Tutorial on Message Sequence Charts

Ekkart Rudolph[a], Peter Graubmann[b], and Jens Grabowski[c]

[a]Technical University of Munich, Institute for Informatics, Arcisstrasse 21, D-80290 München, Germany, eMail: rudolphe@informatik.tu-muenchen.de

[b]Siemens AG, ZFE T SE, Otto-Hahn-Ring 6, D-81739 München, Germany, eMail: gr@zfe.siemens.de

[c]University of Berne, Institute for Informatics , Neubrückstrasse 10, CH-3012 Berne, Switzerland, eMail: grabowsk@iam.unibe.ch

1. Introduction

Message Sequence Charts (MSCs) are a widespread means for the visualization of selected system runs (traces) within communication systems. They can be viewed as a special trace language which mainly concentrates on message interchange by communicating entities (such as SDL services, processes, blocks) and their environment. A main advantage of an MSC is its clear graphical layout which immediately gives an intuitive understanding of the described system behaviour. MSCs have been used informally for a long time by ITU (former CCITT) Study Groups in their recommendations and in industry. Their standardization was suggested at the 4.th SDL Forum October 1989 in Lisbon [7] and agreed upon at the ITU-meeting Helsinki, June 1990 [4]. At the closing session of the ITU study period 1989-1992 in Geneva, May 1992, the new MSC recommendation Z.120 [19] was approved. Within the present study period, as a major achievement, a formal semantics for MSCs based on process algebra has been standardized [20]. A standard document on static syntax requirements is in preparation [15]. Besides formal semantics main emphasis is put on structural concepts [8,9].

The reason to standardize MSCs was to allow systematic tool support, to facilitate the exchange between different tools, and to ease the mapping to and from SDL specifications. Due to the standardization, the importance of MSCs for system engineering has increased considerably. Accordingly, MSCs are used

- for requirement definition [7,18],
- as an overview specification of process communication [18],
- as an interface specification [18],
- as a basis for automatic generation of skeleton SDL specifications [7],
- for simulation and consistency check of SDL specifications [1,2,13,14],
- as a basis for selection and specification of test cases [5,6],

- for documentation [7],
- for object oriented design and analysis (object interaction) [12].

2. Why yet another Specification Language?

One way to understand the meaning and the usefulness of MSCs may be by relating them to other specification languages like SDL, LOTOS or Petri Nets. In practice, the MSC language is used most frequently in connection with SDL and indeed, in addition, it also has been standardized in the same ITU-T study group as SDL. What was the reason for the introduction of yet another standard language besides SDL? SDL processes and MSCs can be looked at as two different kinds of system representations which are complementary in many respects. SDL provides a clear and comprehensive behaviour description within individual SDL processes, whereas the communication between several processes is represented in a fairly indirect manner and thus the description of the communication behaviour in SDL for many purposes is not sufficiently transparent. Contrary to that, MSCs focus on the communication behaviour of system components and their environment by means of message exchange. MSCs provide a clear description of system traces in form of message flow diagrams. In contrast to SDL, the set of specified MSCs usually covers a partial system behaviour only since each MSC represents exactly one scenario. So, candidates for MSCs are primarily the standard cases. These standard cases may be supplemented by MSCs describing exceptional behaviour altogether providing a use case like representation [12]. Recently, attempts have been made to enhance the language and to make a fairly comprehensive system description feasible by using composition and object oriented techniques. Most certainly, however, the main importance of MSCs also in the future will not lie in complete system descriptions but rather in the specification of special system properties, e.g., of certain system runs which should be allowed or, the other way round, which should be disallowed. More generally, MSCs have been proposed for the intuitive representation of temporal logics expressions. Within the system development process, MSCs play a role in nearly all stages (see Chapter 1) complementing SDL in many respects. In all cases, the strength of MSCs lies in the clear and intuitive description of selected system runs whereas SDL is used for a complete system specification.

It should be pointed out that the usual incompleteness of MSCs does not mean that they have only informal, i.e., illustrative character within a system development. Their role may very well be formalized, e.g., MSCs may represent test purposes for the automatic generation of test cases as it is done within the SAMSTAG method [5,6].

For an illustration of the relation between SDL and MSC, in the following, we use the Inres service specification as a standard example [10]. Let us consider the MSC *conreq* in Figure 1 which describes a selected trace piece of the connection set-up in the *Inres* service specification: An Initiator-user sends a connection request (ICONreq) to the Initiator. The Initiator transmits the request (ICON) to the Responder entity which afterwards indicates the connection request (ICONind) to its user. The MSC is related to corresponding paths in SDL process diagrams (Figure 2) where the path corresponding to the MSC in Figure 1 is indicated by bold arrows. Obviously, the trace described by the MSC can be represented also in form of SDL diagrams.

