Message Sequence Chart (MSC) - A Survey of the new CCITT Language for the Description of Traces within Communication Systems

Jens Grabowski

Universität Bern
Institut für Informatik
Länggassstrasse 51
CH-3012 Bern

grabowsk@iam.unibe.ch

Ekkart Rudolph

Siemens AG München
ZFE BT SE 53
Otto-Hahn Ring 6
D-W8000 München 83

rudolph@ztivax.zfe.siemens.de

Abstract: Message Sequence Charts (MSCs) are a widespread means for description and graphical visualisation of selected system runs within distributed systems, especially telecommunication systems. Various kinds of MSCs with similar expressive power are used frequently within industry and standardisation bodies. Therefore, the CCITT (Comité Consultatif International Télégraphique et Téléphonique) attempts to harmonise their use by means of the new standard language *Message Sequence Chart (MSC)* in 1992 [Z120]. This paper presents a motivation for the MSC standardisation. The history of the standardisation process is briefly sketched. The MSC language is introduced. Some language constructs which may need further elaboration are pointed out and possible enhancements are proposed. Finally, an approach towards the definition of a clear MSC semantics is described.

Keywords: CCITT, Formal Description Technique (FDT), Distributed System, Message Sequence Chart (MSC), SDL, Trace Language

1 The motivation for the MSC standardisation

Within the software life cycle increasing attention is paid to the stages of specification and design since the quality of all following stages essentially depend on them. In particular, in the field of communication systems this has been taken into account by the development of standardised formal description techniques (FDTs) like SDL, Estelle, and LOTOS [Hog 89]. An FDT specification, however, is useful only if it is checked with respect to syntactic and particularly semantic correctness.

Apart from a general correctness proof (e.g. absence of deadlocks) the consistency of a FDT specification with respect to prescribed requirements has to be checked. A convenient way to describe such requirements is offered by system traces which are presented suitably in form of message flow diagrams called Message Sequence Charts (MSCs).

An MSC shows sequences of messages exchanged between entities (such as SDL services, processes, or blocks) and their environment (cf. fig. 3-2). Formally, an MSC describes the partial ordering of message events, i.e. message sending and message consumption.

MSCs have been used for a long time within international standardisation bodies (e.g. CCITT, ISO/IEC) and within industry, following different conventions under various names such as Arrow Diagrams [Q699], Extended Sequence Charts [GraRu 89], Information Flow Diagrams [Q65], Message Flow Diagrams [CCHK 90], or Time Sequence Diagrams [ISO 87]. These MSC variants mainly differ with respect to syntax and terminology. There are only minor semantic differences (cf. [Tog 92]) and therefore a standardisation was feasible.

The main reason for the MSC standardisation was to sustain tool support, to provide feasibility of MSC exchange between different tools, to ease the mapping to and from FDT specifications, and to harmonise their use within the CCITT Study Groups. The CCITT developed the MSC language definition [Z120], which was approved by the CCITT members in May 1992.

The new MSC recommendation defines the MSC syntax and includes an informal semantics. The informal semantics is given by means of relating MSCs to SDL specifications (cf. section 3.1). This is mainly due to the fact that the CCITT group which provided the MSC language definition also maintains SDL. Since our work is closely related to this group we also use SDL to explain the meaning of MSCs.

The new MSC recommendation includes two syntactical forms, MSC/GR as a graphical and MSC/PR as a pure textual representation (cf. section 3.2). In this paper we mainly rely upon MSC/GR representation since its symbols offer an intuitive understanding of their meaning.

The remaining part of the paper is organised in the following way: In chapter 2, the history of the MSC standardisation is described. The MSC language is introduced in chapter 3. Within chapter 4 and 5 possible language modifications and enhancements are discussed and future trends in MSC language development are described. In chapter 6 we explain an approach towards a clear MSC semantics and finally, a brief outlook is given.

2 The History of the MSC standardisation

Within the SDL user guidelines of 1988 [Z100-D] only a short section has been devoted to MSCs as one of the auxiliary diagrams though within an integrated tool set MSCs may very well play an important role. This was pointed out at the SDL Forum 1989 in Lisbon within the paper *"Putting Extended Sequence Charts to Practice"* [GraRu 89].

The terminology *Extended Sequence Charts* (ESC) was used for MSCs enhanced by SDL symbols and a few further constructs. ESCs were presented as a means for stepwise refinement and enrichment of MSCs from which finally SDL specifications may be derived. The role of MSCs and ESCs within the whole software life cycle, from requirement specification until test case specification, was pointed out.

Due to the great interest which MSCs found at this SDL Forum, their standardisation in graphical and textual representation within the CCITT was suggested. The standardisation was approved at the CCITT meeting in Helsinki, June 1990. It was decided there to first concentrate on the basic language constructs of MSCs, i.e. message flow diagrams without further extensions, e.g. by SDL symbols, and in particular to work out a clear semantics for them. One of the reasons for this restriction was to avoid too much overlap with SDL.

At the same meeting also a first contribution on the formalization of MSCs was presented (updated version [Til 91]). This formalization was focusing on equivalence relations for MSCs and on merging of instances within MSCs in order to provide a formal relationship between different levels of abstraction. It was pointed out in this contribution that by merging of instances a more general time ordering for events was obtained than originally defined for the basic language of MSCs. These early investigations have influenced the inclusion of higher level concepts contained in the final MSC recommendation [Z120].

In Helsinki the standardisation activities for MSCs still were intended to be part of the new *"SDL Methodology Guidelines"* [Belina 92], which were aiming at a guideline for the effective use of SDL. Soon it was recognised that the standardisation of MSCs would go beyond the SDL guidelines. It was also felt that MSCs should not be related only to SDL. Though it was not the intention to develop a fourth FDT, in addition to SDL, LOTOS, and ESTELLE, MSCs were looked at as another specification language which may be used in combination with other languages for system development. Consequently, at the next CCITT-meeting February 1991 in Geneva MSCs were chosen to become a separate recommendation.

At this meeting also the inclusion of further language constructs, e.g. *conditions* (representing system states), *timer constructs* and some higher level structural concepts, e.g. *macros* going beyond pure MSCs was agreed upon. (All mentioned MSC language elements are thoroughly explained within chapter 3.) The language constructs were adjusted to cover the needs of other CCITT recommendations employing Message-, Signal-, or Information-Flow Diagrams. In particular recommendation Q.65 [Q65]: *"Stage 2 of the method for the characterisation of services supported by an ISDN"* was involved. Q.65 is a provider for other recommendations in this area.

At the next CCITT meeting in Recife, December 1991,a first selection of higher level constructs took place, keeping *coregion*, *substructure*, *macro* and postponing the MSC language constructs for *remote procedure calls* and *grouping of instances* to the next study period. In addition, it was decided to include a *create* and *stop* of MSC instances.

The final session of CCITT Study Group X Geneva, May 1992 approved the new MSC recommendations with a few changes. In particular, *substructure* was renamed to *submsc* and the *macro* concept was found to be not yet mature enough and hence postponed to the next CCITT study period.

3 An introduction to the MSC language

Within this chapter the MSC language is introduced. First the meaning of MSCs is explained by relating them to SDL specifications, MSC/PR and MSC/GR are described, the basic and afterwards the structural language constructs are introduced.

