Tutorial on Message Sequence Charts (MSC'96)

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

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

[b]Institute for Telematics, University of Lübeck, Ratzeburger Allee 160, D-23538 Lübeck, Germany, eMail: jens@itm.mu-luebeck.de

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

Abstract

MSC is a trace language which in its graphical form admits a particularly intuitive representation of system runs in distributed systems while focusing on the message interchange between communicating entities and their environment. For the first time the MSC recommendation Z.120 (MSC'92) was approved at the ITU meeting Geneva 1992. A new revised MSC recommendation Z.120 (MSC'96) was approved at the closing session of the last study period in April 1996.

Whereas in MSC'92 main emphasis was put on the elaboration of basic concepts and a corresponding formal semantics, in the new MSC version - MSC'96 - structural language constructs, essentially composition and object oriented concepts, play a dominant role. With these new concepts, the power of MSC is enhanced considerably in order to overcome the traditional restriction to the specification of only a few selected system runs. Within the tutorial, the use of MSC is demonstrated by means of the ISDN supplementary service 'Completion of Calls to Busy Subscriber (CCBS)'.

1 Introduction

Message Sequence Chart (MSC) has matured within a very short period of time to a considerably powerful and expressive language. Before the approval of the first MSC recommendation Z.120 in 1992, MSC was used merely as an informal, illustrative language, e.g., in form of auxiliary diagrams within the SDL community [10]. In the meantime, MSC has advanced to a formal and descriptive language, i.e., a formal description technique (FDT). The edition of the first MSC recommendation in 1992 [22] has increased use and popularity of the MSC language beyond the expectation of most people. Main reason for this success was that for the first time a systematic tool support became possible for MSC due to its standardisation. Nevertheless, the language constructs defined in MSC'92 [7, 20] appeared to be not sufficient to describe comprehensively even parts of an information system. MSC therefore was considered fruitful only in combination with other languages, dominantly SDL and TTCN (e.g., [9]).

During the last ITU study period (1993 - 1996) the MSC language obtained a great impulse by the development of a corresponding formal semantics based on process algebra [17, 18, 24]. Though the idea of combining MSC with composition mechanisms from process algebra goes back to the early days of MSC standardisation [21] and was carried forward within the GEODE tool [5], a satisfactory formulation was found only recently after the development of the formal MSC semantics. In some respects MSC'96 [23] now looks like a graphical representation of process algebra whereas MSC'92 was influenced very much by ideas coming from Petri Nets with its composition mechanisms based on conditions [8, 11]. A closer look shows that the condition based composition mechanisms from MSC'92 have not been dropped but incorporated into process algebra based techniques in MSC'96. Surprisingly after all, no other language construct in MSC'96 has been discussed more extensively than the role of conditions. Even the first Internet meeting in the history of ITU and a major part of a conference in Russia (St. Petersburg) have been dedicated to this subject.

Thus eventually, MSC'96 has become a powerful synthesis of concepts taken from process algebra, Petri Nets and, beyond that, from object oriented modelling. Recently, also the object oriented community has shown increasing interest in the MSC standard as a means for the formalisation of Use Cases. This is now even under discussion within the "Unified Method for Object Oriented Development" [3]. Due to the new language concepts within MSC'96 - generalised ordering, inline expression, reference, High Level MSC (HMSC) - the range of applicability of MSC has increased considerably. The specification of Use Cases [14], i.e., of main scenarios together with all accompanying side cases, is one of the most promising candidates for the application of MSC'96 [2, 16]. This way, the traditional restriction of MSC to the specification of only few selected scenarios, which was considered as the major shortcoming of MSC'96, can be overcome. The treatment of the CCBS example which has been chosen for this tutorial is carried through in this spirit.

2 Short description of the CCBS example

An *Integrated Services Digital Network* (ISDN) (e.g., [15]) is a fully digital network that provides a large variety of data and telecommunications services. In its simplest form, an ISDN is merely an enhancement to the telephone local loop that will allow both voice and data to be carried over the same twisted pair. *Supplementary services* provide additional capabilities to ISDN users, so they may exert greater control over how the network handles their transmission paths. E.g., *call forwarding*, *call waiting* are a few of the capabilities that are meant to be supplementary ISDN services.