The correspondence between Figure 1 and Figure 2 may serve to give a good intuitive

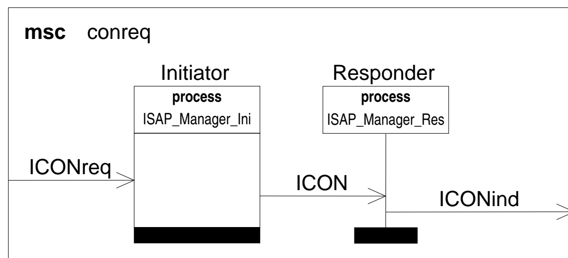


Figure 1. Connection Request

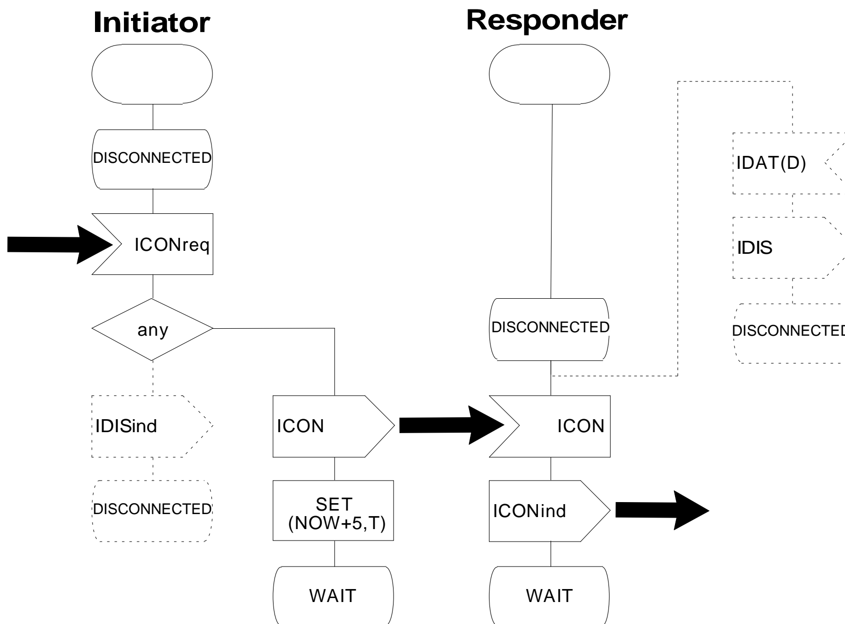


Figure 2. SDL-diagram corresponding to the MSC in Figure 1

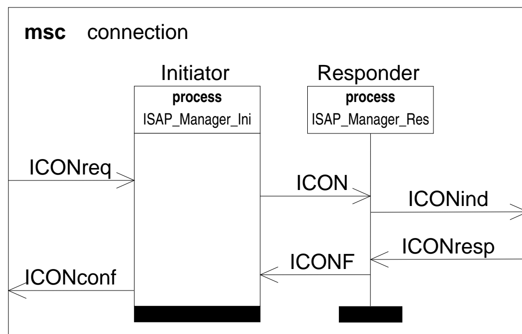
idea about the meaning but also the usefulness of MSCs. It also demonstrates that an MSC describing one possible scenario can be viewed as an SDL skeleton [7,18]. Obviously, the MSC in Figure 1 is far more transparent than Figure 2, since it concentrates on the relevant information, namely the instances (Initiator, Responder) and the messages involved in the selected trace piece (ICONreq, ICON, ICONind).

3. The MSC Language

3.1. MSC/PR and MSC/GR

In analogy to the SDL recommendation Z.100 [17] the new MSC recommendation Z.120 [19] includes two syntactical forms, MSC/PR as a pure textual and MSC/GR as a graphical representation. An MSC in MSC/GR representation can be transformed automatically into a corresponding MSC/PR representation. The other way round, the same problems arise as in SDL since MSC/PR (like SDL/PR) does not include graphical information such as height, width, or alignment of symbols and texts. An example of the MSC/GR and the corresponding MSC/PR representation is shown in Figure 3.

The MSC/PR presently contained in Z.120, lists message sending and receiving events




```
msc connection;
inst Initiator, Responder;
  instance Initiator: process ISAP_Manager_Ini;
    in ICONreq from env;
    out ICON to Responder;
    in ICONF from Responder;
    out ICONconf to env;
  endinstance;
  instance Responder: process ISAP_Manager_Resp;
    in ICON from Initiator;
    out ICONind to env;
    in ICONresp from env;
    out ICONconf to Initiator;
  endinstance;
endmsc;
```