3.1 The meaning of MSCs

MSCs show the message flow between entities like blocks, services, or processes. We explain the meaning of an MSC by relating it to SDL process diagrams (cf. [BHS 91], [Z100]). Let us consider the MSC in fig. 3-2 (a) which describes a selected trace piece of the connection set-up in the Inres service specification [Hog 92]. It could equally be represented using SDL process diagrams with certain additions and modifications (cf. fig. 3-1, dashes stand for not followed branches, bold arrows indicate the message flow).

The diagram in fig. 3-1 contains at least the same information as the MSC in fig. 3-2 (a). Within the MSC 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.

However, obviously the MSC is much more transparent, since it concentrates on the relevant information, namely the instances (*Initiator*, *Responder*) and the messages involved in the selected trace piece (*ICONreq*, *ICON*, *ICONind*). Beyond that, what is even more important, the relation of MSCs to an SDL specification may be rather sophisticated. The MSC instances very often represent collections of (SDL) processes on a higher level of abstraction such as blocks, thus, reflecting the stepwise development of a specification according to refinement strategies.

Generally, the relation between an MSC and an SDL specification can be characterised in the following way (for ACT cf. [Hog 88]):

"Each sequentialization of an MSC describes a trace from one equivalence class of nodes to another equivalence class of nodes of an Asynchronous Communication Tree (ACT) presenting the behaviour of an SDL specification."

In any case the correspondence between fig. 3-1 and fig. 3-2 (a) may serve to give a good intuitive idea about the meaning of an MSC. It also demonstrates that an MSC describing one possible scenario can be looked at as an SDL skeleton (cf. [GraRu 89], [Belina 92]).

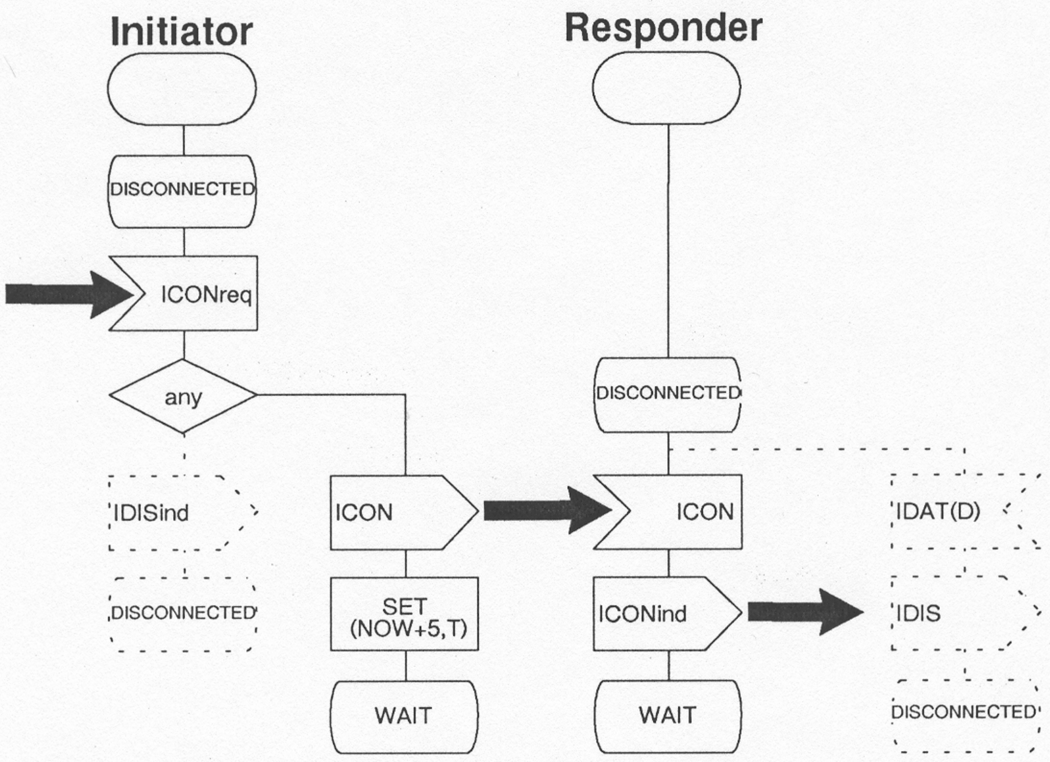


Figure 3-1: (Non-standard) combined SDL - message flow diagram

3.2 MSC/PR and MSC/GR

Analogous to the SDL recommendation [Z100] the new MSC recommendation 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 easily into a corresponding MSC/PR representation. The other way round the same problems arise as in SDL since MSC/PR (and SDL/PR) include no graphical information like height, width, or alignment of symbols and text. An example of the MSC/GR and the corresponding MSC/PR representation is shown in fig. 3-2.

3.3 Basic language constructs of MSCs

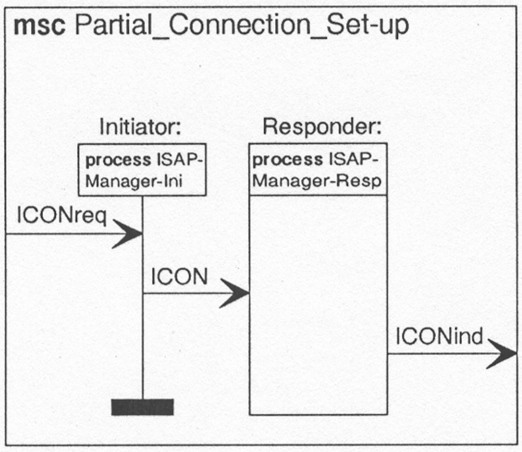
The basic language of MSCs includes all constructs which are necessary in order to specify the pure message flow. For MSCs these language constructs are *instance, message, action, setreset* (time supervision), *settime-out* (timer expiration), *stop, create* and *condition.*

3.3.1 Instance and message

The most basic 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 represented by vertical lines or alternatively by columns (fig. 3-2 (a)). 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 horizontal arrows with a possible bend to admit message overtaking or crossing (e.g. fig. 3-3 (a)). The head of the message arrow denotes the message consumption, the opposite end the message sending. In addition to the message name, message parameters in parentheses may be assigned to a message.

Along each instance axis (column) 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.



(a) MSC in MSC/GR representation

```
msc Partial_Connection_Set-up;
inst Initiator, Responder;

  instance Initiator: process ISAP-Manager-Ini;
     in ICONreq from env;
     out ICON to Responder;
  endinstance;

  instance Responder: process ISAP-Manager-Resp;
        in ICON from Initiator;
        out ICON to env;
  endinstance;

endmsc;
```

(b) MSC of (a) in MSC/PR representation

Figure 3-2: MSC in MSC/PR and in MSC/GR representation

3.3.2 System environment

Within an MSC the system environment is represented by the frame symbol which forms the boundary of an MSC diagram (cf. fig. 3-2, 3-3). Contrary to instances, no ordering of communication events is assumed.

3.3.3 Actions and timer constructs

Within one MSC it is possible to indicate *actions* and timer handling. An action is represented by a rectangle containing an arbitrary text. The timer handling contains two constructs: 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 setting of a timer is represented by a small rectangle, whereas *time-out* and *reset* are described by special timer arrows. A timer arrow starts at a corresponding set symbol (rectangle) and ends below at the same instance. A textual timer description (e.g. name and duration) may be associated with the arrows. To each set a corresponding time-out or reset has to be specified and vice versa. Action and timer constructs are shown within fig. 3-3.

