The Standardization of Message Sequence Charts

Jens Grabowski

Universität Bern
Länggassstrasse 51
CH-3012 Bern

Peter Graubmann, Ekkart Rudolph

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

Abstract

In this paper the most relevant issues of the standardization of the Message Sequence Chart (MSC) language within the CCITT Study Group X are discussed. The history of the new MSC recommendation Z.120 is sketched. Different types of diagrams which are closely related to MSCs are compared, since they build the basis for the MSC language. We distinguish these diagrams from the standardized MSC language by using the term Sequence Charts (SCs). Subsequently, the MSC language is introduced and several approaches towards a forthcoming formal MSC semantics are presented.

1 Introduction

Sequence Charts (SCs) are a widespread means for the graphical visualization of selected system runs (traces) within communication systems. They can be viewed as a special trace language, which mainly concentrates on sending and consumption of messages by synchronously or asynchronously communicating processes. Obviously, main advantage of an SC is its clear graphical layout (e.g. fig. 1, 2), which immediately gives an intuitive understanding of the described system behaviour. SCs have been used successfully in industry and standardization bodies.

In industry, SCs are used mainly as a requirement language to define crucial system traces and as a test case description language. Beyond that, SCs with additional concepts, in particular those enhanced by SDL language elements [7] or SC composition mechanisms [8], offer methods for developing SDL specifications, which serve as the basis for an implementation.

Several working groups within standardization bodies (e.g. CCITT, ISO/IEC) have defined their own variants of SCs, e.g. arrow diagrams, information flow diagrams, or time sequence diagrams. They are applied in standard definitions (e.g. parts of ISDN [15, 14]) or used to illustrate standards specified by formal description techniques like SDL, Estelle, or LOTOS [1]. The various sorts of SCs mainly differ with respect to syntax and terminology. There are only minor semantic differences [18] and hence, a standardization is feasible.

Since there is a need to provide a more formal basis for SCs and to harmonize their use within industry and standardization bodies, the CCITT Study Group X (CCITT SG X) developed the Message Sequence Chart (MSC) recommendation Z.120 [21] from 1989 – 1992. It is based on the family of non standardized SC types, but also includes some additional features like *submsc* and *coregion*.

2 The history of the MSC language

Within the SDL user guidelines of 1988 [20] only a short section has been devoted to SCs as one of the auxiliary diagrams, though SCs 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"* [7]. The terminology *Extended Sequence Charts* (ESC) was used for SCs enhanced by SDL symbols and a few further constructs. ESCs were presented as a means for stepwise refinement and enrichment of SCs from which finally SDL specifications may be derived. Beyond that, the role of SCs within the entire software development process, from requirement specification until test case specification, was pointed out.

Due to the great interest SCs found at this SDL Forum, their standardization in graphical and textual representation within the CCITT was suggested. The standardization was approved at the CCITT meeting in Helsinki, June 1990, and the new language was called *Message Sequence Chart* (MSC). It was decided there to first concentrate on the basic language con-

structs of MSC, i.e. on message flow diagrams without further extensions, and in particular to work out a clear semantics for them. One of the reasons for this restriction, especially to refrain from adding SDL symbols to MSCs, was to avoid too much overlap with SDL [19]. In Helsinki it was not yet decided to prepare a separate recommendation for MSCs. The standardization activities for MSCs were intended to be part of the new *"SDL Methodology Guidelines"* [1], which were aiming at a guideline for the effective use of SDL. Soon after, however, it was recognised that the standardization of MSCs would go beyond the SDL guidelines and it was also felt that MSCs should not be related only to SDL.

Though it was not the intention to develop a fourth formal description technique, in addition to SDL, LOTOS, and ESTELLE, the MSC language was looked at as another specification language which may be used in combination with further languages for system development. Consequently, at the next CCITT meeting in Geneva, Feb. 1991, MSCs were chosen to become a separate recommendation. At this Geneva meeting also the inclusion of further language constructs going beyond *pure* MSCs was agreed upon.

These concepts were elaborated until the CCITT meeting in Recife, Dec. 1991. The language constructs were adjusted to cover the needs of other CCITT recommendations employing Message-, Signal-, or Information-Flow Diagrams. Particularly recommendation Q.65: *"Stage 2 of the method for the characterisation of services supported by an ISDN"* [14] was involved. Q.65 is fundamental for other recommendations in this area. At the CCITT meeting in Recife also the form of the draft MSC recommendation was modified to get in accordance with the SDL recommendation [19].

At the closing session of the CCITT study period 1989 – 1992 in Geneva, May 1992, the new MSC recommendation Z.120 was approved [21]. The work on MSCs in the current CCITT study period 1993 – 1996 started at a CCITT SG X interims meeting in Geneva, Nov. 1992. During this meeting the discussion focused on different approaches towards a formal MSC semantics (cf. section 5) and on the first feedback by MSC users.

3 Comparison of different SC variants

Within this chapter different types of SCs which served as precursor for MSCs are compared. The investigated SCs are Extended Sequence Charts [7], Time Sequence Diagrams according to [11], Arrow Diagrams according to [15], Information Flow Diagrams [14], Sequence Charts according to [3], Message Flow Diagrams [4], Synchronous Interworkings [13] and Siemens-SCs [16]. At the end of this chapter similarities and differences of the investigated SC variants are summarized in tabular form. It was the aim of the MSC standardization that MSC should cover the most common language concepts of various SC variants. Hence MSCs are included into this summary in order to show to which extend this has been achieved.

3.1 The most common SC constructs

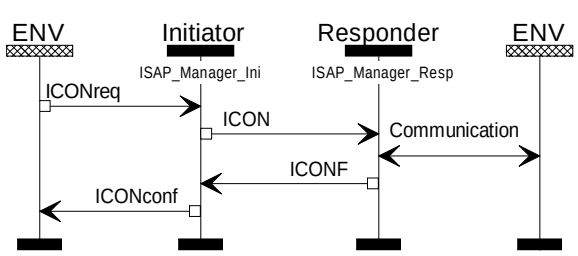
All sorts of SCs show the information flow between instances (e.g. blocks, services, or processes) for specific system runs. They abstract from the internal entity structure by concentrating on the relevant information, namely the instances and the exchanged information. Just by looking at the different diagrams (e.g. fig. 1, 2) we find some SC constructs which are quite similar for all sorts of SCs. These constructs are *instances* which exchange information, *messages* (or signals) which describe the information exchange and *time representation*.