Figure 3. MSC in MSC/GR and in the corresponding MSC/PR

in association with an instance. Recently, a better readable notation was requested, in particular in cases where MSC/PRs were not only used internally by tools, but also edited by humans. Thus, a new *event oriented* textual representation was elaborated [3,11] where events are listed in form of a possible execution trace and not ordered with respect to instances. The event oriented textual grammar is closer to the graphical grammar than the instance oriented syntax and details concerning the graphical representation are expressible contrary to the present instance oriented textual syntax. This is particularly important for applications where an MSC semantics variant is preferred describing global event ordering, i.e., a global time scale, in contrast to the present partial ordering interpretation of MSC diagrams. Such applications arise in case of validation, tracing, debugging, simulation, and test case specification. Consequently, a PR-GR transformation is easier (less ambiguous) for the event oriented than for the instance oriented textual grammar. Furthermore, the event oriented grammar is demanded within special stages of system development: it is particularly applied when execution sequences from, e.g., simulations, have to be recorded. For the example of Figure 3 the following event oriented textual syntax description is obtained:

```
msc connection;
inst Initiator, Responder;
Initiator:    instancehead process ISAP_Manager_Ini;
Responder:    instancehead process ISAP_Manager_Resp;
Initiator:    in ICONreq from env;
Initiator:    out ICON to Responder;
Responder:    in ICON from Initiator;
Responder:    out ICONind to env;
Responder:    in ICONresp from env;
Responder:    out ICONconf to Initiator;
Initiator:    in ICONF from Responder;
Initiator:    out ICONconf to env;
Initiator:    endinstance;
Responder:    endinstance;
endmsc;
```

In order to be able to combine the advantages of both textual syntax forms a mixture of syntax forms is allowed.

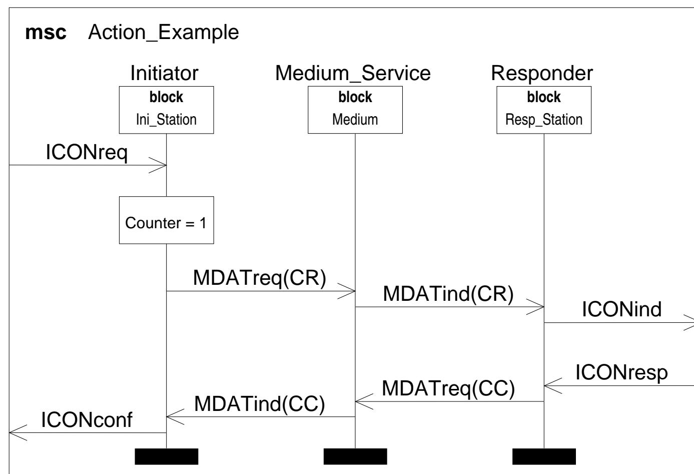


Figure 4. Action

3.2. Basic Language Elements

The basic language of MSCs includes all constructs which are necessary in order to specify the pure message flow. For MSCs these language constructs are instance, message, environment, action, timer set, timer reset, time-out, instance creation, instance stop, and condition.

3.2.1. Instances and Messages

The most fundamental language constructs of MSCs are instances (e.g., entities of SDL systems, blocks, processes, or services) and messages describing the communication events. In the graphical representation, instances are shown by vertical lines or, alternatively, by columns (Figure 3). Within the instance heading an entity name, e.g., process type, may be specified in addition to the instance name.

The message flow is presented by arrows which may be horizontal or with a downward slope with respect to the direction of the arrow to indicate the flow of time. In addition, the horizontal arrow lines may be bended to admit message overtaking or crossing (Figure 5b). The head of the message arrow denotes the event *message consumption*, the opposite end the event *message sending*. In addition to the message name, message parameters in parentheses may be assigned to a message (Figure 4).

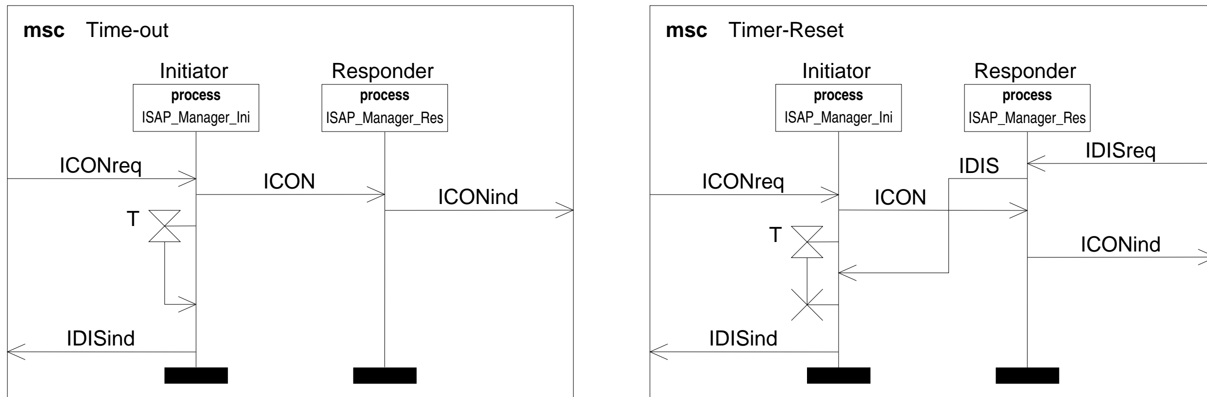
Along each instance axis a total ordering of the described communication events is assumed. Events of different instances are ordered only via messages, since a message must be sent before it is consumed.

3.2.2. System Environment

Within an MSC, the system environment is graphically represented by the frame symbol which forms the boundary of an MSC diagram. In contrast to instances, no ordering of communication events is assumed within the environment.

3.2.3. Actions

In addition to message exchange, actions triggered by messages may be specified in MSCs. An action is graphically represented by a rectangle containing arbitrary text (Figure 4).



