ELSEVIER

Computer Networks and ISDN Systems 28 (1996) 1629–1641

COMPUTER NETWORKS and ISDN SYSTEMS

Tutorial on Message Sequence Charts

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

[a] Technical University of Munich, Institute for Informatics, Arcisstrasse 21, D-80290 München, Germany
[b] Siemens AG, ZFE T SE, Otto-Hahn-Ring 6, D-81739 München, Germany
[c] Medical University of Lübeck, Institute for Telematics, Ratzeburger Allee 160, D-23538 Lübeck, Germany

Abstract

An introduction to the ITU standard language Message Sequence Chart (MSC) is provided. It is pointed out that MSC in many respects is complementary to the ITU specification and design language SDL. MSC in combination with SDL or other languages, now plays a role in nearly all stages of the system development process. Since MSC has been standardized in the same study group as SDL, the language form is quite analogous, e.g. it has a graphical (MSC/GR) and a textual (MSC/PR) syntax form. The MSC language in the present recommendation Z.120 (MSC'92), comprises basic language elements — instance, message, environment, action, timer, process creation and termination, condition — and structural language elements — "coregion" and "submsc". It is demonstrated how global and non-global conditions may be used for the composition of MSCs. Whereas in MSC'92 the main emphasis is put on the elaboration of basic concepts and a corresponding formal semantics, in the new MSC version (MSC'96) structural language constructs, essentially composition and object oriented concepts, will play a dominant role. With these new concepts, the power of MSC is enhanced considerably in order to overcome the traditional restriction of MSC to the specification of few selected system runs.

Keywords: MSC; SDL; Object oriented modelling; Composition techniques; System engineering; Requirement specification

1. Introduction

Message Sequence Charts (MSCs) are a widespread means for the visualization of selected system runs (traces) within communication systems. MSC 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 4th SDL Forum, October 1989, in Lisbon [7] and agreed upon at the ITU-meeting in Helsinki, June 1990 [4]. At the closing session of the ITU study period 1989–1992 in Geneva, May 1992, the new MSC

* Corresponding author. E-mail: rudolphe@informatik.tu-muenchen.de.

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 exact 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 Section 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 Fig. 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

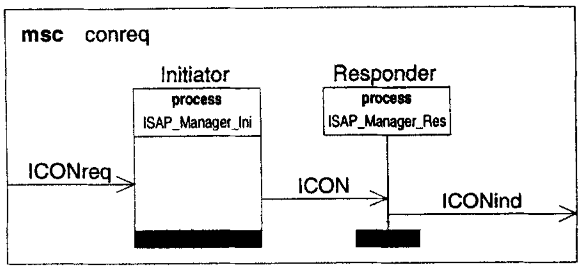


Fig. 1. Connection request.

(ICONind) to its user. The MSC is related to corresponding paths in SDL process diagrams (Fig. 2) where the path corresponding to the MSC in Fig. 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 Fig. 1 and Fig. 2 may serve to give a good intuitive 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 Fig. 1 is far more transparent than Fig. 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 Fig. 3.

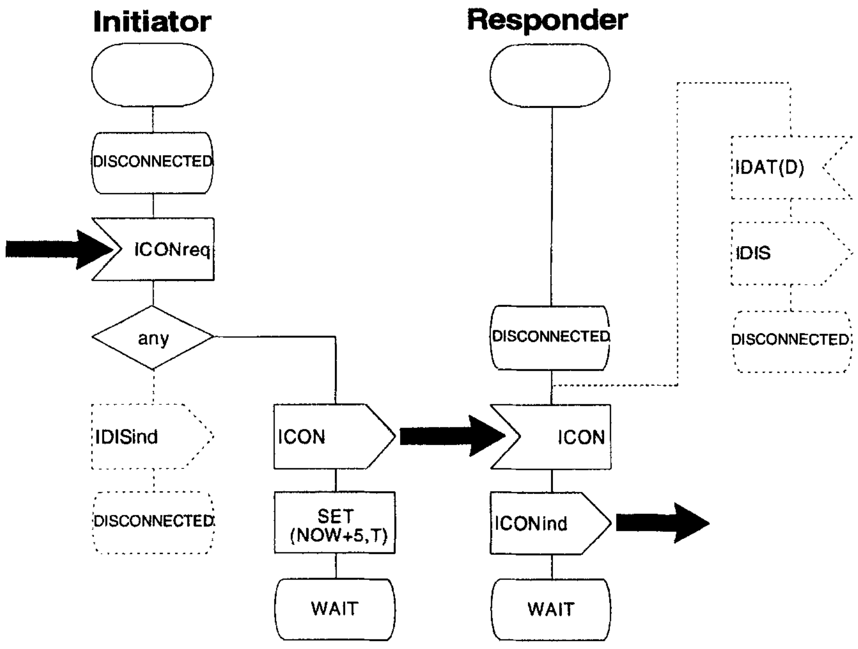


Fig. 2. SDL-diagram corresponding to the MSC in Fig. 1.