Instances commonly are represented by vertical axes or columns which often are characterized by attached descriptions, e.g. type, or instance names. Messages usually are denoted by horizontal arrows. Sender and receiver of a message are indicated by origin and head of the message arrow respectively. The exchanged information is characterized by additional arrow inscriptions concerning message name, parameter names, or parameter values.

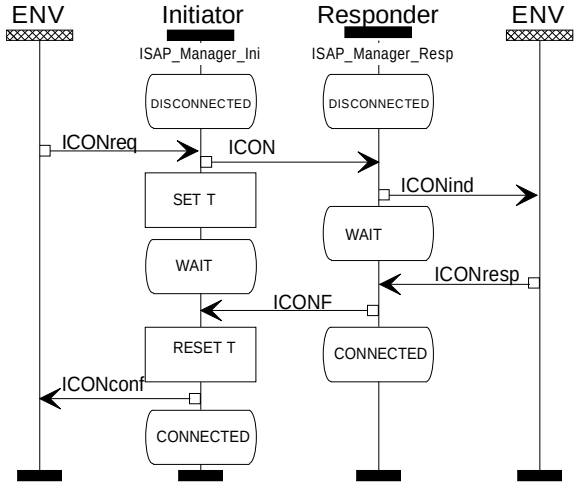
Within SCs no global time is assumed. Every instance axis has its own time scale and time is running from top to bottom. Different axes may be related by communication events, e.g. sending, reception, or consumption of a message. The rules of the time relation between the communication events are determined by the communication mechanism. By that, one can distinguish between *asynchronous* and *synchronous communication*. Asynchronous communication means that a message must be sent before it can be received and be received before it can be consumed. Synchronous communication describes a simultaneous sending and consumption of a message.

3.2 Extended Sequence Charts

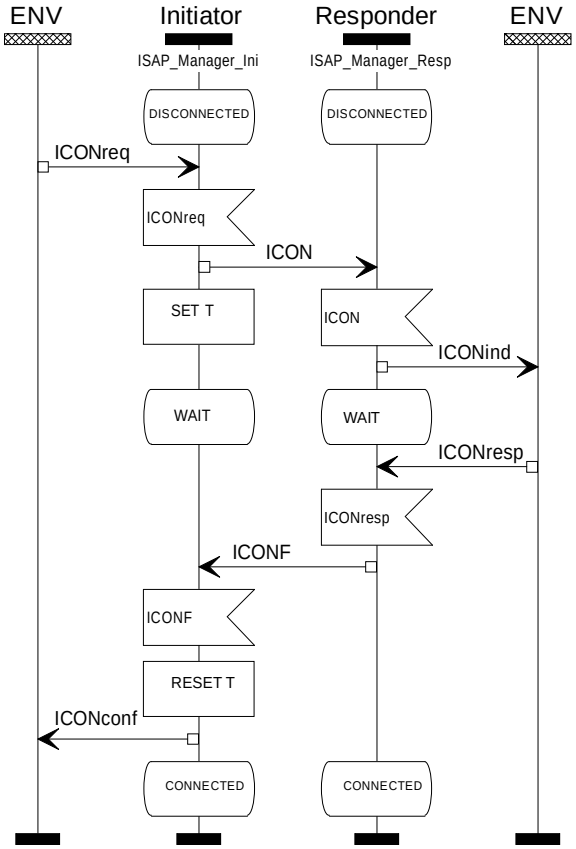
Extended Sequence Charts (ESCs) are described within [7]. The authors define three sorts of ESCs, namely the *standard form*, the *state form* and the *state input form*, which provide a four step method to develop



(a) ESC standard form



(b) ESC state form



(c) ESC state input form

Figure 1: Extended Sequence Charts

SDL specifications. The method is based on the stepwise enrichment of ESCs with SDL symbols. Each step concentrates on a specific aspect of SDL. The first three steps are reflected by the three ESC forms.

All variants of ESCs include the language elements *instance*, *environment*, and *message*. Instances and environments are represented by vertical axes. They communicate via messages which are described by solid arrows. Different entities in the system environment can be characterized by different environment axes. They are distinguished from instance axes by showing different head symbols. For all sorts of ESCs it is assumed that the events along an instance axis are totally ordered. Contrary to that, along an environment axis no time ordering is assumed. Furthermore, due to SDL, for all ESCs an underlying asynchronous communication mechanism is supposed.

ESC standard form. The *ESC standard form* describes the pure message flow between the instances of a system and the system environment. The standard form (fig. 1 (a)) includes all aforementioned language constructs and additionally the *dialog*. A dialog is represented by a bi-directional arrow and is meant to be an abbreviation for a message with a corresponding reply message.

ESC state form. By enriching the standard form with SDL symbols the *ESC state form* (fig. 1 (b)) is gained which is meant to be a refinement of the standard form. All symbols of the standard form are contained in the state form apart from the dialog which on this stage of design should be resolved by messages. Beyond that the state form includes the SDL symbols *state*, *start*, *task*, *comment*, *stop*, *ESC decision-result* and *ESC create*. The latter two are SDL symbols specifically adjusted to the needs of ESCs.

ESC state input form. The *ESC state input form* takes the asynchronous communication mechanism via

FIFO buffers of SDL into account. Therefore, it allows to specify sending, reception and consumption of messages as separate events. Within the state input form the set of symbols used in the state form is extended by an SDL input (fig. 1 (c)). The *input* is used in addition to the message symbol to distinguish between reception (i.e. queue entry) and consumption of a message. This distinction allows the description of special SDL situations like crossing or overtaking of messages.

3.3 Time Sequence Diagrams

Time Sequence Diagrams (TSDs) according to [11] are used for describing OSI services (fig. 2). They illustrate how sequences of interactions between *service users* and a *service provider* are related in time. Every service user has its own *local view* on the service provider. The local views are represented by vertical axes. The exchange of so-called *service primitives* (SPs) is denoted by horizontal arrows which touch the local views. A reference number may be associated with SPs. In this case, identical reference numbers are associated with related SPs. Where additional information is needed, it can be presented by means of notes associated with the related SP.

The communication events are totally time ordered along each local view. Relations between events on different axes can be expressed by diagonal dashed lines (fig. 2 (a)), or in simple cases by solid lines (fig. 2 (b)). In addition a wavy line may be used to express unrelated events. At top of a local view may be a further description (e.g. a circle above the axis) with an identification for reference purposes. Where useful, a TSD indicates a relationship existing among local views by means of lines joining the circles representing these local views.