(a) Time-out

(b) Timer Reset

Figure 5. MSC timer constructs

3.2.4. Timer

Timer handling in MSCs encloses the setting of a timer and a subsequent time-out (timer expiration) or the setting of a timer and a subsequent timer reset (time supervision). The corresponding MSC/GR constructs are shown in Figure 5. The new timer symbols, differing from the graphical symbols defined in Z.120, have been requested and elaborated within the present study period. They will be part of MSC'96 [11].

The individual timer constructs (timer setting, reset/timeout) may be split between different MSCs in cases where the whole scenario is obtained from the composition of several MSCs (cf. Section 3.4). The setting of a timer is represented by an hour-glass connected with the instance axis by a (bended) line symbol. The reset symbol is presented by a cross (X), again connected with the instance axis by a (bended) line symbol. Time-out is described by an arrow which is connected to the hour-glass symbol. An (optional) timer description containing name and duration may be associated with each timer symbol. In complete system descriptions, for each timer setting a corresponding time-out or timer reset, respectively, has to be specified and vice versa. The corresponding symbols, however, do not necessarily appear within the same MSC.

3.2.5. Creation and Termination of Instances

Creation and termination of instances within communication systems are quite common events. This is due to the fact that most communication systems are dynamic systems where instances appear and disappear during system lifetime. Consequently, a system designer needs features to describe such events. The corresponding MSC language elements are shown in Figure 6. The create symbol is a dashed arrow which may be associated with textual parameters. A create arrow originates from a parent instance and points at the instance head of the child instance. Correspondingly to messages, a create event may be labeled with a parameter list.

An instance can terminate by executing a process stop event. Execution of a process stop is allowed only as last event in the description of an instance. The termination of an instance is graphically represented by a stop symbol in form of a cross at the end of the instance axis (Figure 6).

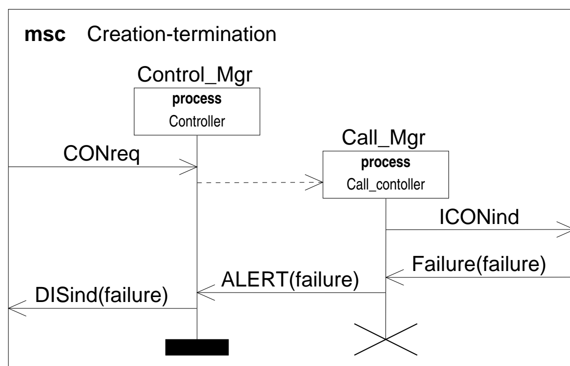


Figure 6. Instance creation and termination

3.2.6. Conditions

A condition describes a state referring to a non-empty set of instances specified in the MSC. A condition either describes a global system state referring to all instances contained in the MSC, in the following called global condition, or a state referring to a subset of instances, also called non-global condition. If it refers to one instance only, a condition is called local. Conditions can be used to emphasize important states within an MSC or for the composition and decomposition of MSCs (Section 3.4). In the MSC/GR representation local, global and non-global conditions are represented by hexagons covering the instances involved (Figure 8).

In the textual representation the condition has to be defined on every instance it refers to, using the keyword **condition** together with the condition name. If the condition refers to more than one instance the keyword **shared** together with an instance list denotes the set of instances to which the condition is attached. By means of the keyword **shared all**, a condition referring to all instances may be defined.

3.3. Structural Language Elements

The structural language elements of MSCs include all constructs which can be used to specify more general MSCs or to refine MSCs. The current MSC recommendation offers the *coregion* and the *submsc* constructs.

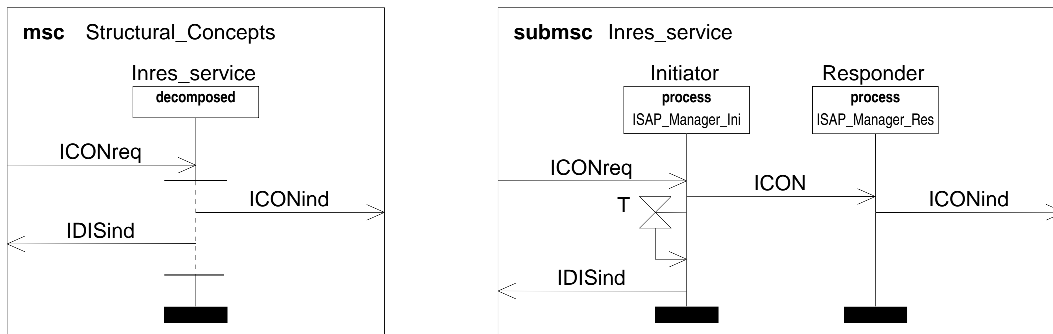
3.3.1. Coregion

Along an MSC instance, normally a total ordering of message events is assumed. This may not be appropriate for instances referring to a level higher than SDL processes. To cope with this, a *coregion* is introduced. A coregion is graphically represented by a dashed section of an MSC instance. Within a coregion, the specified communication events are not ordered. At present, only message events may be specified within a coregion, however, a coregion may contain an arbitrary mixture of message inputs and outputs (Figure 7).

