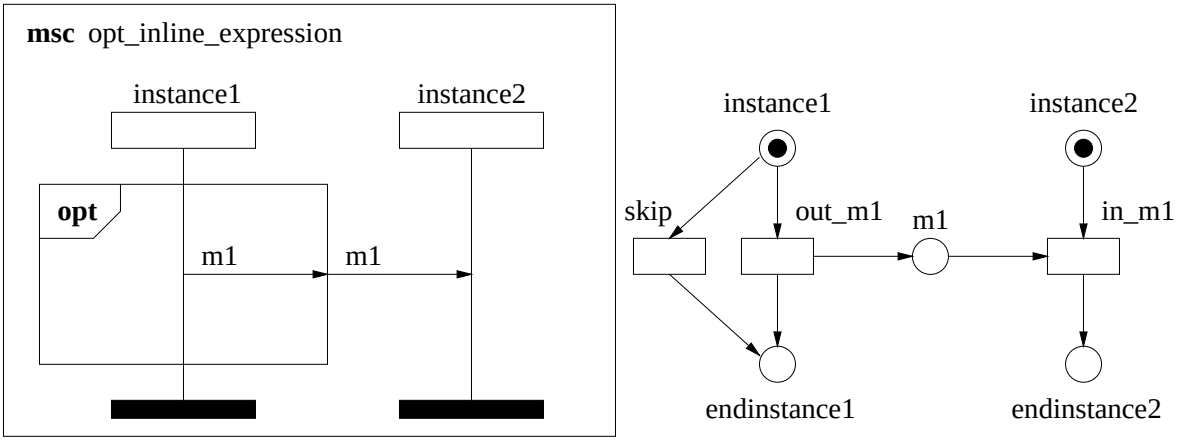

Figure 4. Optional inline expression with message gate

Within the corresponding Petri net, obviously a conflict arises between the transitions 'skip' and 'out_m1'. In case, where the transition 'out_m1' fires, the transition 'in_m1' is enabled and may fire. In case, where the transition 'skip' fires, the transition 'in_m1' is deadlocked.

Similar examples can be constructed for the *alt* operator. Gates for general ordering relations lead to corresponding problems. In case of the *alt* operator, conflicts could be avoided if all alternatives contain the same gate interface. In general, however, a considerable number of additional static semantic rules would be necessary to avoid deadlock situations. Therefore, it has been decided in MSC'96 to leave it to the user's own responsibility.

The *loop* operator may cause several problems due to the repetition of events inside the inline expression, in particular in connection with gates (see figure 5). A message entering an inline expression with a *loop* operation cannot be received several times inside if it

is sent only once outside. A consistent solution may be obtained only if the messages arise from a corresponding loop expression. In case, where the gate refers to a general ordering relation, the interpretation is even more problematic: The semantics of general ordering relations in connection with the repetition of events is not defined. *Loop* expressions may lead to interpretation problems also for timer constructs. Timer in inline expressions with a *loop* operation cannot expire several times without being set again. This situation corresponds to the message gate problem, only that in the case of timer constructs the connection between individual timer events is not defined via gates but via name identification.