3.4 Arrow Diagrams

Arrow Diagrams (ADs) or Time Sequence Diagrams according to CCITT recommendation Q.699 [15] are used to describe the interworking between the digital subscriber signalling system layer 3 protocol and the signalling system no. 7 ISDN user part. The basic model for an *interworking entity* (IE) consists of three subentities, namely an *incoming signalling system* (ISS), a *call control* (CC) and an *outgoing signalling system* (OSS). ISS and OSS are represented by vertical axes, whereas a CC is denoted by a column. It is assumed that the events along the axes and the columns are totally ordered. ISS, OSS and CC communicate via so-called *primitives* which are represented by dotted arrows. Arrows with a waved line, if present, represent tones or announcements sent inband.

Within a CC column the following possibilities exist to indicate the relationship between incoming and outgoing primitives. A solid line denotes unconditionally related primitives, a dotted line describes a relation which is only valid within the described context and a waved line denotes unrelated primitives. Furthermore, a collection of CC actions and functions performed on transmission or reception of a signalling message are formulated by means of special symbols (which can be found in [15]).

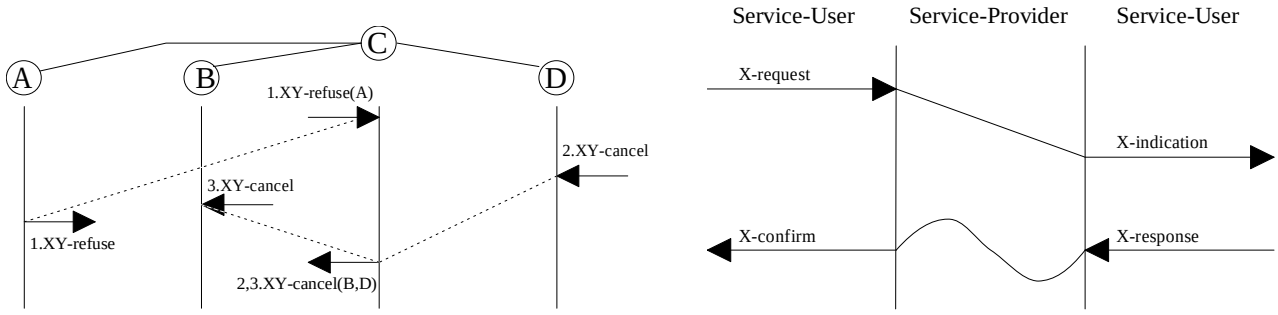
IEs may communicate with other IEs by means of so-called *signalling messages*. Signalling messages are represented by solid arrows and can only be exchanged between the ISS and OSS entities of different IEs. Fig. 3 shows an arrow diagram containing a complete IE and part of a second IE. An AD may include more than one complete IE.

3.5 Information Flow Diagrams

Information Flow Diagrams (IFDs) [14] are used within stage 2 of the overall method for deriving switching and signalling recommendations for ISDN services. They are closely related to TSDs (cf. section 3.3). An IFD (fig. 4) shows the information exchange of *functional entities* (FEs) which are represented by columns. At the top of an IFD each FE is represented by a circle which includes a type information. Since in general more than one instance of one FE type can be instantiated, an instance identification (e.g. FE1 in fig. 4) is associated to each circle. Relations between FEs (e.g. r_i in fig. 4) are indicated by solid lines.

Within the IFD definition messages exchanged between FEs (called *information flows*) and messages exchanged between a FE and a user (called *user inputs* and *user outputs*) are distinguished, even though the same symbols are used. Here, we refer to all these message exchanges as *messages*. Messages are shown as arrows. A descriptive name (e.g. ESTABLISH X) is written in capitals above the arrow and a label (e.g. req.ind) is written below in lowercase characters. If necessary the content of a message (i.e. value of a message parameter) can be shown in lowercase letters enclosed in brackets, following the message name.

Reception and emission of messages are shown by horizontal lines across the relevant FE columns. The absence of a line indicates the lack of reception or emission. A reference number (e.g. 100-106 in fig. 4) is assigned to each reception and emission in the overall sequence at which they are shown. The most significant FE actions can be shown within the FE column.



(a) TSD with more than two local views

(b) TSD with two-part communications

Figure 2: Time Sequence Diagrams for fictitious OSI services

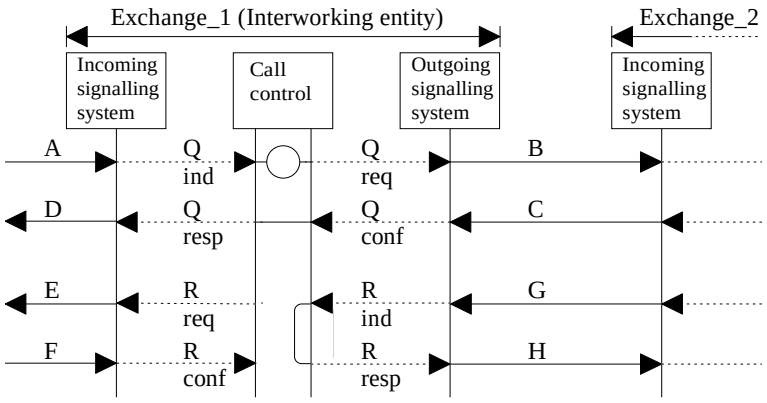


Figure 3: Arrow Diagram

For these actions the following holds. Actions shown below a line representing a message reception are dependent upon that reception (i.e. they cannot be carried out beforehand). Actions shown above a line representing a message emission must be completed prior to the emission. For an actions shown immediately below a line representing a message emission no constraint on the relative ordering of emission and action is intended.

3.6 Sequence Charts for EWSD

Sequence Charts according to [3] (EWSD-SCs) are employed at the Siemens AG (Germany) for the specific needs of the EWSD (Elektronisches Wählsystem Digital) system development. EWSD-SCs show the communication links between entities like blocks, processes, services, and environments. Entities are represented by columns. Horizontal arrows denote communication links. Therefore, a list of messages may be associated with an arrow. Messages denoting physical signals are represented with additional stars (e.g. ***DIAL TONE***>). Furthermore, EWSD-SCs offer facilities to express logical relations like OR and AND between messages (fig. 5). Additionally, global initial and final system states by means of *conditions* can be specified. Also there exist a possibility to parameterize messages and conditions. Special symbols may be used to express time delay or time supervision.