For our purpose, the explanation of the MSC language, we selected another supplementary service, namely *'Completion of Calls to Busy Subscriber'* (CCBS). For its description we rely on the European Telecommunication Standard (ETS) No. 300 359-1 [6].

The CCBS service enables user A, encountering a busy destination B, to have a call completed without having to make a new call attempt when destination B becomes not busy. When user A requests the CCBS service, the network will monitor for destination B becoming not busy. When this happens then the network will wait a short period of time in order to allow the resources to be re-used by B originating a new call. If this is not the case within a given time frame, then the network will automatically recall user A. After user A accepts the CCBS recall, the network will automatically generate a CCBS call to destination B.

3 Specification of the CCBS example

3.1 HMSC 'CCBS_SERVICE'

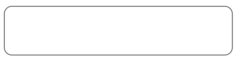
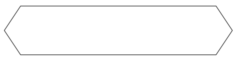
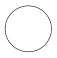
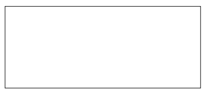
The High-level MSC (HMSC) 'CCBS_SERVICE' (Figure 1) provides the specification of the CCBS on the top level. The HMSC 'CCBS_SERVICE' starts in the condition 'CCBS_Idle'. The CCBS request from User A (reference 'REQUEST') is either accepted and the CCBS activation (reference 'ACTIVATION') is processed or it is rejected (reference 'REJECT') and the HMSC returns to the initial 'CCBS_Idle' state. After a successful request the system arrives in the state 'CCBS_Activated'. The subsequent reference 'MONITORING' refers itself to an HMSC which specifies the monitoring of user B and user A. When both user A and user B are found to be not busy the recall is started by sending a recall indication to user A and the system reaches the state 'CCBS_Free'. If user A accepts the recall then the CCBS service is completed (reference 'INVOCATION') and the system changes into state 'CCBS_Init'. If user A does not reply in time to the recall the CCBS service is canceled (reference 'CANCEL') and the system returns to the initial 'CCBS_Idle' state. In case the CCBS recall is successfully completed the resources can be released (reference 'RELEASE'). The CCBS supplementary service may be disrupted by user A in all intermediate states (reference 'DEACTIVATION') causing the system to return into the initial 'CCBS_Idle' state.

The HMSC 'CCBS_SERVICE' shows already a good part of the main language constructs defined for use in an HMSC. HMSCs provide a means to graphically define how a set of MSCs can be combined. An HMSC is a directed graph where each node is either:

- a *start symbol*: ▽

  Note: There is exactly one start symbol in each HMSC.

- an *end symbol*: △

- an *MSC reference*: 
- a *condition*: 
- a *connection point*: 
- a *parallel frame*: 

  The use of a parallel frame will be discussed in Section 4.1.