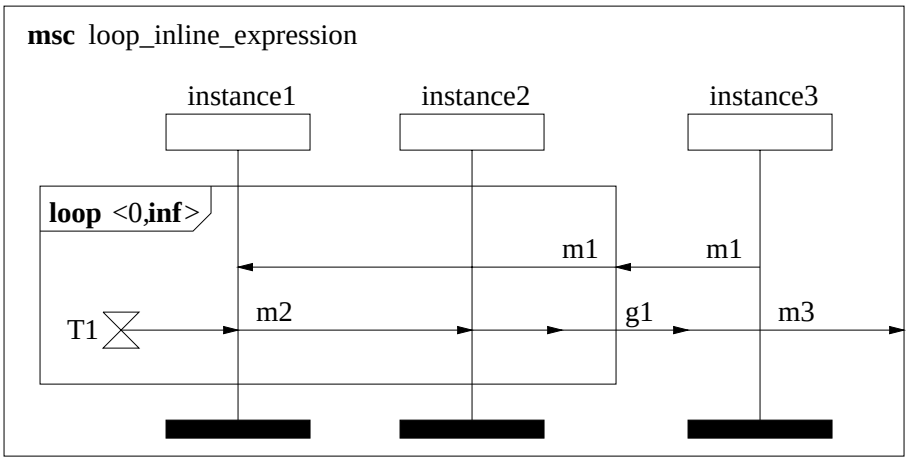

Figure 5. Inline expression with loop operator

Within MSC 'loop_inline_expression', the message 'm1' is sent only once but, according to the *loop* expression, it should be possibly received and consumed several times. Therefore, after the first repetition, a deadlock results on 'instance1'. The inverse case, where the message is leaving the inline expression, may be interpreted in form of lost messages after the first repetition.

The ordering of message 'm2' with respect to message 'm3' is not obvious in case of the loop. It may be interpreted in the way that all repeated message events 'm2' are ordered with respect to the single message event 'm3'.

MSC 'loop_inline_expression' also contains the expiration of timer 'T1' which according to the *loop* expression may possibly be repeated several times. Again, this leads to a deadlock on 'instance1' after the first repetition.

4.3. Inline expressions containing the exc operator

The *exc* operator may be employed within inline expressions and MSC references containing operator expressions. The semantics is defined in MSC'96 in form of a shorthand notation: The *exc* operator can be viewed as an alternative inline expression where the second operand is the entire rest of the MSC. The meaning of the operator is that either the events inside the *exc* inline expression are executed and then the MSC is finished or

the events following the *exc* inline expression are executed. For nested inline expressions this rule may lead to inconsistencies. For an *exc* inline expression in one of the sections of a parallel inline expression, the semantics is not precise. It is not clear whether the events in the other sections of the parallel expression shall be executed or not.

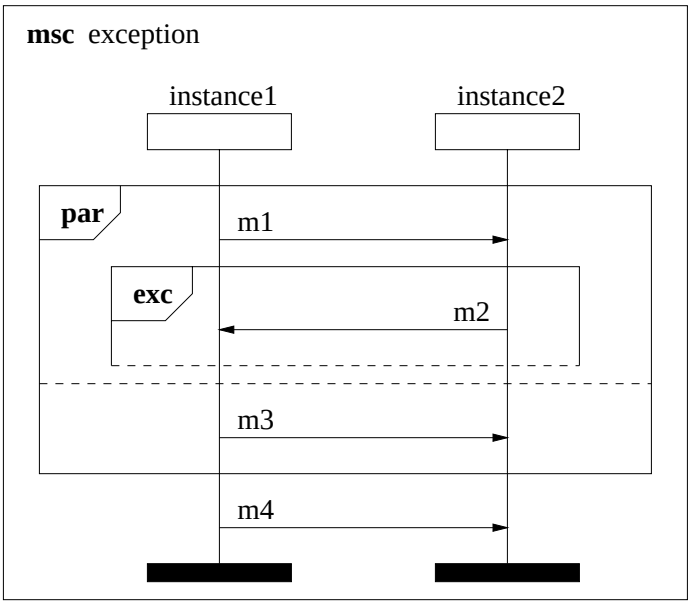

Figure 6. MSC with nested *exc* inline expression

Within MSC 'exception' in figure 6, the message 'm3' is exchanged in parallel with the exception region containing the message 'm2'. It is not clear how to interpret the entire rest of the MSC with respect to the exception region. One possibility to solve this ambiguity concerning nested expressions would be to relate the *exc* region to the section in which it is contained. After the execution of the exception inline expression the rest of the section in which it is contained is skipped. A probably more realistic solution would be to forbid exception regions within nested inline expressions. The same problems occur in MSC references with operator expressions (see section 5).