3.7 Message Flow Diagrams

Message Flow Diagrams (MFDs) are the user interface of the CARA system [4] for developing protocol specifications. A MFD shows the message flow between *protocol entities* (PEs) which are represented by vertical lines. Messages are represented by arrows. They are sent and received at *ports* which are owned by PEs. Messages are transmitted via so-called *links*. A link may represent any communication media like lower protocol levels or specific physical media. A MFD assumes asynchronous communication which means that along a PE axis communication events (head and origin of message arrows) are totally ordered and a

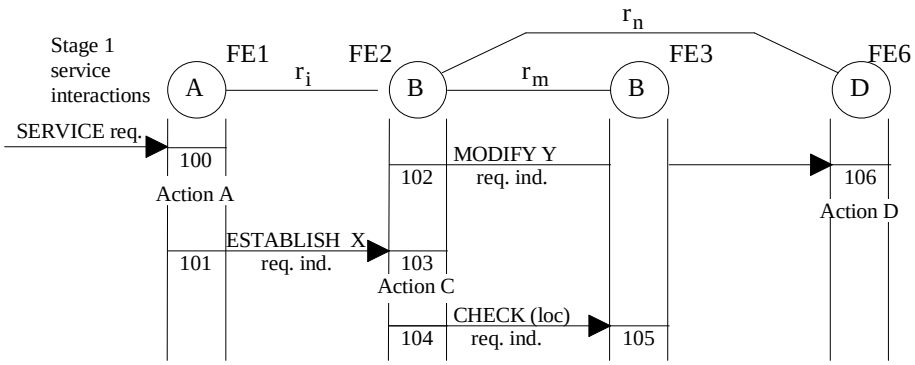
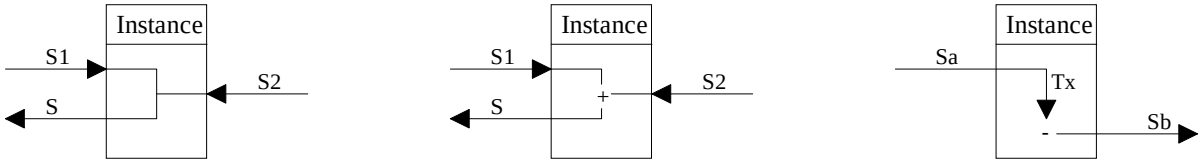


Figure 4: Information Flow Diagram



(a) AND relation (S = S1 ∧ S2) (b) OR relation (S = S1 ∨ S2) (d) Time delay

Figure 5: Examples for EWSD-SCs

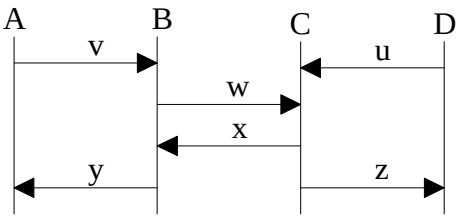
message must be sent before it can be received. The MFD language includes a special symbol, the so-called *black hole*, to specify the loss of a message. Contrary to all other mentioned SC variants, MFDs include a data concept. It is possible to use and to declare data within a MFD and to define rules for the processing of data. MFDs are a part of the CARA system. CARA is based on the model of *communicating rule systems* (CRS) and assumes that the behaviour of a protocol can be expressed by a set of MFDs. Therefore, MFDs are transformed into rules which are interpreted and analysed like Prolog programs. During the process of protocol specification the CARA system analyses MFD inputs and tries to apply rules which are in the already developed CRS database. If this is not possible a new rule is created or the user is asked for help. The result of this stepwise protocol development is a complete CRS which represents the protocol specification. Based on CRS a formal semantics for MFDs is defined. It is given by the formalism used to map the graphical MFD language into a CRS and the interpreter of the CRS within the CARA system.

3.8 Synchronous Interworkings

Synchronous Interworkings (IWs) [13] are used in the requirement phase of the development process at PKI Nürnberg (Germany) for describing the message interactions between *functional blocks*. For IWs a graphical and a textual representation is defined (fig. 6). Within the graphical form, functional blocks are represented by vertical lines and messages are represented by horizontal arrows. Communication between two functional blocks is meant to be *synchronous*. IWs can be parameterized. A parameter of an IW stands for message names, process names, or message parameter names. IWs do not include symbols which may be used to describe further actions of a functional block. For IWs a formal semantics definition exists which is explained in section 5.3.

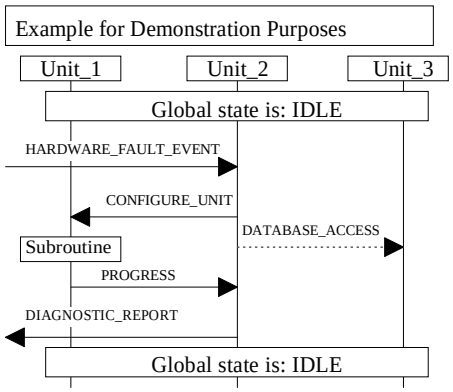
3.9 Siemens-SC

At an CCITT SG X meeting in Geneva, Nov. 92, the Siemens AG (Germany) presented another SC variant which we name Siemens-SC [16]. Within Siemens-SCs communicating entities are represented by vertical axes. They offer facilities for describing synchronous and asynchronous communication by means of dashed and solid arrows. Furthermore, there exist symbols to represent *actions*, *timer sets*, *time-outs* and *conditions* (symbols to denote system states). Siemens-SCs have a textual and a graphical representation (fig. 7). They are used to describe the normal behaviour and exclude special situations like message overtaking, process creation, or process termination.




```
INTERWORKING example
PROCESSES A, B, C, D ENDPROCESSES
D SENDS u TO C
A SENDS v TO B
B SENDS w TO C
C SENDS x TO B
B SENDS y TO A
C SENDS z TO D
ENDINTERWORKING
```

Figure 6: Graphical and textual representation of a Synchronous Interworking




```
TITLE: "Example for Demonstration Purposes"
SYN: OP1 = HARDWARE_FAULT_EVENT
SYN: OP2 = CONFIGURE_UNIT
SYN: OP3 = DATABASE_ACCESS
SYN: OP5 = PROGRESS
SYN: OP6 = DIAGNOSTIC_REPORT
OBJ: ENVL Unit_1 Unit_2 Unit_3
COMMENT: "Global state is: IDLE"
MSG: OP1 ENVL Unit_2
MSG: OP2 Unit_2 Unit_1
RPC: OP3 Unit_2 Unit_3
ACTION: Unit_1 "Subroutine"
MSG: OP4 Unit_1 Unit_2
MSG: OP5 Unit_2 ENVL
COMMENT: "Global state is: IDLE"
```

Figure 7: Graphical and textual representation of a Siemens-SC

3.10 Summary

Within this section the similarities and differences between the examined SC variants are summarized. The comparison is arranged in the tables 1 – 3, each of which considers a specific aspect. Although the MSC language is introduced not until the subsequent chapter, we include respective information here. This shows that MSC covers most of the similar concepts and offers therefore a broad and more formal basis. However, the MSC language definition is not yet totally elaborated. Some language constructs are under discussion, or in preparation. They are mentioned within the tables.