3.3.2. Submsc

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.

An MSC instance can be refined by another MSC, which is then called *submsc*. By means of the keyword **decomposed** a submsc with the same name is attached to the re-



(a) MSC with coregion (b) Refinement of Inres_service in (a)

Figure 7. Sample application of structural concepts

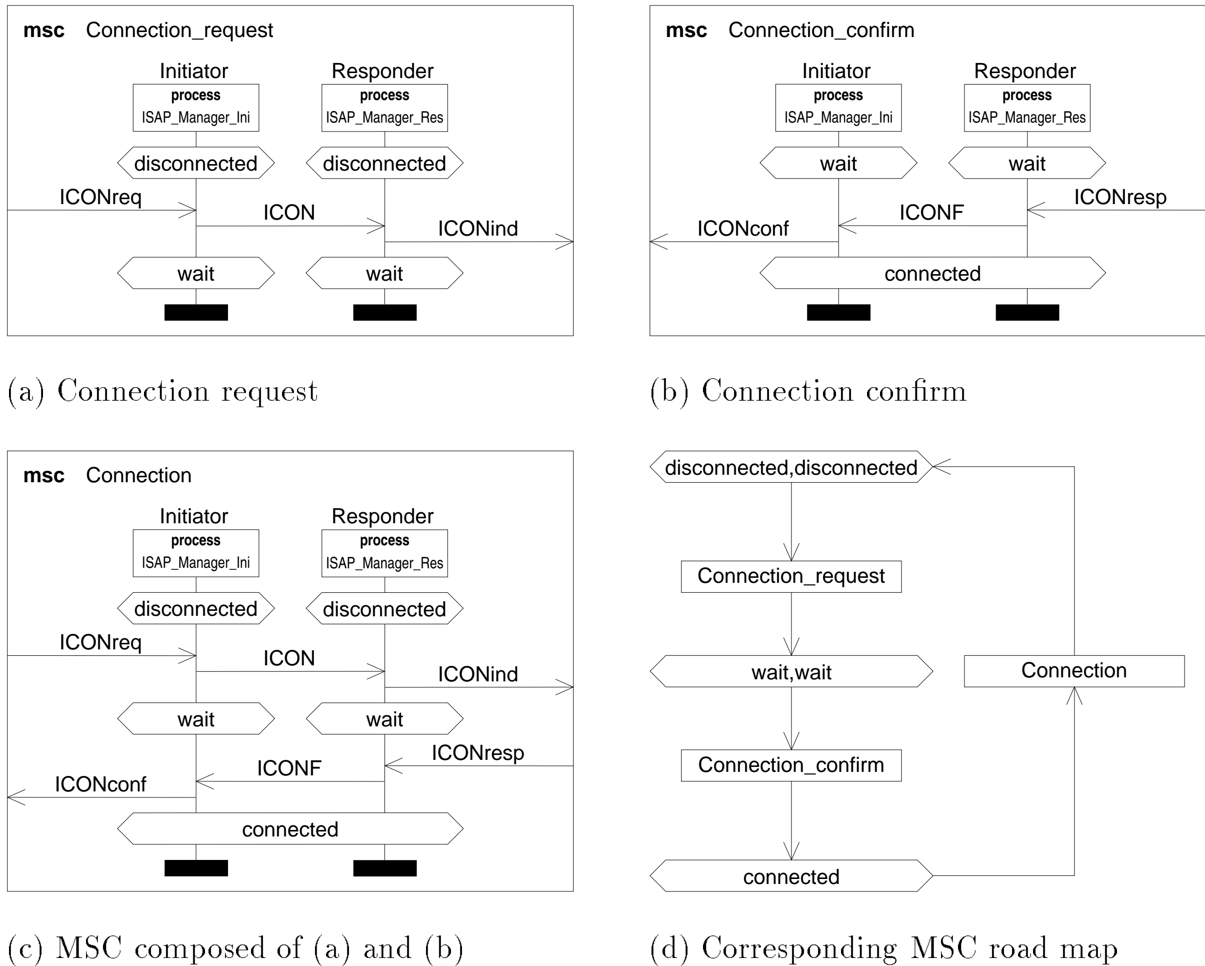
fined instance. The submsc represents a decomposition of this instance without affecting its observable behaviour. The messages addressed to and coming from the exterior of the submsc are characterized by the messages connected with the submsc border (frame symbol). Their connection with the external instances is provided by the messages sent and consumed by the corresponding decomposed instance, using message name identification. It must be possible to map the external behaviour of the submsc to the messages of the decomposed instance. The ordering of message events specified along the decomposed instance must be preserved in the submsc.

Actions and conditions within a submsc may be looked at as a refinement of actions and conditions in the decomposed instance. Contrary to messages, however, no formal mapping to the decomposed instance is assumed, i.e., the refinement of actions and conditions needs not obey formal rules. Figure 7 shows the refinement of the instance Inres_service by a submsc.