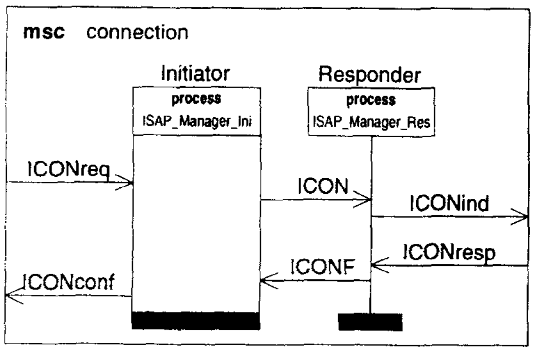
The MSC/PR presently contained in Z.120, lists message sending and receiving events in association with an instance. Recently, a better readable notation was requested, in particular in cases where MSC/PRs were not only used internally by tools, but also edited by humans. Thus, a new *event oriented* 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 Fig. 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 ICONF 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.

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



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

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

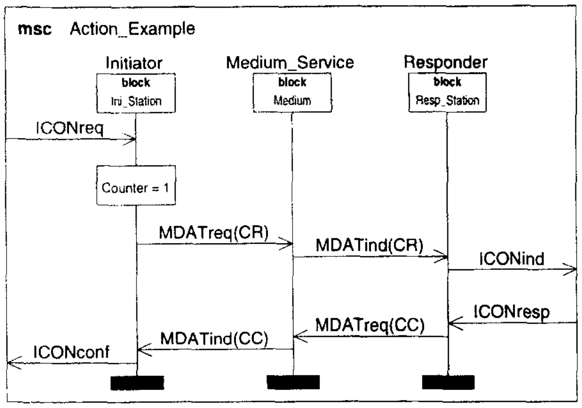


Fig. 4. Action.

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 message describing the communication events. In the graphical representation, instances are shown by vertical lines or, alternatively, by columns (Fig. 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 (Fig. 5(b)). 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 (Fig. 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 describing an internal activity of an instance may be specified. An action is graphically represented by a rectangle containing arbitrary text (Fig. 4).

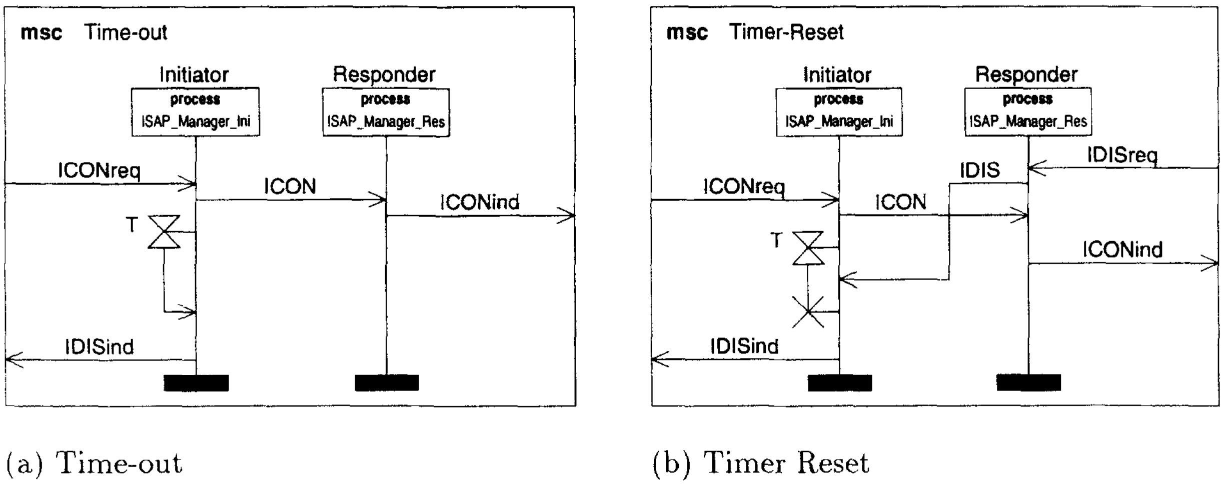


Fig. 5. MSC timer constructs.