4 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. Finally, we describe the so-called *composition rules for MSCs* since they are useful to combine several MSCs in order to come up with more complete MSC specifications.

4.1 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 [2, 19]. Let us consider the MSC in fig. 9 which describes a selected trace piece of the connection set-up in the Inres service specification [10]. It could equally be represented using SDL process diagrams with certain additions and modifications (fig. 8, dashed symbols stand for not followed branches, bold-lined arrows indicate the message flow). The diagram in fig. 8 contains at least the same information as the MSC in fig. 9. 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, the MSC in fig. 9 obviously is much more transparent than fig. 8, 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

	SC representation		symbols concerning			representation of	
	graphical	textual	SDL	entity actions	timer	local states	nonlocal states
ESC standard form	X		X				
ESC state form	X		X	SDL task	SDL timer	SDL state	
ESC state input form	X		X	SDL task	SDL timer	SDL state	
Time Sequence Diagrams	X						
Arrow Diagrams	X			X			
Information Flow Diagrams	X			X	X		
EWSD-SCs	X			X	X		X
Message Flow Diagrams	X						
Synchronous Interworkings	X	X					
Siemens-SC	X	X		X	X	X	X
Message Sequence Charts	X	X		X	X	X	X

Table 1: Diagram representation, symbols at entity axes and state representation

	communication		synchronization	formal basis	
	asynchr.	synchr.		semantics	combining diagrams
ESC standard form	X				
ESC state form	X				
ESC state input form	X				
Time Sequence Diagrams	X		X		
Arrow Diagrams	X		X		
Information Flow Diagrams	X				
EWSD-SCs	X				
Message Flow Diagrams	X			rule system	set union of rules
Synchronous Interworkings		X		process algebra	merging, sequencing
Siemens-SC	X	X			
Message Sequence Charts	X	(in prep.)	(in preparation)	(in preparation)	composition rules

Table 2: Communication, synchronization and formal basis

	architectural information (e.g. communication links)	external notes	other SC specific characteristics
ESC standard form			*dialog* symbol
ESC state form			
ESC state input form			
Time Sequence Diagrams	at the top of the diagram	X	different diagrams for two- and multi-party comunication
Arrow Diagrams		X	symbols denoting functions performed on transmission or reception of messages
Information Flow Diagrams	at the top of the diagram	X	
EWSD-SCs	within the diagram		relations between messages (e.g. AND, OR)
Message Flow Diagrams			- *black hole* (spec. of message loss) - data handling
Synchronous Interworkings			
Siemens-SC			abbreviations for long names
Message Sequence Charts			*coregion* (cf. section 4.3)

Table 3: Architectural information, external notes and other SC specific characteristics

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. [9]):

"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. 8 and fig. 9 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 [1, 7].

4.2 MSC/PR and MSC/GR

Analogous to the SDL recommendation [19] 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. 9.

4.3 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*, *action*, *set→reset* (time supervision), *set→time-out* (timer expiration), *stop*, *create* and *condition*.

Instance, message and system environment. 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 shown by vertical lines or alternatively by columns (fig. 9 (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. 10 (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.

Within an MSC the system environment is represented by the frame symbol which forms the boundary of an MSC diagram (e.g. fig. 9, 10). Contrary to instances, no ordering of communication events is assumed.

Actions and timer constructs. Within an MSC it is possible to indicate *actions* and timer handling. An action is represented by a rectangle containing 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. 10.

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. 10 (b). 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.

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 (cf. section 4.5). In the MSC/GR representation global and nonglobal conditions are represented by hexagons covering the involved instances.

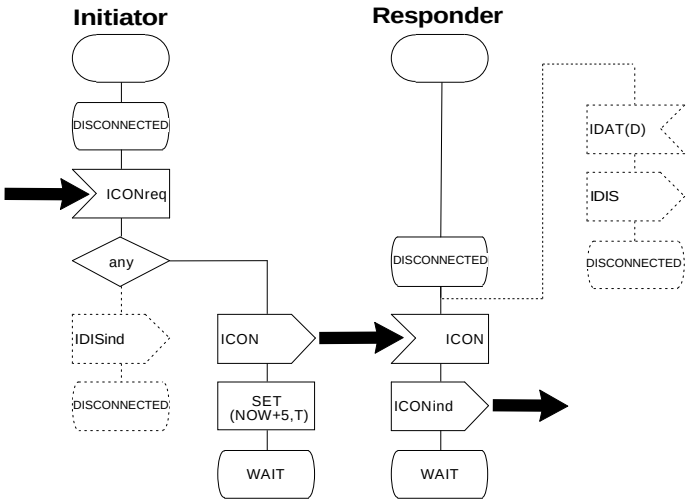
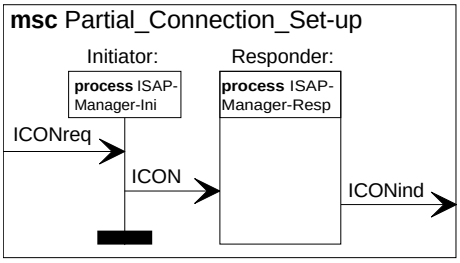


Figure 8: (Non-standard) combined SDL - message flow diagram




```
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 ICONind to env;
   endinstance;
endmsc;
```

Figure 9: MSC in MSC/PR and in the corresponding MSC/GR representation

In fig. 10 (c) 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.