3.3.4 Instance stop and instance creation

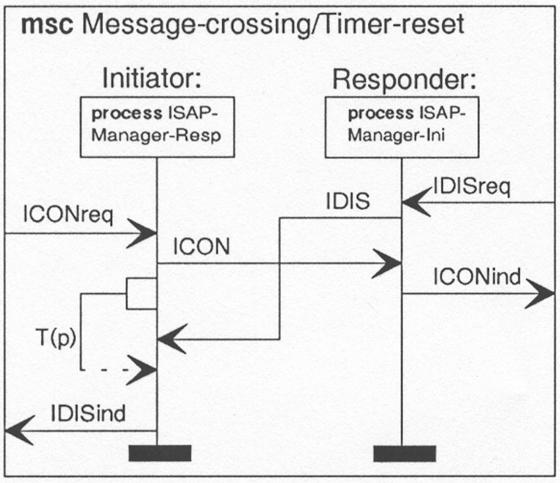
Creation and termination of instances within communication systems are quite common events. This is due to the fact that most communication systems are dynamic systems where instances appear and disappear during system lifetime. Consequently, a system designer needs features to describe such events.

The corresponding MSC language elements are shown in fig. 3-3 (d). The *create* symbol is a dashed arrow which may be associated with textual parameters. A create arrow originates from a father instance and points at the instance head of the child instance. The termination of an instance graphically is represented by a cross (*stop* symbol) at the end of the instance axis.

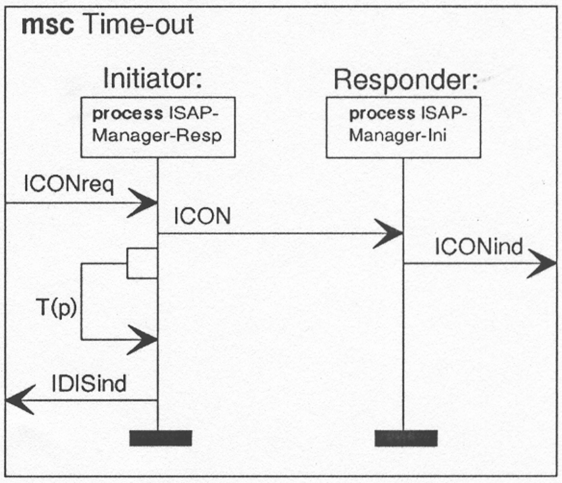
3.3.5 Conditions

A *condition* either describes a global system state referring to all instances contained in the MSC (*global condition*) or a state referring to a subset of instances (*nonglobal condition*). Conditions can be used to emphasise important states within an MSC or for the composition and decomposition of MSCs (see chapter 5).

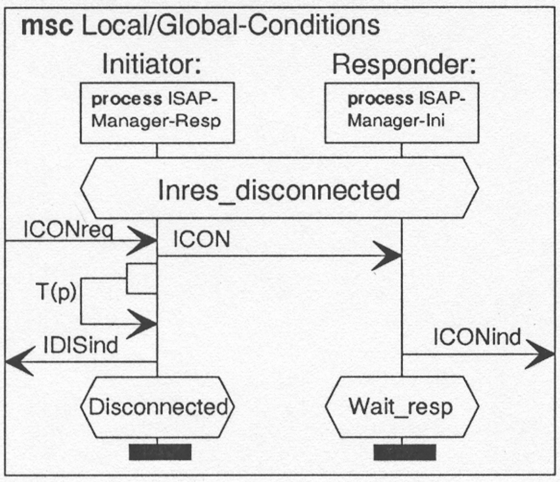
In the MSC/GR representation global and nonglobal conditions are represented by hexagons covering the involved instances (cf. fig. 3-3 (c), (e)). In fig. 3-3 (e) the instance *Medium_service* is not covered by the condition *Disconnected* and therefore it is not involved in the state to which the condition refers.



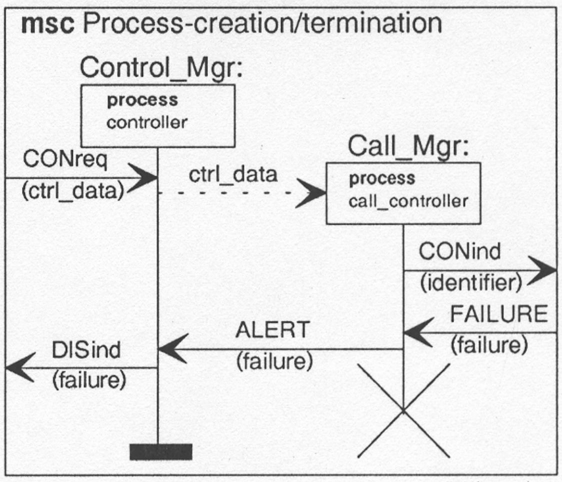
(a) MSC with message crossing (IDIS,ICON) and *set→reset* (time supervision)



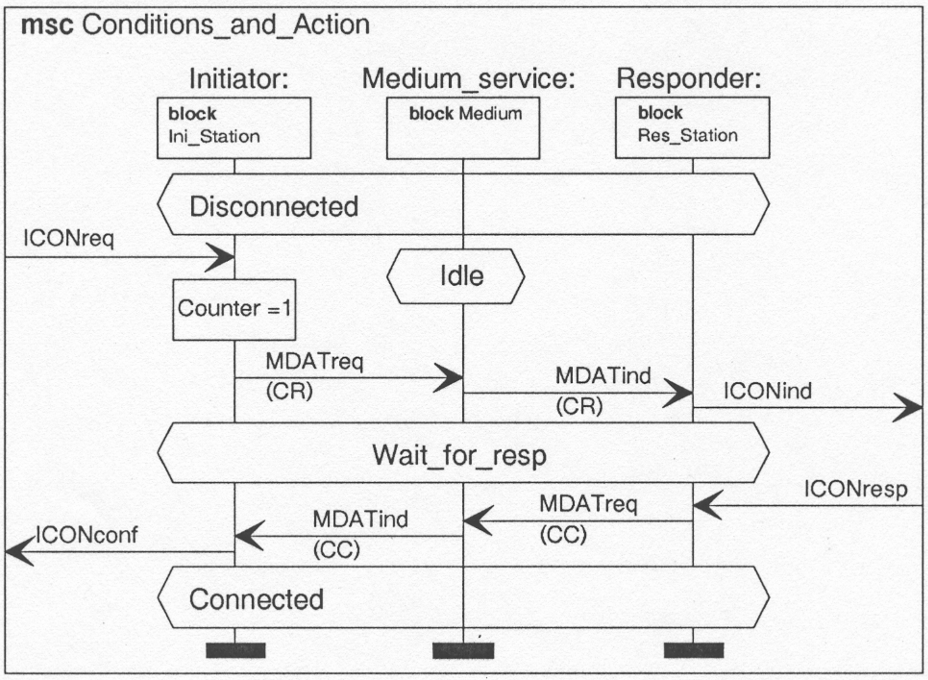
(b) MSC with *set→time-out* (timer expiration)