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 Fig. 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. Timeout 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 run time. Consequently, a system designer needs features to describe such events. The corresponding MSC language elements are shown in Fig. 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 (Fig. 6).

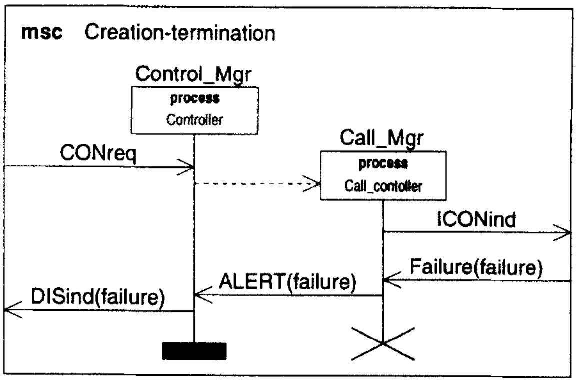


Fig. 6. Instance creation and termination.

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 (Fig. 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 MSC 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 (Fig. 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.

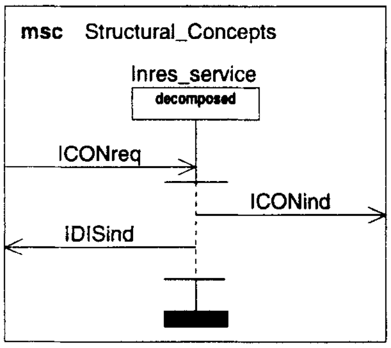
An MSC instance can be refined by another MSC, which is then called *submsc*. By means of the keyword **decomposed** a submsc with the same name is attached to the refined instance. The submsc represents a decomposition of this instance without affecting its observable behaviour. The messages addressed 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.

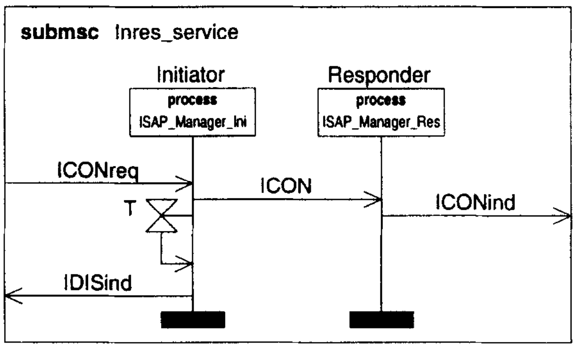
3.4. Composition of MSCs

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 may be used employing certain composition and decomposition rules which presently are part of the methodology guidelines [18]. According to the recent ITU-T work this composition may be defined within roadmaps (cf. Section 4).