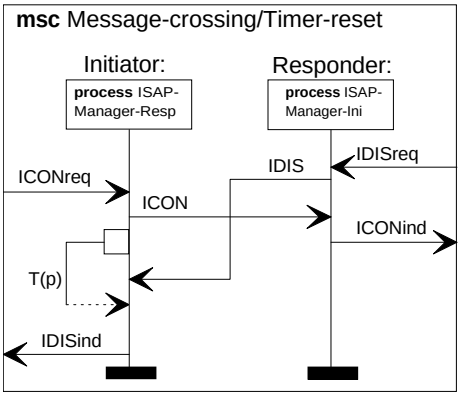
In the MSC/PR representation conditions are introduced at two different places: on the level of MSCs in form of global conditions and on the level of instances referring to an arbitrary set of instances. In the second case the condition may be local, i.e. attached to just one instance. If the condition 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 global condition syntax form may preferably at least at the beginning and at the end of an MSC.

4.4 Structural language elements

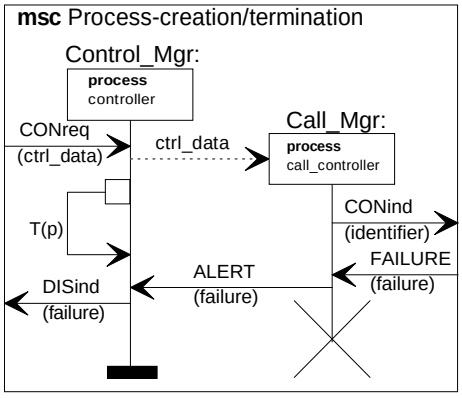
The structural language elements of MSCs include all constructs which can be used to specify more general MSCs or to refine MSCs. Therefore, the current MSC recommendation offers the *coregion* and the *submsc*.

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 *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. An example containing coregions is given in fig. 11 (a).

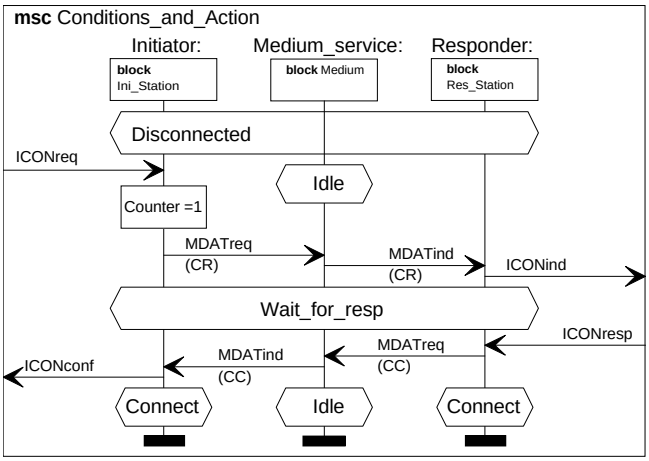
Submsc. An MSC instance can be refined by another MSC, which then is called *submsc*. A submsc is attached to the refined instance by means of the keyword *decomposed*. The submsc represents a decomposition of this instance without affecting its observable behaviour. The messages addressed to and coming from the exterior of the submsc are characterised by the messages connected with the submsc



(a) Message crossing and *set→reset*



(b) *Set→time-out*, *create* and *stop*



(c) *Conditions* and *action*

Figure 10: MSCs with basic language constructs

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.

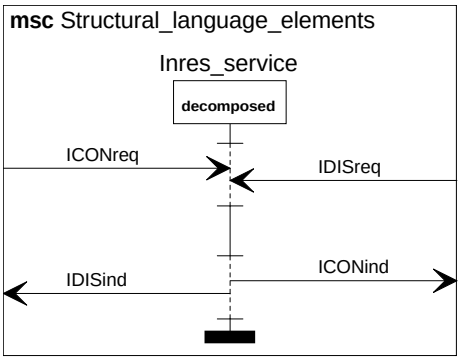
Actions and conditions within a submsc may be looked at as a refinement of actions and conditions in the decomposed instance. Contrary to messages, however, no formal mapping to the decomposed instance is assumed, i.e. the refinement of actions and conditions need not obey formal rules. In fig. 11 (b) the refinement of the instance *Inres_service* (fig. 11 (a)) is shown.

4.5 Composition and decomposition rules

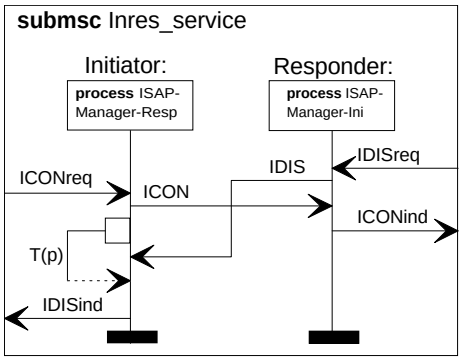
Since one MSC only describes a partial system behaviour, it is advantageous to have a number of simple MSCs that can be combined in different ways. To determine possible combinations the already introduced (global and nonglobal) conditions can be used employing certain *composition and decomposition rules*. The presented rules are not yet part of [21] but it is intended to include them in the future. Presently they are part of the SDL methodology guidelines [1].

4.5.1 The meaning of composition and decomposition of MSCs

MSCs can be composed by name identification of *final* and *initial* (global or nonglobal) conditions. The



(a) MSC with *coregions*



(b) Refinement of *Inres_service* in (a)

Figure 11: MSCs with structural language elements

other way round, MSCs can be decomposed at *intermediate* (global or 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. 12. The MSC *Complete_system_run* (e) 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.

4.5.2 Composition of MSCs

The composition of MSCs is defined by means of global and nonglobal conditions.

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. 12). 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*.

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. 12. The MSC *Connection_failure* (f) is a composition of the MSCs *Response_failure* (c) and *Request_failure* (d) 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 listed instances to which they are attached.

4.5.3 Decomposition of MSCs

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

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'*

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(*MSC2*) = I(*MSC2*)' *Condition*
(3) I(*MSC3*) = *Condition* I(*MSC3*)'

E.g. in fig. 12 the MSC *Connection_failure* (f) can be decomposed into the MSCs *Response_failure* (c) and *Request_failure* (d) at the local condition *Disconnected*.

5 Towards a formal MSC semantics

Within this section three approaches towards a formal MSC semantics are sketched. All three have been briefly presented at an CCITT SG X meeting, Nov. 1992 in Geneva. They can be considered the starting points from which the MSC semantics discussion within CCITT SG X will proceed. The approach discussed first (section 5.1) uses an interleaving model and is based on finite automata [6, 12]. The second approach (section 5.2) adopted a full partial order representation for system traces and is based on Petri net theory [5]. The third approach (section 5.3) again applies an interleaving model and is based on process algebra [13].