3.4. Composition of MSCs

Since one MSC only describes a partial system behaviour, it is advantageous to have a number of simple MSCs that can be combined in different ways. To determine possible combinations the already introduced (global and non-global) conditions are used employing certain composition and decomposition rules. The presented rules are not yet part of Z.120 [19] but it is intended to include them in future. Presently, they are part of the SDL methodology guidelines [18].

MSCs can be composed by name identification of final and initial (global or non-global) conditions. The other way round, MSCs can be decomposed at intermediate (global or non-global) conditions. Initial conditions denote the starting states, final conditions represent end states, and intermediate conditions describe arbitrary states within MSCs. An example of an MSC composition by means of non-global conditions is shown in Figure 8. The MSC *Connection* (Figure 8c) is a composition of the MSCs *Connection_request* (Figure 8a) and *Connection_confirm* (Figure 8b). Composition and decomposition of MSCs obey the rules for global and non-global conditions, given below. Here, global conditions refer to all instances involved in the MSC whereas non-global conditions are attached to a subset of instances.



(a) Connection request

(b) Connection confirm

(c) MSC composed of (a) and (b)

(d) Corresponding MSC road map

Figure 8. Composition and decomposition of MSCs

3.4.1. Composition by Means of Global Conditions

Two MSCs, *MSC1* and *MSC2*, can be composed if both MSCs contain the same set of instances and if the initial condition of *MSC2* corresponds to the final condition of *MSC1* according to name identification. The final condition of *MSC1* and the initial condition of *MSC2* become an intermediate condition within the composed MSC.

3.4.2. Composition by Means of Non-global Conditions

Two MSCs, *MSC1* and *MSC2*, can be composed by means of non-global conditions if for each instance which both MSCs have in common *MSC1* ends with a non-global condition and *MSC2* begins with a corresponding non-global condition. Corresponding to the MSC-composition, MSCs can be decomposed due to intermediate conditions.

4. Outlook

The MSC standardization activities during the ITU-T study period 1989-1992 have concentrated on the elaboration of the syntax and informal semantics for basic MSCs. A revised version of Z.120 containing enhancements and modifications is planned for 1996.

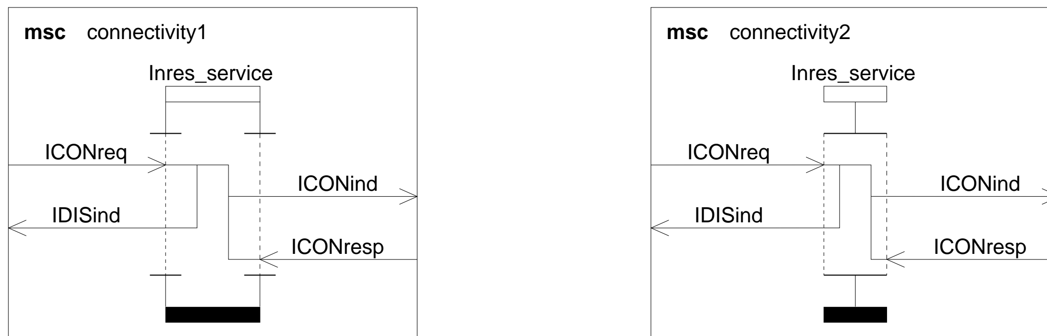


Figure 9. Causal relations defined by connections (extension of coregion)

Enhancements and modifications of MSCs can be classified with respect to basic MSCs and structural concepts. For MSC basic concepts, the elaboration of an enhanced textual syntax has been the main goal of the standardization activities, apart from the search for new more intuitive timer symbols (Section 3.2.4). New structural concepts mainly concern generalized causal ordering, MSC composition, and object-oriented modeling.

4.1. Generalized Coregion

The present Z.120 is restricted to total event ordering on MSC instances (normal case) and coregions which denote complete unordering of the contained events. Since MSC instances may refer to *higher order* entities like, e.g., SDL blocks or SDL systems, language constructs for the specification of more general causal orderings within one instance are demanded. Obviously, language constructs concerning generalized causal ordering are necessary also for decomposed instances and submscs.

Accordingly, dependencies (or lack of dependencies) between events as a generalization of the coregion construct have to be expressed. As a straightforward generalization, the coregion is enhanced by special symbols called *connections* denoting the causal ordering (Figure 9).

No event ordering is prescribed by the vertical dashed borders of the column itself. The connections define the ordering in an intuitive manner, they indicate the time direction from top to bottom (which is a generalization of the time ordering concept for instances in the present Z.120): Event e1 is causally ordered with respect to event e2 (e1>e2) if and only if e1 and e2 are connected and e2 can be reached from e1 by following the vertical connections in the direction from top to bottom only. Accordingly, in the example in Figure 9 we have the following causal ordering:

ICONreq<ICONind<ICONresp
ICONreq<IDISind

4.2. Submsc

The assignment of a submsc to a decomposed instance by means of name identification in general is too narrow. There are cases, were instances with the same name in different MSCs should be decomposed into different submscs. The decomposition clause may be extended by an optional term which names the MSC that defines the decomposition (Figure 10). At the same time, the term *submsc* becomes superfluous.