(a) MSC with coregion



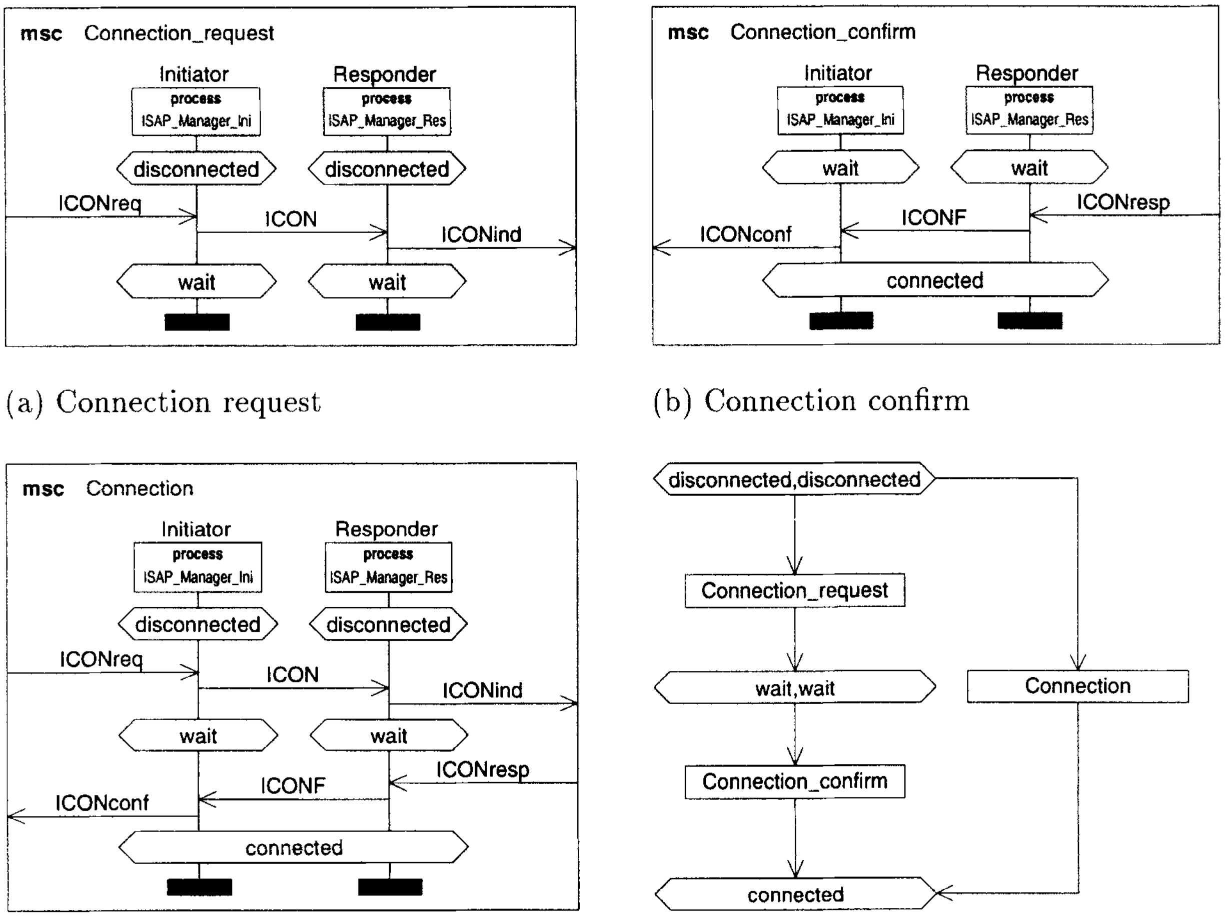
(b) Refinement of Inres_service in (a)

Fig. 7. Sample application of structural concepts.

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 Fig. 8. The MSC *Connection* (Fig. 8(c)) is a composition of the MSCs *Connection _request* (Fig. 8(a)) and *Connection _confirm* (Fig. 8(b)). 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.

3.4.1. Composition by means of global conditions

Two MSCs, *MSC1* and *MSC2*, can be composed sequentially if both MSCs contain the same set of instances and if the initial conditions 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.



(a) Connection request

(b) Connection confirm

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

(d) Corresponding MSC road map

Fig. 8. Composition and decomposition of MSCs.

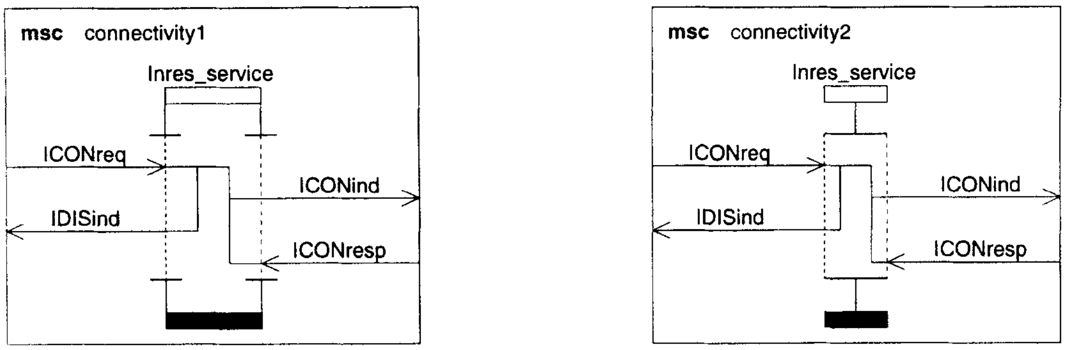


Fig. 9. Causal relations defined by connections (extension of coregion).

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. 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.

It should be noted that these language constructs are still under discussion in the ITU-T. Their selection and presentation within this tutorial, in many respects reflect the standpoint and the preferences of the authors.