5. MSC REFERENCES

MSC references are comparable in their functionality with inline expressions. Inline expressions can be represented equivalently by references with operator expressions. The gate interface of an inline expression corresponds to the gate interface of an MSC reference together with the gate interface of the referenced MSCs. The other way round, apart from some minor discrepancies, MSC references may be substituted by inline expressions with the additional rule that simple expressions containing only one MSC reference or sequentially composed MSC references are replaced by their expanded form. Because of this equivalence, we discuss only problems specific for MSC references in this section. All

conflicts listed for inline expressions apply to MSC references as well.

5.1. Decomposed instances in MSC references

Decomposed instances must not be attached to MSC references according to MSC'96. Contrary to that, decomposed instances may appear within referenced MSCs. This is consistent since the decomposed instance may only appear in the referenced MSC but it has to be ruled out explicitly that a non-decomposed instance with the same name is contained in the enclosing MSC. In case of a decomposed instance in a referenced MSC, the enclosing MSC may contain instances of the refining MSC. Within MSC'96, the semantics of such a combination is not clear.

5.2. Cyclic connectivity graphs caused by MSC references with gates

According to the drawing rules and static semantics rules for messages, deadlocks are excluded within MSC'92, i.e. the corresponding connectivity graph cannot contain loops. Contrary to that, general ordering relations in MSC'96 may lead to cyclic graphs since no special drawing rules (or static semantics rules in the textual grammar) are prescribed. Possible deadlocks can be prevented by forbidding general ordering symbols with upward slope.

This, however, is not sufficient since MSC references with gates may lead to deadlocks even if special drawing rules for messages and general ordering relations are obeyed. The problem arises since the graphical order of gate definitions on an MSC frame need not agree with the order of corresponding actual gates on the MSC reference. This may be illustrated by the example in figure 7:

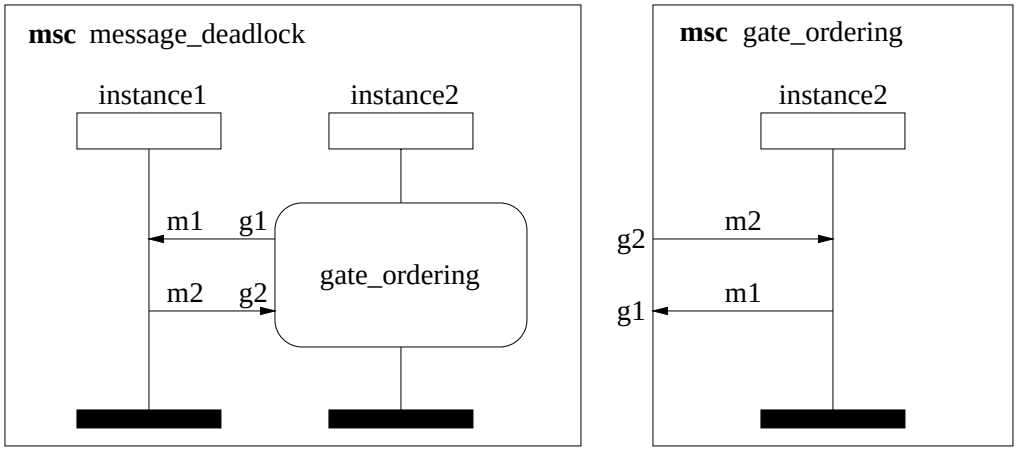

Figure 7. Deadlock caused by message gates

Obviously, the MSC 'message_deadlock' with the MSC reference 'gate_ordering' leads to a deadlock since the order of gate definitions, 'g1' and 'g2', in 'MSC gate_ordering' has been interchanged for actual gates in MSC 'message_deadlock'. This problem may be solved by an additional drawing rule: The graphical order of actual gates has to agree

with the order of gate definitions. In the textual grammar, additional static semantics rules are necessary which exclude loops of the connectivity graph including message events for the referenced MSC. Similar problems occur if the gates refer to generalized ordering relations.