(c) MSC with global and local *conditions*



(d) MSC with *create* and *stop*



(e): MSC with *conditions* and *action*

Figure 3-3: MSCs with basic language constructs

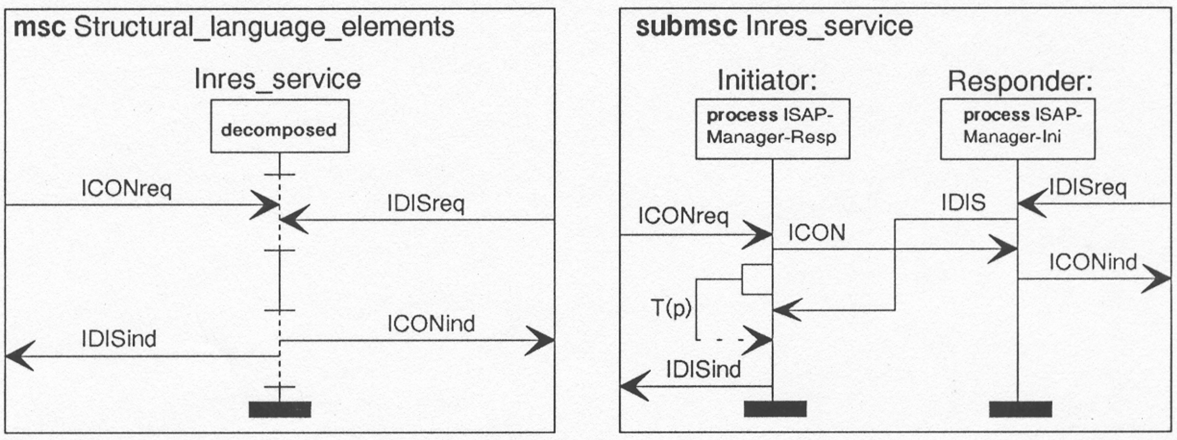
In the MSC/PR representation conditions are introduced at two different places: on the level of MSCs in form of global conditions and on the level of instances referring to an arbitrary set of instances. In the second case the condition may be local, i.e. attached to just one instance. If the condition refers to several instances then the keyword *shared* together with an instance list denotes the set of instances to which the condition is attached. By means of the keywords *shared all,* also in the second case a condition referring to all instances may be defined. However, for a clear structuring of an MSC in MSC/PR representation, the syntax for global conditions may preferably be put at the beginning and at the end of an MSC.

3.4 Structural language elements of MSCs

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

3.4.1 The coregion

Along an MSC instance message events are totally ordered. This may be not appropriate for instances referring to a higher level than SDL processes. Therefore a so-called *coregion* is introduced. A coregion denotes a piece of an MSC instance where the specified communication events are not ordered. Within one coregion only sending (origins of message arrows) or only consumption events (arrow heads) may be specified. Examples containing coregions are given in fig. 3-4 (a) and fig. 4-4 (b).



(a) MSC with *coregions* (b) Refinement of *Inres_service* in (a)

Figure 3-4: MSCs with structural language elements

3.4.2 Refinement of MSCs (submsc)

An MSC instance can be refined by another MSC, which then is called *submsc*. A submsc is attached to the refined instance by means of the keyword *decomposed*. The submsc represents a decomposition of this instance without affecting its observable behaviour. The messages addressed to and comming from the exterior of the submsc are characterised 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 need not obey formal rules. In fig. 3-4 (b) the refinement of the instance *Inres_service* (fig. 3-4 (a)) is shown.

4 Modifications and possible enhancements of MSCs

Within this chapter, we propose some modifications and enhancements of the MSC language. Some of the proposals are very specific and some are rather general. However, this chapter reflects our ideas concerning the future development of MSCs.

4.1 Modifications of the MSC language

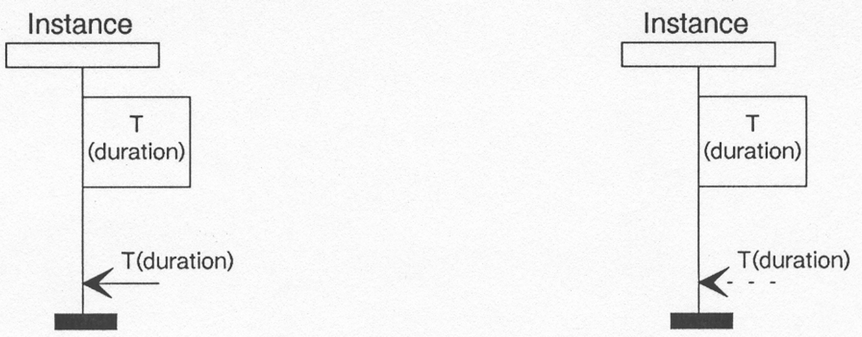
The proposed modifications of the current MSC language concern *timer handling* and *conditions*.

4.1.1 Timer handling

Within the MSC recommendation the timer handling contains two constructs: setting of a timer and subsequent timer expiration (time-out situation) or setting of a timer and subsequent timer reset (time supervision).

Contrary to SDL, currently there are no separate language constructs for timer *set, reset,* and *time-out* for MSCs. This is in agreement with the common practice in industry where usually the complete time-out situation or the time supervision is specified.

For an extensive use of MSC composition mechanisms (cf. chapter 5) this kind of timer handling, however, may be too narrow. Within one MSC e.g. it may be desirable to specify only the timer setting. There is no problem in MSC/PR to split the present MSC timer constructs into separate timer actions. In the graphical representation the rectangle symbol may represent a separate timer set if timer name and (optionally) timer duration is assigned. A separate timer expiration may be represented by an message arrow where the origin is not connected to a timer set rectangle. Correspondingly, a separate timer reset may be indicated by a dashed arrow where the origin is not connected to a timer set rectangle. An example of this proposal is given in fig. 4-1.



(a) Separate *set* and corresponding *time-out* construct

(b) Separate *set* and corresponding *reset* construct

Figure 4-1: (Non-standard) separate *set, time-out, reset* constructs

4.1.2 Conditions

The MSC/PR representation distinguishes between *global* and *nonglobal conditions* (cf. section 3.3.5), although it is possible to describe global conditions by means of nonglobal conditions and although there exists only one graphical symbol for conditions. In this case the MSC/PR and the MSC/GR representation are somehow divergent, because there are two MSC/PR concepts for the same MSC/GR symbol. The reason for introducing this redundancy was to shorten the textual description of MSCs by means of global conditions.

In order to avoid the mentioned redundancy without loosing the elegance of global conditions, we propose to replace the global condition in the MSC/PR representation by introducing an optional *condition declaration area* before the instance descriptions. Within the *condition declaration area* all nonlocal conditions (conditions which must be known by more than one instance) have to be declared. Within the instance description the declared conditions can be referred to.

As an example for our proposal we translate the MSC in fig. 3-3 (e) which includes global and nonglobal conditions into MSC/PR according to the MSC recommendation (fig. 4-2 (a)) and into a variant according to our proposal (fig. 4-2 (b)).