Figure 10. New decomposition clause

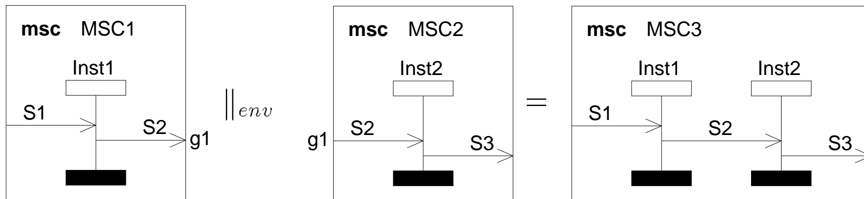


Figure 11. Environmental merge

4.3. MSC Composition

In addition to the sequential composition of MSCs based on conditions, real world telecom examples demand further composition operators, such as parallel composition, alternative composition, iteration and interrupt handling. As a main representative in the following, a special parallel merge operation with synchronization, the environmental merge, is sketched. The environmental merge may be understood as a *horizontal* merge operation. A special application could be the horizontal paging. Some other parallel merge operations, called synchronization merge and synchronization condition merge, are under consideration but not discussed here (cf. [16]).

The environmental merge operator $\|_{env}$ (Figure 11) identifies every message sent to or received from a gate in the environment of the first MSC *MSC1* with the message received from, respectively sent to, the equally named gate in the environment of the second MSC *MSC2*. The explicit definition of gates may be omitted for messages with unambiguous names in an MSC. The environmental merge operator then identifies the equally named messages to, respective from, the environment in both MSCs.

Composition techniques demand an intuitive means for the description of composition operations and the most obvious description technique is provided by MSC overview diagrams or road maps (Figure 8d). Road maps may include symbols denoting parallel merge, alternative merge and disruption.

MSC road maps may be used to represent all possible MSC compositions in a compact manner: the rectangles represent MSCs, the hexagons the initial and final conditions of these MSCs. There are three interpretations of road maps which are under discussion:

1. The road map only serves as an additional auxiliary diagram in order to provide a better overview about the MSCs contained in the MSC document. The road map does not contain information additional to the set of MSCs, i.e., the road map can be derived from the set of MSCs without additional information.

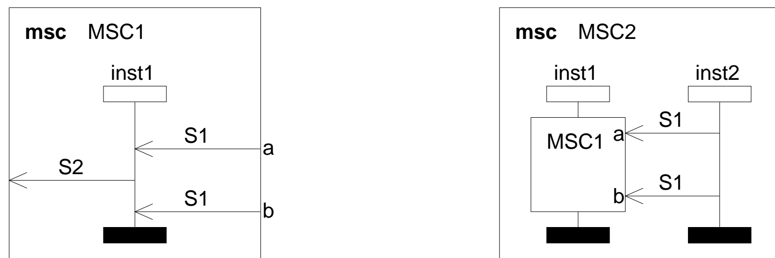


Figure 12. MSC type with gates

2. The road map actually defines the compositions of MSCs. The prescribed compositions only have to be in agreement with the conditions contained in the MSCs according to the definition of allowed combinations.

3. The compositions of MSCs are exclusively defined by the road map. Conditions have no meaning for composition.

4.4. Object oriented techniques

Object oriented techniques often provide a graphically more intuitive representation than pure composition techniques. In practice, a combination of pure composition techniques and object oriented techniques has proven to be most powerful.

One of the major achievements of SDL'92 compared with SDL'88 was the introduction of a more powerful scheme for handling recurring patterns. In SDL'92 these concepts were labeled as *object orientation*. It might be very helpful in the future use of the powerful combination of MSC and SDL to have a corresponding concept for MSC. The main reason for the introduction of an MSC type concept again is to have a means for handling recurring patterns (as an alternative for, e.g., a macro construct). MSC type instances are a compact way to express MSC composition. It is quite evident that *context parameters* can be introduced. Typically messages, i.e., actually message types, can be used as parameters. Possibly, instances (instance kind) can also be parameterized.

Each MSC can be seen as a definition of an MSC type. MSC types can be used in other MSC types. MSC types may be defined inside of an MSC or separately (Figures 12, 13)

An MSC type may be connected to its environment via message-gates. In the example in Figure 12, the MSC type *MSC1* is used within the MSC type *MSC2*, the letters *a* and *b* at the environment of *MSC1* and within the reference symbol to the MSC type *MSC1* in *MSC2* denote message gates. Gates are used to define the connection points when an MSC type is used in another type.

It is obvious that when the type concept has been established, it is not far to real object orientation including inheritance and virtuality. Inheritance means that one MSC type is a specialization of another MSC type. The idea behind virtuality is that virtual types enclosed in the general type may get a new definition in the specialization.

Virtual means that it is possible to adapt an MSC to special configurations or situations by redefining virtual MSC parts. E.g., the MSC type *MSCwithVirt* includes the inline definition of the virtual type *MSCvirt*. *MSCinheritance* inherits *MSCwithVirt* with the redefinition of *MSCvirt*. This redefinition means that *MSCinheritance* is obtained replacing *MSCvirt* in *MSCwithVirt* by the inline redefinition of *MSCvirt*.