5.3. Gate interface of MSC reference expressions

Within MSC'96, the relation between the gate interface of the MSC reference and the interface of the MSCs referenced in the expression may be quite intricate. Not all gate definitions of the referenced MSCs must have a correspondence in the gate interface of the MSC reference. Gate definitions of the referenced MSCs which have no corresponding gate interface are propagated to the environment. The other way round, the interface of the MSC reference must match the interface of the MSCs referenced in the expression, i.e. any gates attached to the reference must have a corresponding gate definition in the referenced MSCs. However, the interface of the MSC reference in general does not match the interface of only one of the referenced MSCs. It is not always obvious how the gate interface of an operator expression is built of the gate interfaces of the individual referenced MSCs. The referenced MSCs of an alternative expression may have different gate interfaces so that the gate interface of the expression has to be adjusted accordingly. For parallel and sequential operator expressions, it even seems that the gate interfaces have to be different for the individual referenced MSCs in order to be meaningful. In any case, a clear semantics definition is demanded urgently.

5.4. Substitution in MSC references

MSC'96 essentially contains four semantic rules for the substitution in MSC references:

- All substitutions in a substitution list are thought to be applied in parallel. Thus the order in which the substitutions take place is not relevant.
- The substitution of an MSC name must share the same gate interface as what it replaces.
- If an MSC definition to which substitution is applied contains MSC references, then the substitution should be applied also to the MSC definitions corresponding to these MSC references.
- An MSC containing references and substitutions is illegal if the application of the substitutions results in an illegal MSC.

In some cases, additional static semantics rules are necessary: The parallel substitution of the same element is not excluded. The following MSC expression is syntactically correct: 'MSC1 *subst* msg m1 *by* m2, msg m1 *by* m3'. The result, however, is unclear. It is not defined which operation has to be executed first.

As the substitution is defined recursively within MSC'96, all substitutions in an MSC definition are carried over to the MSCs, referenced within this MSC. In case that the referenced MSCs contain further substitution lists, this may lead to undefined overlapping, i.e. it is not clear which substitution has to be carried out first. Apart from such semantic problems, the substitution concept seems to be too narrow in practice: Only instance

names but not instance types may be replaced. For messages, only the message names may be substituted but in practice also the substitution of message parameters is requested. Beyond that, also the substitution for other language constructs, like timer and conditions, may be useful.

6. HIGH LEVEL MSCS

Whereas in MSC'92 the composition was based completely on the merging of final and initial conditions, within MSC'96, the composition of MSCs is defined by means of HMSCs. The conditions in MSC'96 play a restrictive role defined by a set of static semantics rules. Compared with MSC'92, the composition mechanisms of MSC'96 are much more flexible, e.g. MSCs may be composed with or without initial or final conditions. This flexibility, however, leads to some semantic problems which need further clarification.

6.1. HMSCs without end symbol

According to the MSC'96 standard, an HMSC need not contain an end-symbol, i.e. it may be completely cyclic. In case, where an MSC reference refers to a cyclic HMSC, this leads to a dead branch because the events following the MSC reference will not be executed any more. Therefore, an MSC reference pointing to a cyclic HMSC should be disallowed.

6.2. Conflicts in the semantics of MSC composition

For HMSCs, similar problems as for operator expressions may occur in connection with instance creation, instance stop and timers contained in the referenced MSCs. The following situations which are allowed for HMSCs according to MSC'96 lead to inconsistencies:

- Instances which are terminated within one referenced MSC may be contained in the subsequent referenced MSC or in a parallely executed MSC.
- In HMSCs containing free loops, instances in referenced MSCs may be created several times.
- In HMSCs containing free loops, instances in referenced MSCs may be terminated several times.
- Instances with the same name may be contained in one referenced MSC as decomposed, in another one as non-decomposed.
- Instances with the same name may vary in different MSC references by the instance type.
- The same timer may expire in subsequent referenced MSCs without being set between them again.

These inconsistencies should be disallowed by additional static semantics rules.

6.3. Composition rules for HMSCs

At a first glance the role of conditions on the level of HMSCs is not immediately visible. Since the HMSC conditions are not mandatory, they only play a restrictive role. Their use becomes more obvious in examples taken from practice: HMSCs without conditions become difficult to handle since conditions, representing global system states, provide natural check points.