4.2 Enhancements of the MSC language

In this section we propose some new MSC language elements which may be included later in the MSC recommendation. The proposed MSC constructs concern *macros*, *synchronous communication*, *instance grouping*, *coregion*, and *data types*.

```
msc Conditions_and_Action;
inst Initiator, Medium_service, Responder;
  instance Initiator: block Ini_Station;
    condition Disconnected shared Responder;
    in ICONreq from env;
    action Counter=1;
    out MDATreq(CR) to Medium_service;
  endinstance;
  instance Medium_service: block Medium;
    condition Idle;
    in MDATreq(CR) from Initiator;
    out MDATind(CR) to Responder;
  endinstance;
  instance Responder: block Res_Station;
    condition Disconnected shared Initiator;
    in MDATind(CR) from Medium_service;
    out ICONind to env;
  endinstance;
  condition wait_for resp;
  instance Initiator: block Ini_Station;
    in MDATind(CC) from Medium_service;
    out ICONconf to env;
    condition Connected shared Initiator;
  endinstance;
  instance Medium_service: block Medium;
    in MDATreq(CC) from Responder;
    out MDATind(CC) to Initiator;
  endinstance;
  instance Responder: block Res_Station;
    in ICONresp from env;
    out MDATreq(CC) to Medium_service;
    condition Connected shared Initiator;
  endinstance;
endmsc;
```

(a) MSC/PR description of fig. 3-3 (e) according to [Z120]

```
msc Conditions_and_Action;
inst Initiator, Medium_service, Responder;
condition Disconnected shared Initiator, Responder;
condition Wait_for_resp shared all;
condition Connected shared Initiator, Responder;
  instance Initiator: block Ini_Station;
    condition Disconnected;
    in ICONreq from env;
    action Counter=1;
    out MDATreq(CR) to Medium_service;
    condition Wait_for_resp;
    in MDATind(CC) from Medium_service;
    out ICONconf to env;
    condition Connected;
  endinstance;
  instance Medium_service: block Medium;
    condition Idle;
    in MDATreq(CR) from Initiator;
    out MDATind(CR) to Responder;
    condition Wait_for_resp;
    in MDATreq(CC) from Responder;
    out MDATind(CC) to Initiator;
  endinstance;
  instance Responder: block Res_Station;
    condition Disconnected;
    in MDATind(CR) from Medium_service;
    out ICONind to env;
    condition Wait_for_resp;
    in ICONresp from env;
    out MDATreq(CC) to Medium_service;
    condition Connected;
  endinstance;
endmsc;
```

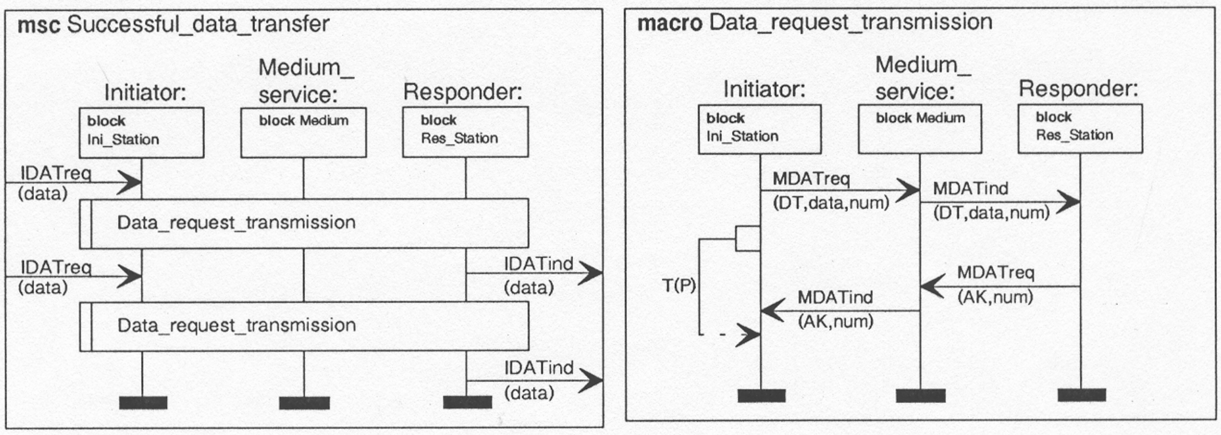
(b) (Non-standard) MSC/PR description of fig. 3-3 (e)

Figure 4-2: Standard and non-standard MSC/PR descriptions of fig. 3-3 (e)

4.2.1 Structuring and modularisation of MSCs (macros)

Macros may be introduced as a means for structuring and modularisation of MSCs and for the reusability of sections of MSCs. A *macro definition* is a section of an MSC which is defined outside of the MSC, yet within the MSC document, and which is inserted at the places where it is called. The macro definition essentially has the structure of an MSC. Apart from the keyword *macro*, in the graphical representation (MSC/GR) the macro definition symbol looks like the MSC frame symbol. The macro call may be indicated graphically by an SDL macro call symbol attached to the instances which are involved (cf. fig. 4-3). A system analysis of

MSCs containing macros is possible without macro expansion (as long as the macro is not contained in a submsc) if the syntax definition excludes messages leaving and entering the macro to and from (macro-)external instances.



(a) MSC with *macro call*

(b) *Macro definition* of the *macro call* in (a)

Figure 4-3: (Non-standard) MSC *macro call* and corresponding *macro definition*

4.2.2 Synchronous Communication (remote procedure calls)

Corresponding to SDL, a *remote procedure call* (RPC) may be introduced in order to indicate that a client instance calls a procedure within another instance. Graphically, it may be represented by an SDL procedure call symbol, attached to the client instance and connected with a message arrow pointing to the server instance. The RPC employs a synchronous communication mechanism. Independently of RPCs, a construct for *synchronous communication* is demanded by users that in contrast to RPCs, may be graphically represented by a bi-directional arrow.

4.2.3 Instance Grouping

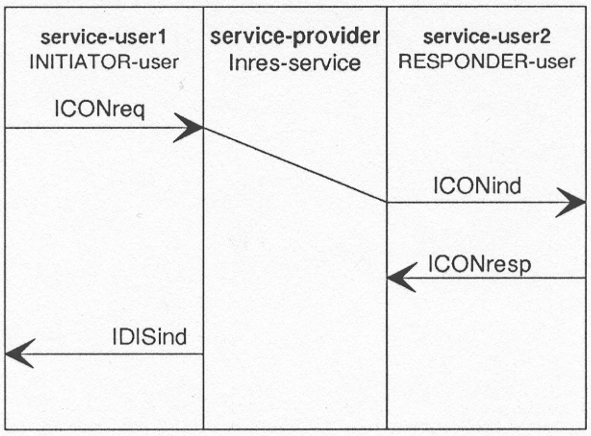
Further hierarchical or functional structuring of MSCs by means of *instance grouping* is required (cf. [Q65]). In particular, it may be helpful to indicate the assignment of process instances to blocks within an MSC. Graphically, instance grouping can be denoted by a horizontal bracket.