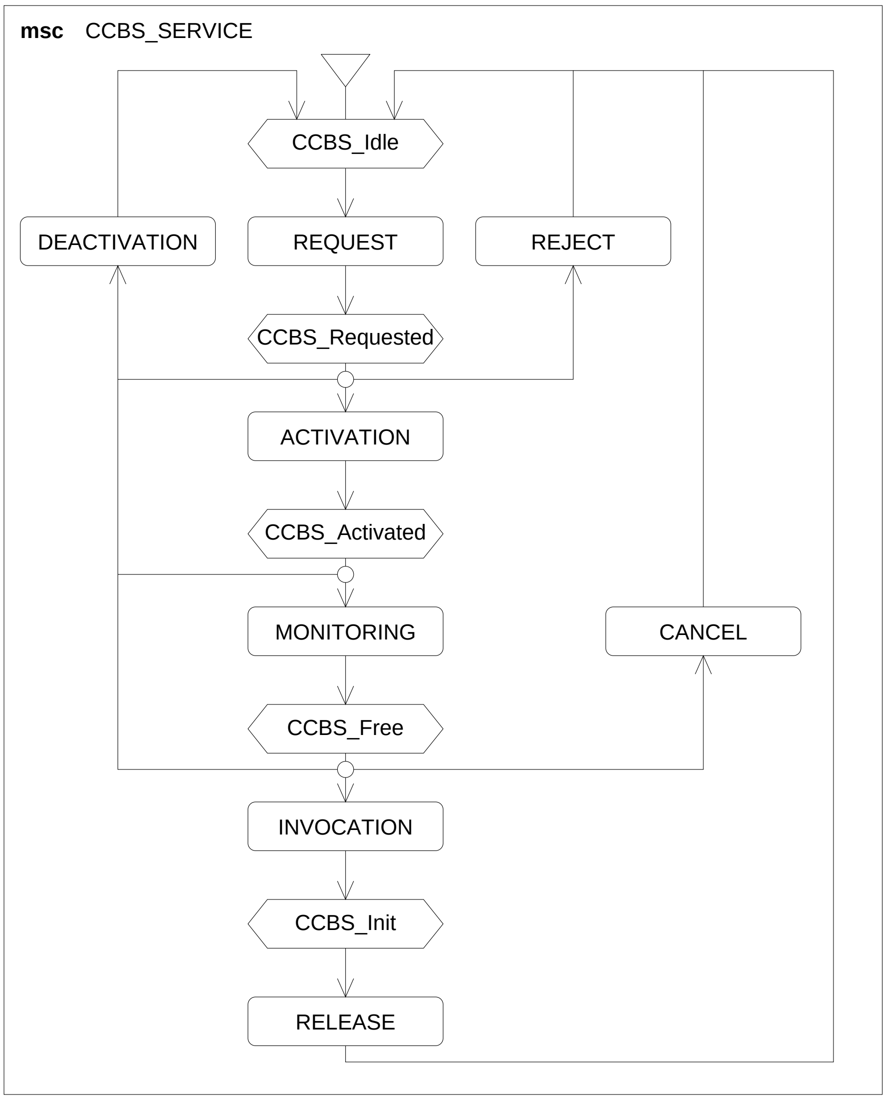


Figure 1

The HMSC 'CCBS_SERVICE' starts with the start symbol graphically represented by a downward pointing triangle. It is connected by an HMSC line-symbol with the condition 'CCBS_Idle'. All conditions on the HMSC level are considered to be *global*. They may be seen as indicating global system states. At the same time, they can be used to guard the composition of MSCs described by HMSCs. Whereas in MSC'92 the composition was based completely on the merging of final and initial conditions, in MSC'96 the composition of MSCs is defined by means of HMSCs. The conditions in MSC'96 play a restrictive role defined by a set of static semantics rules in Z.120. To illustrate these rules, let us consider the MSC reference 'REQUEST' which follows the condition 'CCBS_Idle'. Each MSC reference points to another MSC which defines the meaning of the reference, i.e., the reference construct in the HMSC can be seen as a placeholder for an MSC diagram. Thus, the reference 'REQUEST' refers to the MSC 'REQUEST' which can be found in Section 3.2. This MSC starts with the initial condition 'CCBS_Idle'. The static semantics rules for MSC'96 now state that a HMSC-condition immediately preceding a MSC reference has to agree with the (global) initial condition of the MSC reference (if present) according to name identification. The restrictive role of conditions in MSC'96 leaves sufficient freedom to the system designer since the specification of initial and final conditions in MSC references, but also the specification of HMSC conditions preceding or following a MSC reference, is optional. This way, the HMSC composition mechanism offers great flexibility but also supports composition in the spirit of MSC'92 (except that MSC'96 composition rules ignore non-global conditions, i.e., conditions which refer to a true subset of the instances contained in the MSC). The flexible use of conditions in MSC'96 is demonstrated within HMSC 'MONITORING'. In HMSC 'CCBS_SERVICE' a special generalisation of MSC'92 conditions is used: conditions may contain a name-list. The static rules governing the composition of MSCs now state that the name-list attached to an HMSC condition must be a subset of the name-list of the adjacent initial or final condition of an MSC reference. E.g., the reference 'DEACTIVATION' refers to an MSC (defined in Section 3.6.4) with the condition name list 'CCBS_Requested', 'CCBS_Activated', 'CCBS_Free' thus allowing a composition at several places of the HMSC. The connection points (cf. the branching point after the condition 'CCBS_Requested' as an example) are introduced merely for convenience in order to improve the layout. They have no semantical meaning. The HMSC 'CCBS_SERVICE' does not contain any end symbol since it is cyclic.