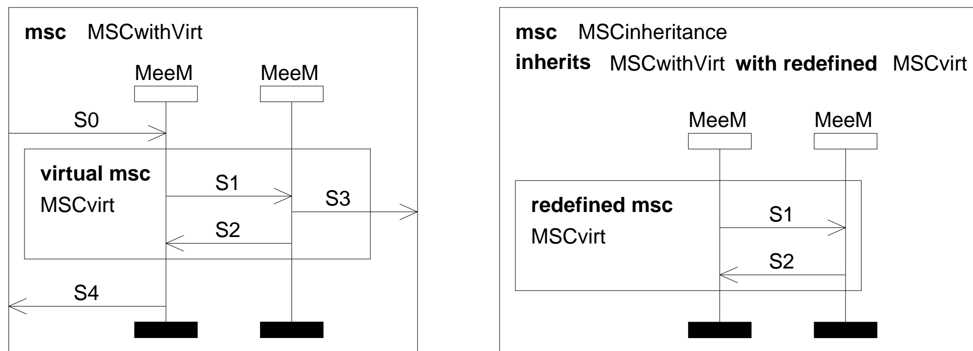


Figure 13. Virtual MSCs and redefinition

In comparison with pure composition techniques, the presented object oriented techniques have the advantage that MSCs which belong together are integrated within a larger frame from the very beginning. Whereas pure composition technique may easily end up in something like a puzzle, object oriented techniques could be compared with a large painting where all parts remain integrated within the context.

REFERENCES

1. B. Algayres, Y. Lejeune, F. Hugonnet, F. Hantz. *The AVALON-Project – A VALidatiON Environment for SDL/MSC Descriptions*. In: SDL'93 - Using Objects (O. Faergemand and A. Sarma, editors). North-Holland, Oct. 1993.
2. A. Ek. *Verifying Message Sequence Charts with the SDT Validator*. In: SDL'93 - Using Objects (O. Faergemand and A. Sarma, editors). North-Holland, Oct. 1993.
3. A. Ek. *Event-Oriented Textual Syntax*. TD44 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, Oct. 1994.
4. J. Grabowski, P. Graubmann, E. Rudolph. *The Standardization of Message Sequence Charts*. In: Proceedings of the IEEE Software Engineering Standards Symposium 1993. Sept. 1993.
5. J. Grabowski, D. Hogrefe, R. Nahm. *A Method for the Generation of Test Cases Based on SDL and MSCs*. Technical Report IAM-93-010, University of Berne, Institute for Informatics, April 1993.
6. J. Grabowski, D. Hogrefe, R. Nahm. *Test Case Generation with Test Purpose Specification by MSCs*. In: SDL'93 - Using Objects (O. Faergemand and A. Sarma, editors). North-Holland, Oct. 1993.
7. J. Grabowski, E. Rudolph. *Putting Extended Sequence Charts to Practice*. In: SDL'89 - The Language at Work (O. Faergemand and M. M. Marques, editors). North-Holland, Oct. 1989.
8. O. Haugen. *Case Studies: MSC and Structural Concepts*. TD17 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, Oct. 1994.
9. O. Haugen. *Structural Concepts in MSC. Report from Associate Rapporteur*. TD18 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, Oct. 1994.

10. D. Hogrefe. *OSI Formal Specification Case Study: The Inres Protocol and Service (revised).* Technical Report IAM-91-012, University of Berne, Institute for Informatics, May 1991, Update May 1992.
11. ITU-T SG 10 Q.9 (Rapporteur). *Correction List for Z.120 (I): Extensions and Modifications of Basic Concepts.* TD31 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, Oct. 1994.
12. I. Jacobson. *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, 1992.
13. F. Kristoffersen. *Message Sequence Chart and SDL Specification Consistency Check.* In: SDL'91 Evolving Methods (O. Faergemand and R. Reed, editors). North-Holland, 1991.
14. R. Nahm. *Consistency Analysis of Message Sequence Charts and SDL-Systems.* In: SDL'91 Evolving Methods (O. Faergemand and R. Reed, editors). North-Holland, 1991.
15. M. A. Reniers. *Syntax Requirements of Message Sequence Charts.* TD59 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, Oct. 1994.
16. E. Rudolph, P. Graubmann, J Grabowski. *Message Sequence Chart: Composition Techniques versus OO-Techniques - 'Tema con Variazioni'.* In: SDL'95 - Proceedings of the 7.th SDL Forum in Oslo, Norway (R. Braek and A. Sarma, editors). North-Holland, Sep. 1995.
17. Z.100 (1993). *CCITT Specification and Description Language (SDL).* ITU-T, June 1994.
18. Z.100 I (1993). *SDL Methodology Guidelines.* Appendix I to Z.100. ITU-T, July 1993.
19. Z.120 (1993). *Message Sequence Chart (MSC).* ITU-T, Sep. 1994.
20. Z.120 B (1995). *Message Sequence Chart Algebraic Semantics.* ITU-T Publ. sched., May 1995.