5.1 An automaton semantics for MSCs

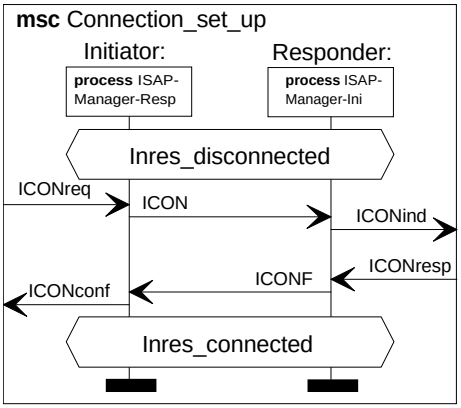
Formally, a single MSC can be interpreted as a graph with two sorts of edges. The nodes represent communication events, e.g. message sending and message consumption. The edges denote the *next-event* and the *signal relation*. The *next-event relation* describes the order of the communication events 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 explicitly defined end states. The automaton which defines the MSC semantics must accept all event traces which are consistent with the partial order of the communication events within the MSC.

Defining end states for an automaton arising from an isolated MSC is rather trivial. But by means of MSC composition rules (cf. section 4.5), a set of MSCs (with conditions) may describe potentially nonterminating 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 ω-automata theory, due to Büchi [17], 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 [6] and [12]. The main advantage of the sketched semantics approach and the hereupon based MSC semantics is its flexibility. According to the chosen set of end states it is possible to analyse MSCs under various points of view.

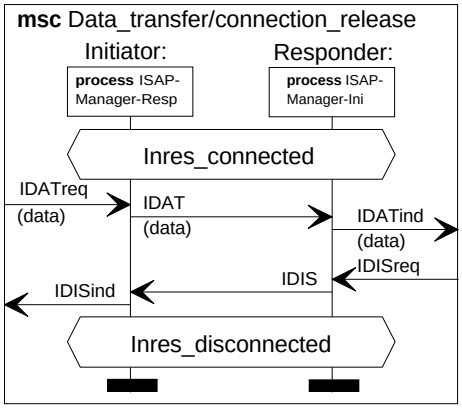
5.2 A Petri Net based MSC semantics

A second approach towards a formal MSC semantics is based on Petri net theory [5]. As the idea of partial ordering of signalling and instance events was one of the leading principles during the definition of MSCs, *occurrence nets* - which are the Petri net way of presenting partial orderings - seem to be particularly well suited for a basis of MSC semantics.

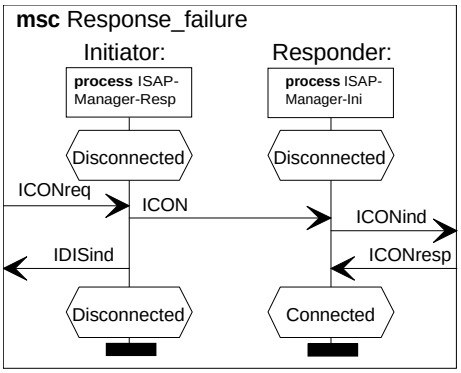
A MSC is defined to describe one particular system trace of a communication system. For this trace



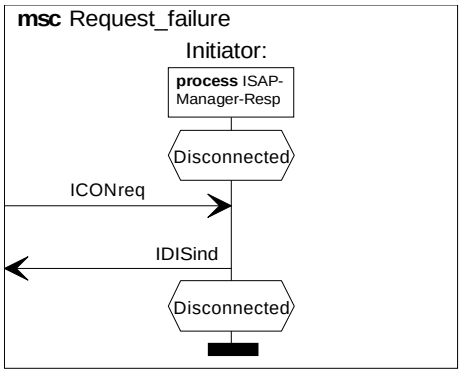
(a) Successful connection establishment



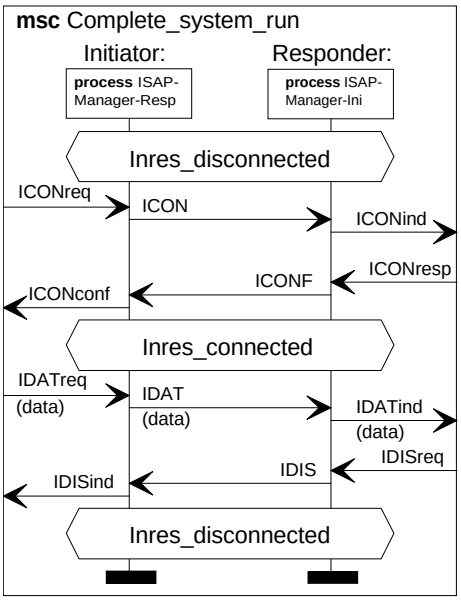
(b) Data transfer and disconnection



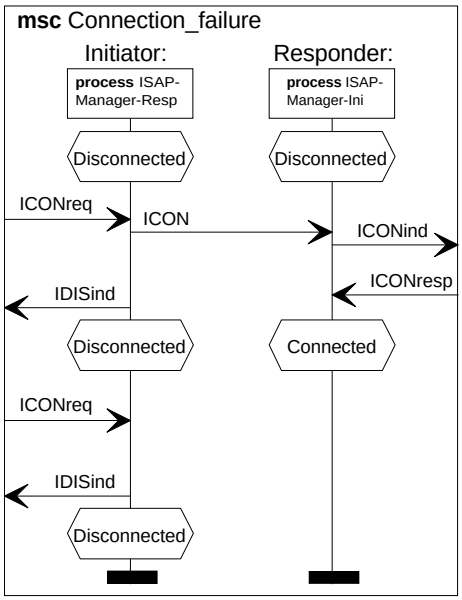
(c) Erroneous transmission of a connection response



(d) Erroneous transmission of a connection request



(e) Composition of (a) and (b)



(f) Composition of (c) and (d)