4.2.4 Weakening the time ordering along MSC instances (coregion)

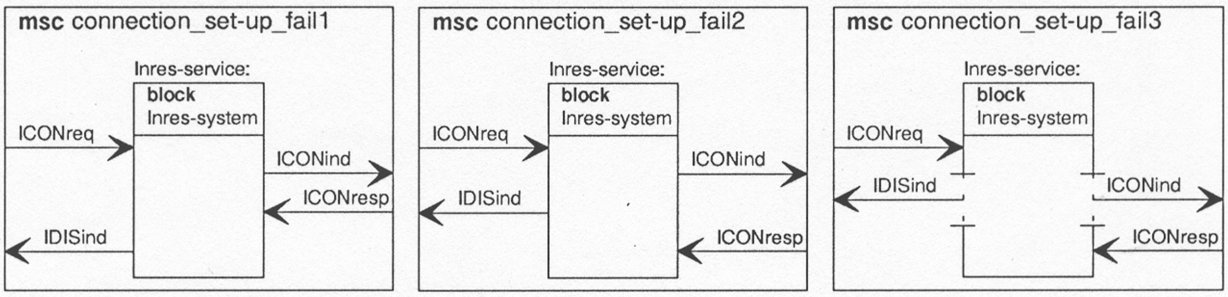
Within the present MSC language the total ordering along one instance axis can be weakened by using the *coregion* construct. Yet this construct presents only one possibility for weakening the event ordering along an MSC instance. In *Time Sequence Diagrams* (TSDs) [ISO

87], which can be looked at as a special kind of MSCs and which are frequently used to describe OSI services, another possibility is offered.

The ordering between message events at two service access points (SAPs) of one service provider can be indicated by diagonal lines (fig. 4-4 (a)). Each SAP is modelled by a single axis and along this axis the message events are totally ordered. Without going into details it should be noted that in general it is not possible to translate one TSD into one MSC (cf. fig. 4-4), since there is no construct, equivalent to the synchronisation primitive (diagonal line) of TSDs in the MSC language. If MSCs should be used for the specification of OSI services, the next CCITT study period must examine further features for the weakening of the total ordering of message events along one MSC instance.



(a) TSD describing an unsuccessful connection establishment of the Inres service



(b) Three MSCs describing the same traces as the TSD in (a)

Figure 4-4: TSD and three corresponding MSCs which describe the same traces.

4.2.5 Inclusion of data types

Within the present MSC recommendation no formal data description is provided. This is appropriate for the employment of MSCs in early stages of system development in order to provide a semiformal specification of communication.

However, for the usage of MSCs within later stages of design and implementation, in particular for system simulation and validation, and for selection and specification of test cases a formal data description by means of ADTs or ASN.1 is necessary. The formal data description may refer to conditions, actions, and to parameters of messages, macros, and timers.

5 Composition and decomposition rules for MSCs

The already mentioned MSC language enhancements increase the expressive power within one MSC. 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 nonglobal) conditions can be used employing certain *composition and decomposition rules*.

5.1 The meaning of composition and decomposition of MSCs

MSCs can be composed by name identification of *final* and *initial* (global or nonglobal) conditions. The other way round, MSCs can be decomposed at *intermediate* (global and nonglobal) conditions.

Initial conditions denote the starting states, final conditions represent end states, and intermediate conditions describe arbitrary states within MSCs. The terms initial, intermediate and final conditions are only used in order to simplify this description, they are not introduced within the MSC recommendation. An example of an MSC composition by means of global conditions is shown in fig. 5-1. The MSC *Complete_system_run* (c) is a composition of the MSCs *Connection_set_up* (a) and *Data_transfer/connection_release* (b).

Composition and decomposition of MSCs obey the subsequent rules for global and nonglobal conditions, whereby global conditions refer to all instances involved in the MSC whereas nonglobal conditions are attached to a subset of instances (cf. section 4.1.2).

5.2 Composition of MSCs

5.2.1 Composition by means of global conditions

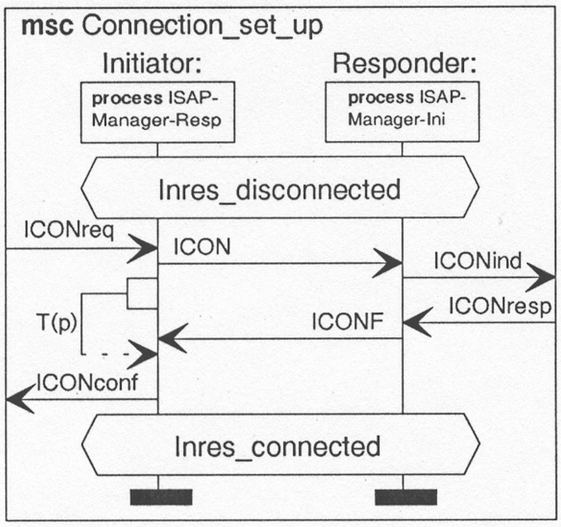
Two MSCs *MSC1* and *MSC2* can be composed if both MSCs contain the same set of instances and if the initial condition of *MSC2* corresponds to the final condition of *MSC1* according to name identification (cf. fig. 5-1). The final condition of *MSC1* and the initial condition of *MSC2* become an intermediate condition within the composed MSC. Symbolically:

(1) *MSC1 = MSC1' Condition*

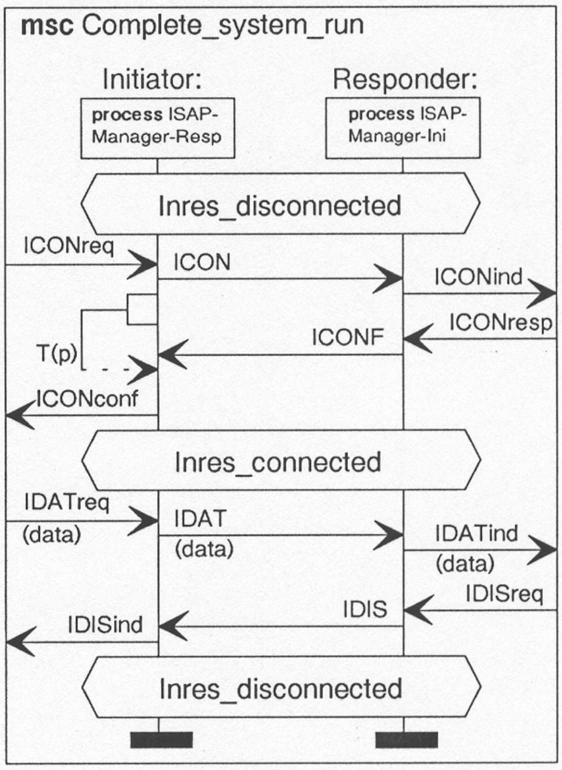
(2) *MSC2 = Condition MSC2'*

(3) *MSC1 * MSC2 = MSC1' Condition MSC2'*

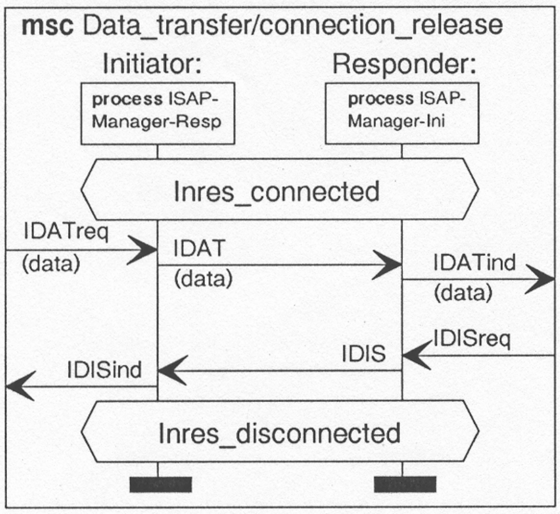
Equation (1) shall denote that *MSC1* can be written as an MSC section *MSC1'* and a subsequent final condition *Condition*. The second equation (2) denotes that *MSC2* starts with the initial condition *Condition* which is followed by the MSC section *MSC2'*. Equation (3) denotes the composition of *MSC1* and *MSC2* (using the asterisk symbol for composition). The composed MSC can be written in form of a starting MSC section *MSC1'*, an intermediate condition *Condition* and a subsequent MSC section *MSC2*.