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 level* entities like, e.g., SDL blocks or SDL systems, language constructs for the specification of more general causal orderings within

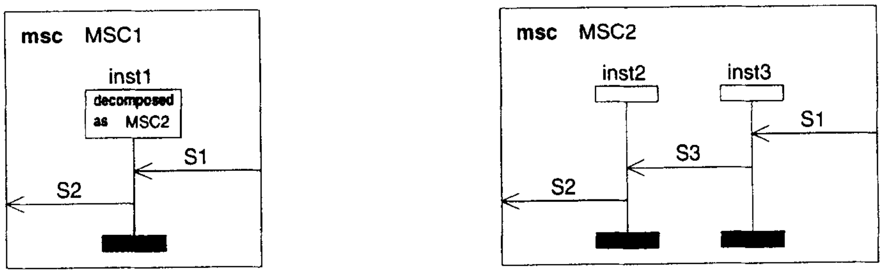


Fig. 10. New decomposition clause.

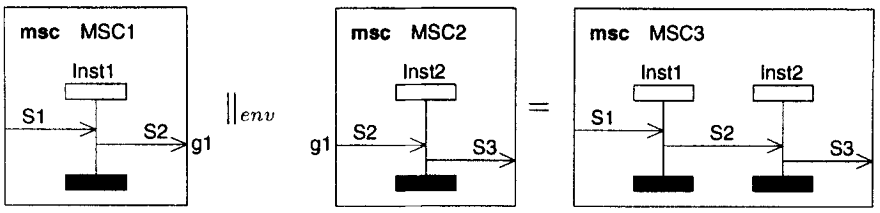


Fig. 11. Environmental merge.

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 (Fig. 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 Fig. 9 we have the following causal ordering:

ICONreq < ICONind < ICONresp and
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 (Fig. 10). At the same time, the term *submsc* becomes superfluous.

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

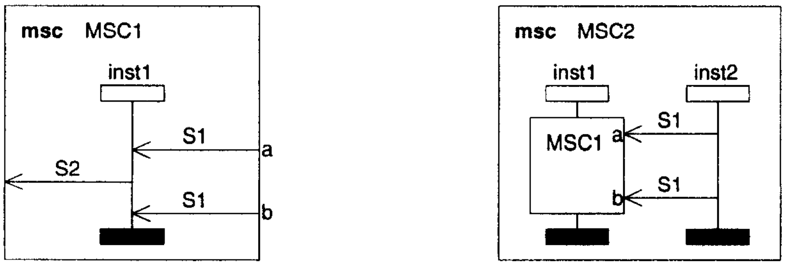


Fig. 12. MSC type with gates.

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 $\|_{env}$ (Fig. 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, respectively 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 (Fig. 8(d)). 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.
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.

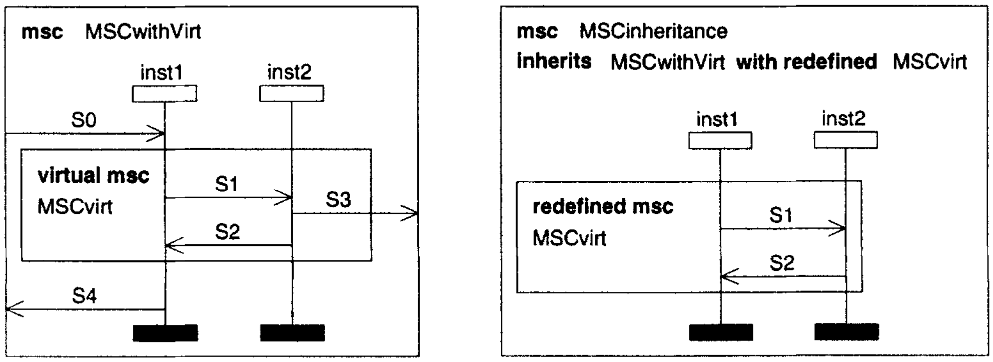


Fig. 13. Virtual MSCs and redefinition.

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 (Figs. 12 and 13).

An MSC type may be connected to its environment via message-gates. In the example in Fig. 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., in Fig. 13 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*.

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. In practice, the MSC type concept has turned out to be very useful, particularly in combination with composition techniques. On the other hand, typical object oriented techniques, i.e., inheritance and virtuality, need further studies before they may be integrated into the MSC language.