Figure 12: Composition and decomposition of MSCs

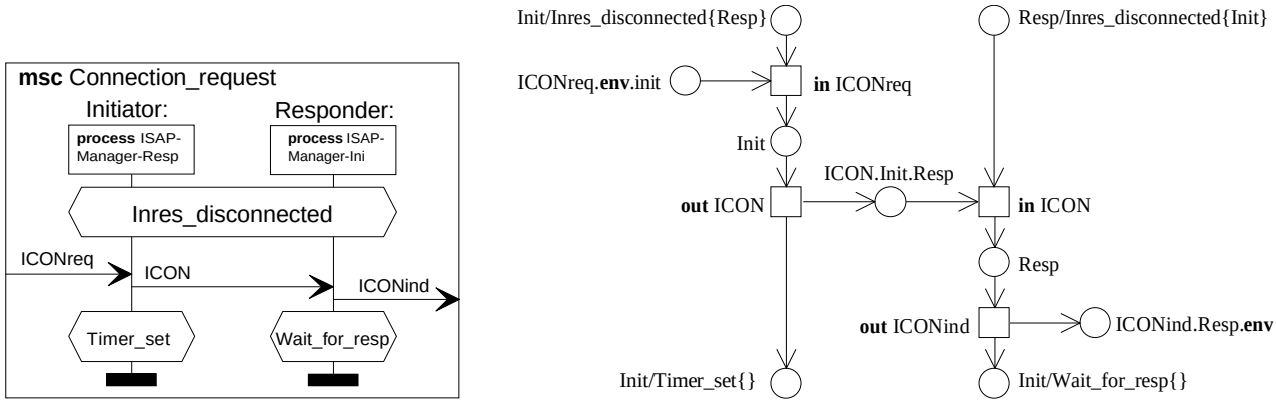


Figure 13: MSC and the corresponding occurrence net representation

a semantics definition shall be provided by means of a (labelled) occurrence net (also called causal net). Occurrence nets immediately reflect the partial ordering of events and therefore suitably describe individual traces within a distributed system. They also are very well suited to handle the composition and decomposition of MSCs, because local and global conditions map onto special sets of occurrence net elements which in their turn describe reachable system states. For further analysis, the rich mathematical theory behind occurrence nets, which was elaborated within Petri net theory, can be employed. Thus, occurrence nets provide an elegant but also rather intuitively intelligible way for the due MSC semantics definition. There is a simple and quite intuitive mapping of MSCs onto occurrence nets (cf. fig. 13).

Each instance axis of an MSC is mapped straight away by converting the instance events one by one into elementary labelled Petri net constructs. Equally labelled net elements are identified during this construction process. As next step, the Petri Net constructs representing the instance axes are glued together via equally labelled places resembling the messages in the MSC. During this construction process the labelling of the net elements takes over a considerable amount of information which ensures the correct identification of corresponding net elements. A list of the elementary net constructs that are used for the transformation of MSC language elements is given in [5]. A detailed description of the necessary labelling, its generation from the MSC syntax and the definition of the construction process is also to be found there.

5.3 A process algebra semantics for MSCs

As mentioned in section 3.8 for Synchronous Interworkings (IWs) a formal semantics is defined [13]. The semantics is used to process, analyse, and combine IWs. Within an IW a message interaction between two entities can be split into two different events: the output and the input of the message. But contrary to MSCs, within IWs communication is meant to be synchronous. A formal semantics for IWs is defined with the use of the algebraic concurrency theory BPA (Basic Process Algebra). To that end BPA is extended by two operators: *IW merging* and *IW sequencing*. The class of IWs consists of all processes definable over these two operators. The IW sequencing denotes a vertical concatenation of two IWs. In case where the IWs have all entities in common, the IW sequencing corresponds to a real sequentialization in time. Events belonging to entities, not common to both IWs, will be unordered by the IW sequencing. The IW merging of two IWs is their interleaved composition with the restriction that the IWs are forced to synchronise on a set of communication actions. This set consists of the communication actions concerning every pair of entities which the IWs have in common. It is obvious that not all IWs are merge consistent. An IW is a process which can be constructed only from atomic actions and applications of the sequence and the merge operators. Thus, the two IW operators provide a means to build up IWs from basic actions, in particular message events.

6 Outlook

The MSC activities during the 1989 – 1992 study period have concentrated on the elaboration of the syntax and informal semantics for basic MSCs. Experience with other languages (e.g. SDL) has shown that language maintenance, tool support and determining the relationship between different languages are signif-

icantly 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 [21]. Enhancements, however, will not be included before 1996. One major step towards an formal description technique can be seen in the inclusion of formal data description. Further possible enhancements of MSCs refer to concepts for abstraction, structuring, composition and object oriented modelling.

Acknowledgements

The authors would like to thank Prof. D. Hogrefe, University of Berne, for the support of this work.

References

[1] Belina, F.: SDL Methodology Guidelines, CCITT, Geneva, May 1992

[2] Belina, F.; Hogrefe, D.; Sarma, A.: SDL with Applications from Protocol Specification, Prentice Hall International (UK) Ltd., 1991

[3] Richlinie P30305-X185-B6-2-35, EWSD Software-Entwicklungshandbuch, Kap. B, Register 6, SDL-Diagramme, Siemens AG, 1988

[4] 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 Laboratry, 1990

[5] Graubmann, P.; Rudolph, E.; Grabowski, J.: Towards a Petri Net Based Semantics Definition for Message Sequence Charts, Proceedings of the 6th SDL Forum, 1993

[6] 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

[7] 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

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

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

[10] Hogrefe, D.: OSI Formal Specification Case Study: the Inres Protocol and Service, Technical report IAM-91-012, University of Berne, May 1992

[11] ISO/IEC JTC 1/SC 21: Revised Text of CD 10731, Information Technology - OSI Service Conventions, ISO/IEC JTC 1/SC 21 N 6341, Januray 1991

[12] Ladkin, P. B.; Leue, S.: Interpreting Message Sequence Charts, IBM Computer Science Research Report, RJ 8965 (80241) September 18, 1992

[13] Mauw, S.; van Wijk, M.; Winter, T.: Syntax and Semantics of Synchronous Interworkings, CCITT SG X Interims Meeting, Geneva, 1992

[14] CCITT Recommendation Q.65: Stage 2 of the method for the characterization of services supported by an ISDN, Geneva, 1988

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

[16] Siemens AG (Germany): Comments on Recommendation Z.120 Message Sequence Chart (MSC), CCITT SG X Interims Meeting, Geneva, 1992

[17] Thomas, W.: Automata on Infinite Objects, in Handbook of Theoretical Computer Science, Elsevier Science Publisher, 1990

[18] Toggweiler, D.: TTCN-Testfallgenerierung für mit Sequence Charts spezifizierte verteilte Systeme, University of Berne, Diploma thesis, 1992

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

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

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