(a) Successful connection set up of the Inres service



(c) Composition of (a) and (b)



(b) Data transfer and normal disconnection of the Inres service

Figure 5-1: MSC composition by means of *global conditions*

5.2.1 Composition by means of nonglobal conditions

Two MSCs *MSC1* and *MSC2* can be composed by means of nonglobal conditions if for each instance (I) which both MSCs have in common *MSC1* ends with a nonglobal condition and *MSC2* begins with a corresponding nonglobal condition. In addition each nonglobal condition of *MSC2* must have a corresponding nonglobal condition in *MSC1*. If I(*MSCi*) (i = 1,2) denotes the restriction of an *MSCi* to the events of instance I, this can be written symbolically:

(1) I(*MSC1*) = I(*MSC1*)' *Condition*

(2) I(*MSC2*) = *Condition* I(*MSC2*)'

(3) I(*MSC1*) * I(*MSC2*) = I(*MSC1*)' *Condition* I(*MSC2*)'

An example is given in fig. 5-2. The MSC *Connection_failure* (c) is a composition of the MSCs *Response_failure* (a) and *Request_failure* (b) via the local condition *Disconnected*. The MSC *Response_failure* contains two instances *Initiator* and *Responder*. The MSC *Request_failure* contains only one instance *Initiator* to which the initial local condition *Disconnected* is attached. The composition of MSC *Response_failure* with MSC *Request_failure* only refers to the instance *Initiator*, i.e. MSC *Response_failure* is continued along instance *Initiator* by MSC *Request_failure*. This also shows the usefulness of nonglobal conditions which makes a composition with respect to a subset of the instances involved in the MSCs possible. Finally, it should be noted that conditions with identical names are discriminated by the instances to which they are attached.

5.3 Decomposition of MSCs

Corresponding to the MSC-composition, MSCs can be decomposed due to intermediate conditions.

5.3.1 Decomposition by means of global conditions

An intermediate condition defines a possible MSC decomposition by splitting an MSC MSC1 at the intermediate condition *Condition* into *MSC2* and *MSC3*, the intermediate condition being converted into a final condition for *MSC2* and an initial condition for *MSC3*:

(1) *MSC1* = *MSC2' Condition MSC3'*

(2) *MSC2* = *MSC2' Condition*

(3) *MSC3* = *Condition MSC3'*

5.3.2 Decomposition by means of nonglobal conditions

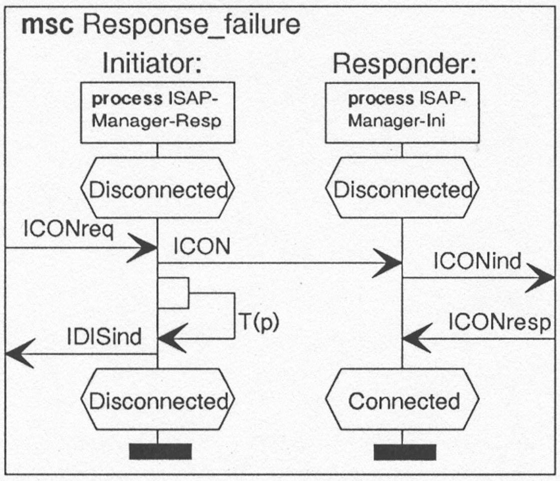
A subset of intermediate nonglobal conditions allows a decomposition of an MSC *MSC1* into *MSC2* and *MSC3* if all nonglobal conditions of this subset refer to different instances and no message is cut into pieces by means of the decomposition, i.e. both message input and the corresponding output belong to either *MSC2* or *MSC3*:

(1) I(*MSC1*) = I(*MSC2*)' *Condition* I(*MSC3*)'

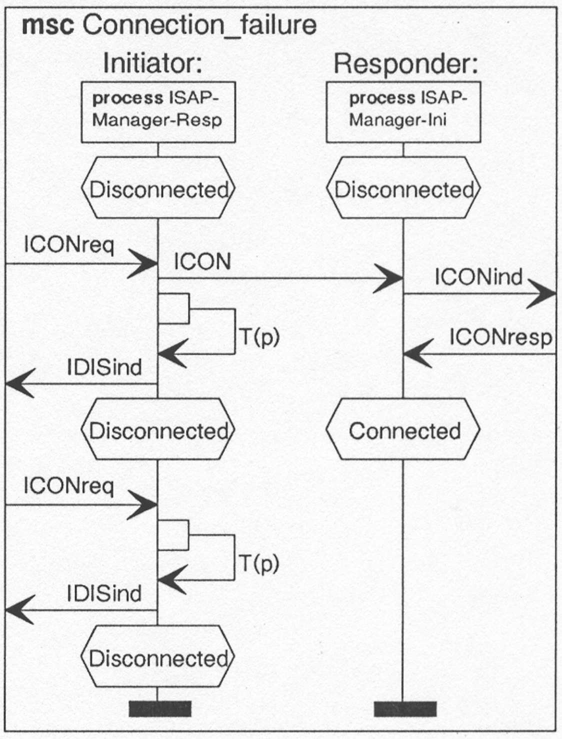
(2) I(*MSC1*) = I(*MSC1*)' *Condition*

(3) I(*MSC2*) = *Condition* I(*MSC2*)

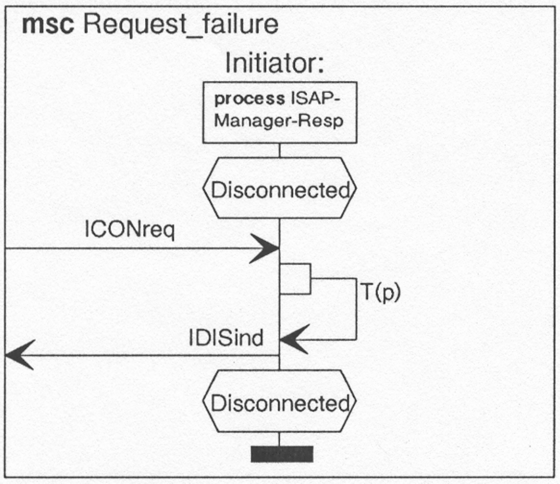
E.g. the MSC *Connection_failure* (fig. 5-2 (c)) can be decomposed into the MSCs *Response_failure* (fig. 5-2 (a)) and *Request_failure* MSC (fig. 5-2 (b)) at the local condition *Disconnected*.