Nevertheless, conditions on the HMSC level could play a more significant role if they are employed for a dynamical choice in form of guards. In fact, such a mechanism is demanded urgently. Within MSC'96, alternatives defined within a referenced MSC cannot be continued differently outside of the reference. In practice, this makes the specification, in particular of exception handling, quite clumsy. E.g. in figure 8, within the HMSC 'setup_attach', the choice made in the referenced MSC 'connect_request' between failure (first alternative) and successful connection (second alternative) cannot be carried over to the subsequent branching in the HMSC. That means, according to the present standard, both branches inside of MSC 'connect_request' can be continued by both branches outside (with the HMSC conditions 'failure' and 'connection'). It would be advantageous to change the semantics and to employ HMSC conditions as guards.

Some immediate deficiencies of the present standard may be removed by a reformulation of the static semantics rules. In case of the composition of simple MSCs without (unique) global initial (final) conditions, the set of initial (final) conditions is defined to be the set of all possible condition names. By means of this default value no empty sets of initial or final conditions can be produced. Such a rule is not carried over consistently to referenced HMSCs and referenced MSC expressions. E.g. the sets of initial (final) conditions of *alt* and *par* MSC expressions are defined as the intersection of the sets of initial (final) conditions of the referenced MSCs. This way, the sets can be empty. Similarly to simple MSCs, the default value in this case should be the set of all possible condition names. Beyond that, for the *exc* operator expression any definition is lacking in the standard.

7. CONCLUSION AND OUTLOOK

Contrary to MSC'92 ([5]), the new standard MSC'96 offers the promising possibility of a fairly comprehensive system specification in an intuitive and transparent manner. In this respect, MSC'96 has been applied already successfully to an ISDN service specification ([12]) and to the formalization of Use Cases ([1]). Practical experience has been gained by the implementation of one of the new MSC'96 concepts (MSC reference) as a prototype component within the Siemens SICAT tool. This has been part of the diploma thesis on which this paper is based ([8]).

Nevertheless, as was pointed out in the preceding chapters, MSC'96 still contains a number of obvious deficiencies, inconsistencies and semantic gaps. Most evidently, MSC inline expressions and MSC reference expressions combined with gate concepts need further elaboration and precise mathematical foundation. This, of course, also demands an intense feedback from tool makers, users and the academic community since standardization is a highly interactive process. Certainly, the completion of the formal semantics for MSC'96 based on process algebra will contribute considerably to a further consolidation