References

[1] B. Algayres, Y. Lejeune, F. Hugonnet and F. Hantz, The AVALON-Project–A VALidatiON environment for SDL/MSC descriptions, in: O. Faergemand and A. Sarma (Eds.), *SDL'93 – Using Objects*, North-Holland, Amsterdam (1993).

[2] A. Ek, Verifying message sequence charts with the SDT validator, in: O. Faergemand and A. Sarma (Eds.), *SDL'93 – Using Objects*, North-Holland, Amsterdam (1993).

[3] A. Ek. Event-Oriented Textual Syntax, TD44 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, October 1994.

[4] J. Grabowski, P. Graubmann and E. Rudolph, The standardization of message sequence charts, *Proc. IEEE Software Engineering Standards Symp. 1993*, September 1993.

[5] J. Grabowski, D. Hogrefe and 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 and R. Nahm, Test case generation with test purpose specification by MSCs, in: O. Faergemand and A. Sarma (Eds.), *SDL'93 – Using Objects*, North-Holland, Amsterdam (1993).

[7] J. Grabowski and E. Rudolph, Putting extended sequence charts to practice, in: O. Faergemand and M.M. Marques (Eds.), *SDL'89 – The Language at Work*, North-Holland, Amsterdam (1989).

[8] O. Haugen, Case Studies: MSC and Structural Concepts, TD17 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, October 1994.

[9] O. Haugen, Structural Concepts in MSC. Report from Associate Rapporteur, TD18 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, October 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, Geneva, October 1994.

[12] I. Jacobson, *Object-Oriented Software Engineering – A Use Case Driven Approach*, Addison-Wesley, Reading, Mass. (1992).

[13] F. Kristoffersen, Message sequence chart and SDL specification consistency check, in: O. Faergemand and R. Reed (Eds.), *SDL'91 – Evolving Methods*, North-Holland, Amsterdam (1991).

[14] R. Nahm, Consistency analysis of message sequence charts and SDL-systems, in: O. Faergemand and R. Reed, (Eds.), *SDL'91 – Evolving Methods*, North-Holland, Amsterdam (1991).
[15] M.A. Reniers, Syntax Requirements of Message Sequence Charts, TD59 (Question 9/10), ITU-T Study Group 10 Meeting in Geneva, October 1994.
[16] E. Rudolph, P. Graubmann and J Grabowski, Message sequence chart: composition techniques versus OO-techniques – 'Tema con Variazioni', in: R. Braek and A. Sarma (Eds.), *Proc. 7th SDL Forum*, Oslo, Norway, North-Holland, Amsterdam (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, September 1994.
[20] Z.120 B (1995), Message Sequence Chart Algebraic Semantics, ITU-T Publ. sched., May 1995.

Jens Grabowski studied Computer Science and Chemistry at the University of Hamburg, Germany, where he graduated with a diploma degree. During his studies he spend two years at the central research laboratory of the Siemens AG in Munich, Germany, where he focusses on protocol specification and protocol validation based on Petri Nets, SDL and MSC. From 1990 to October 1995 he was research scientist at the University of Berne, Switzerland, where he received his Ph.D. degree in 1994. In Berne he managed several research projects in the area of test case specification and test case generation for formally specified protocols. Since October 1995 Jens Grabowski is researcher and lecturer at the Institute for Telematics of the Medical University in Lübeck, Germany. His research activities are directed towards network analysis, formal methods in protocol specification and conformance testing.

Ekkart Rudolph got his diploma and doctoral degree in theoretical physics at the Max Planck Institute for Physics and Astrophysics (Werner Heisenberg Institute) in Munich. Since 1980 he was engaged in software engineering at Corporate Research and Development of Siemens AG in particular in the area of Petri Nets and SDL/MSC. Since 1990 he is responsible as rapporteur for the standardization of Message Sequence Charts within ITU (former CCITT). Since July 1995 he is guest scientist in the group of Manfred Broy at the Technical University of Munich.

Peter Graubmann studied Mathematics and Computer Science at the Technical University of Munich where he received his Diploma in 1980. He then joined Corporate Research and Development of Siemens AG in Munich. There he was involved in the development of system development methods and tools for distributed and parallel systems with focus on formal methods (mainly Petri nets, Trace Theories, CSP, SDL, MSC). Since 1993, he is involved in the international Telecommunications Information Networking Architecture (TINA) Project and spent recently one and a half year as resident researcher in New Jersey, US. Being back in Munich again, he continues to work in this field.