(a) Erroneous transmission of a connection response within the Inres service



(c) Composition of (a) and (b)



(b) Erroneous transmission of a connection request within the Inres service

Figure 5-2: MSC composition and decomposition by means of nonglobal conditions

6 Towards a formal MSC semantics

Within this section one possible approach towards a formal semantics is sketched. The approach has been worked out at University of Berne within the research project *"Conformance Testing - A Tool for the Generation of Test Cases"*, funded by Swiss PTT (F&E project, contract no. 233). The approach uses an interleaving model and is based on finite automata (cf. [GHLLN 92], [LL 92-1], [LL 92-2]).

6.1 An automaton semantics for MSCs

Formally, a single MSC can be interpreted as a graph with two sorts of edges. The nodes represent communication events, e.g. message sending and message consumption. The edges denote the *next-event* and the *signal relation*. The *next-event relation* describes the order of the communication events along the instance axis. The *signal relation* represents the order between sending and consumption of a message. This graph is called a *next-event/signal (ne/sig) graph*.

The *ne/sig graph* of an MSC can be interpreted as a *global state transition graph* (GSTG), containing all possible global states specified by the MSC. The GSTG corresponds to an automaton without explicitely defined end states. In our case the automaton must accept all event traces which are consistent with the partial order of the communication events within the MSC. The semantics of the MSC is given by the behaviour of the constructed automaton.

6.2 Semantics for MSCs with composition mechanisms

Defining end states for the automata in the above case is trivial. But by means of composition rules (cf. chapter 5), a set of MSCs (with conditions) may describe potentially non-terminating sequences. In this case, the whole set of MSCs is translated into a single *ne/sig graph*, which may contain event loops and nondeterministic choices.

To find proper end states a termination criterion from -automata theory, due to Büchi [Tho 90], is used. Unfortunately there is no unique suitable end-state set that turns a GSTG into a Büchi automaton. Instead, various possible end-state sets correspond to liveness properties of MSCs. Examples of such sets are given in [GHLLN 92] and [LL 92-2].

6.3 Remarks on the sketched MSC semantics

The main advantage of the sketched semantics approach and the here upon based MSC semantics is its flexibility. According to the chosen set of end states it is possible to analyse MSCs under various points of view. Other approaches towards a formalization of MSCs, like

[Til 91] or [CCHK 90], do not provide a semantics for MSCs and are not able to handle MSCs with composition rules.

Finally, we like to mention that the *ne/sig graph* presents a general abstract syntax for communicating processes. It can be interpreted in many ways. Within the mentioned research project, a subset of SDL, powerful enough to specify examples like the Inres service [Hog 92], is also translated into a *ne/sig graph* which is then interpreted as a queue automaton. As a result a simple and compact SDL semantics is obtained.

7 Outlook

The MSC activities during the 1989-1992 study period have concentrated on the elaboration of the syntax and informal semantics for basic MSCs. Since the approval of the new recommendation Z.120, the MSC language has become an essential part of several SDL toolsets. In particular, within GEODE [EDGLB 91], SDT [SDT 92], and SIGRAPHSET [GraRu 89] corresponding tool components like MSC editor, simulator, and consistency checker have been included.

Experience with other languages (e.g. SDL) has shown that language maintenance, tool support and determining the relationship between different languages are significantly enhanced by the availability of a formal semantics. Therefore, additional work will be necessary for an elaboration of a formal MSC semantics that in particular will help to establish a formal relationship between MSCs and SDL.

Consequently, MSC activities during the study period 1993-1996 will concentrate on a formal semantics definition, resulting in a revision of Z.120 [Z120]. Enhancements, however, will not be included before 1996. One major step towards an FDT can be seen in the inclusion of formal data description. Further possible enhancements of MSCs refer to abstraction, structuring, modularisation and composition concepts and to object oriented modelling.

Acknowledgements

The authors would like to thank Prof. D. Hogrefe, University of Berne, for the support of this work and P. Graubmann, Siemens Munich, for valuable suggestions and comments.

Literature

[Belina 92] Belina, F. (ed.): SDL Methodology Guidelines, Appendix I to CCITT Recommendation Z.100, Geneva, 1992

[BHS 91] Belina, F.; Hogrefe, D.; Sarma, A.: SDL with Applications from Protocol Specification, Prentice Hall, 1991

[CCHK 90] Cockburn, A.A.R.; Citrin, W.; Hauser, R.F.; von Känel, J.: An Environment for Interactive Design of Communication Architectures, IBM research division, Zurich Research Laboratory, 1990

[EDGLB 91] Encontre, V.; Delboulbe, E.; Gavaud, P.; Leblanc, P.; Boussalem, B.: Combining Services, Message Sequence Charts and SDL: Formalism, Method and Tools, in SDL'91 Evolving Methods -O. Faergemand and R. Reed (editors), North-Holland, 1991

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

[GraRu 89] Grabowski, J.; Rudolph, E.: Putting Extended Sequence Charts to Practice, SDL'89 The Language at Work - O. Faergemand and M. M. Marques (editors), North-Holland, 1989

[GHLLN 92] Grabowski, J.; Hogrefe, D.; Ladkin, P.; Leue, S.; Nahm, R.: Conformance Testing - A Tool for the Generation of Test Cases, Interim report of the F&E project contract no. 233, funded by Swiss PTT, University of Berne, May 1992

[Hog 88] Hogrefe, D.: Automatic Generation of Test Cases from SDL Specifications, CCITT SDL Newsletter 12, 1988

[Hog 89] Hogrefe, D.: Estelle, LOTOS und SDL - Standard Spezifikationssprachen für verteilte Systeme, Springer Verlag, 1989

[Hog 92] Hogrefe, D.: OSI Formal Specification Case Study: the Inres Protocol and Service (revised), Technical report IAM-91-012, University of Berne, 1991, Update May 1992

[ISO 87] ISO TC97/SC21: OSI Service Conventions, Technical Report ISO/TR 8509, 1987

[LL 92-1] Ladkin, P.; Leue, S.: An Automaton Interpretation of Message Sequence Charts, Technical report IAM-92-012, University of Berne, 1992, [LL 92-2] Ladkin, P.; Leue, S.: An Analysis of Message Sequence Charts, Technical report IAM-92-013, University of Berne, 1992,

[Q65] CCITT Recommendation Q.65: Stage 2 of the Method for the Characterisation of Services Supported by an ISDN, CCITT, 1988

[Q699] CCITT Recommendation Q.699: Interworking between the Digital Subscriber System Layer 3 Protocol and the Signalling System No. 7 ISDN User Part CCITT, 1988

[SDT 92] Telesoft Malmö/Sweden: SDT Newsletter, Issue no 1/92, Malmö, 1992

[Tho 90] Thomas, W.: Automata on Infinite Objects, in Handbook of Theoretical Computer Science, chapter 4, pages 132-191, Elsevier Science Publisher, 1990

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

[Tog 92] Toggweiler, D.: TTCN-Testfallgenerierung für mit Sequence Charts spezifizierte verteilte Systeme, University of Berne, Diploma thesis, March 92

[Z100] CCITT Recommendation Z.100: Specification and Description Language (SDL), Geneva, 1992

[Z100-D] CCITT Recommendation Z.100: Specification and Description Language (SDL) - Annex D: SDL User Guidelines, 1988

[Z120] CCITT Recommendation Z.120: Message Sequence Chart (MSC), Geneva, 1992