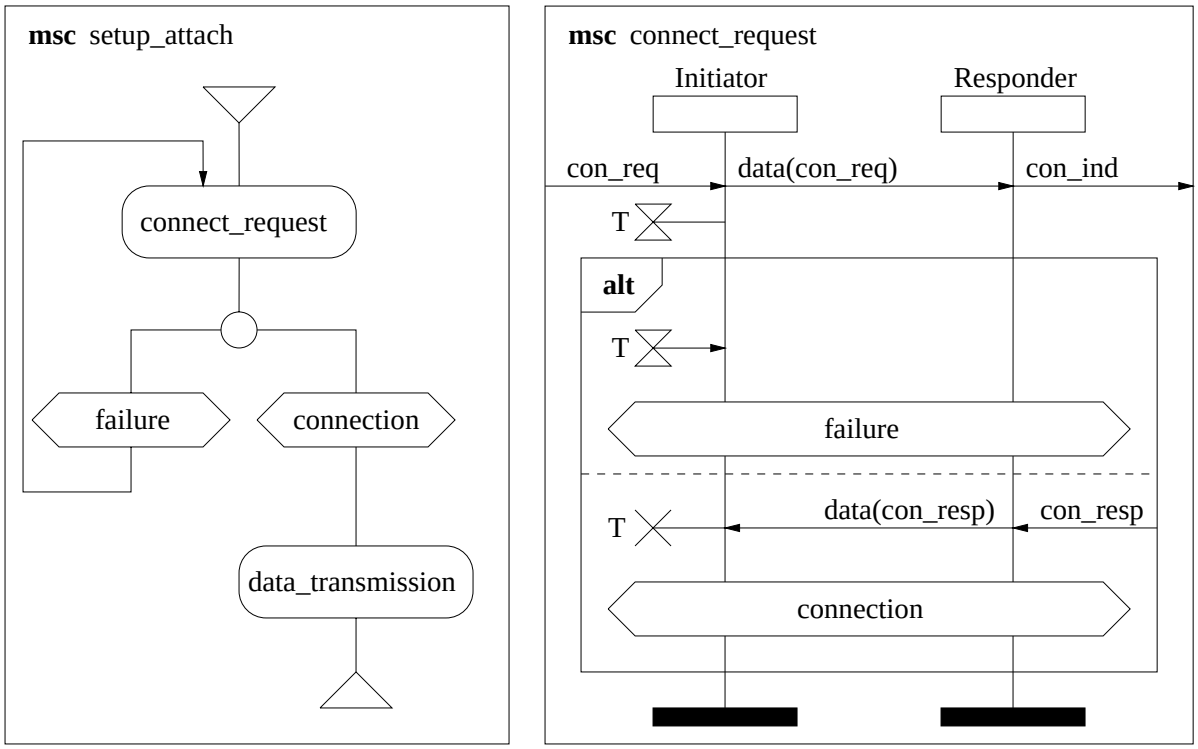

Figure 8. HMSC and MSC reference for connection setup

of the language.

Whilst most of the new MSC'96 concepts need further elaboration, a number of important concepts have been left out in the Z.120 since they were found either to be not sufficiently mature or they have not been sufficiently supported yet by the standardization community. The list contains disruption and interruption operator, parallel composition with synchronisation mechanism, strong sequencing operator, concepts for synchronous communication, formal data concepts, inclusion of non-functional properties. The list may be easily extended. All these open items are included in the working program for the next ITU-study period which is supposed to result in an addendum to MSC'96 in 1998 and in a new recommendation in 2000 (MSC2000).

ACKNOWLEDGEMENTS

We thank Dieter Kolb for interesting discussions about MSC'96. We are grateful to Sjouke Mauw and Michel Reniers for having a critical look at our paper.

REFERENCES

1. M. Andersson and J. Bergstrand. Formalizing Use Cases with Message Sequence Charts. Master's thesis, Lund Institute of Technology, 1995.

2. P. Graubmann, E. Rudolph, and J. Grabowski. Towards a Petri Net Based Semantics Definition for Message Sequence Charts. In O. Færgemand and A. Sarma, editors, *SDL'93 - Using Objects*. North-Holland, 1993.
3. Ø. Haugen. MSC'96 – The advanced MSC. SISU Report L–2002–4, September 1996.
4. Ø. Haugen. The MSC-96 Distillery. In A. Cavalli and A. Sarma, editors, *SDL'97*. North-Holland, 1997.
5. ITU-T. *Z.120 – Message Sequence Chart (MSC)*. ITU-T, Geneva, 1994.
6. ITU-T. *Z.120 – Message Sequence Chart (MSC)*. ITU-T, Geneva, 1996.
7. I. Jacobson. *Object-Oriented Software Engineering – A Use Case Driven Approach*. Addison-Wesley, 1992.
8. S. Loidl. Interpretation und Werkzeugunterstützung von Message Sequence Charts (MSC'96). Master's thesis, Technische Universität München, 1996. in German.
9. S. Mauw. The formalization of Message Sequence Charts. *Computer Networks and ISDN Systems - SDL and MSC*, 28(12), June 1996.
10. S. Mauw and M.A. Reniers. An algebraic semantics of Basic Message Sequence Charts. *The Computer Journal*, (37), 1994.
11. S. Mauw and M.A. Reniers. High-level Message Sequence Charts. In A. Cavalli and A. Sarma, editors, *SDL'97*. North-Holland, 1997.
12. E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts (MSC'96), Forte/PSTV'96, Kaiserslautern, October 1996.