Contrary to plain MSCs (see Figure 2), instances and messages are not shown within an HMSC. This way, HMSCs can focus completely on the composition aspects. A more intuitive (non-standard) name for HMSCs is *road map* which gives a very good characterisation how HMSCs are used in practice.

HMSCs, like normal road maps, may easily become quite complex if no further structuring mechanism is employed. Fortunately, HMSCs are hierarchical in the sense that a reference in an MSC may again refer to an HMSC. Within the reference 'MONITORING' this refinement mechanism is employed. This demonstrates that MSC'96 in general supports top down design nicely (apart from a few shortcomings discussed below).

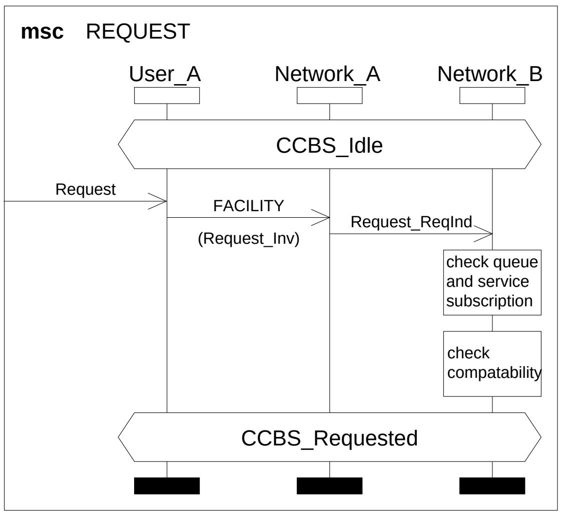


Figure 2

```
msc   REQUEST;
      inst          User_A, Network_A, Network_B;
      User_A:       instance;
      Network_A:    instance;
      Network_B:    instance;
      all:          condition CCBS_Idle;
      User_A:       in Request from env;
                    out FACILITY (Request_Inv);
      Network_A:    in FACILITY (Request_Inv);
                    out Request_ReqInd;
      Network_B:    in Request_ReqInd;
                    action check queue and service subscription;
                    action check_compatibility;
      all:          condition CCBS_Requested;
      User_A:       endinstance;
      Network_A:    endinstance;
      Network_B:    endinstance;
endmsc;
```

Figure 3 : MSC/PR representation of Figure 2

3.2 MSC 'REQUEST'

When the network encounters a busy destination B it retains the call information for the CCBS supplementary service for a certain period. During this time, user A can activate the CCBS supplementary service. The request for the CCBS service is described in Figure 2.

MSC 'REQUEST' is an example for a basic MSC containing only language con-

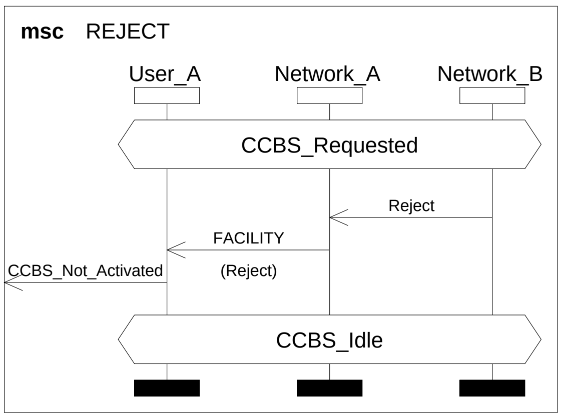


Figure 4

structs which have been already included in MSC'92, namely *environment* (frame around the diagram), *instance* (vertical line with horizontal bars on top and bottom), *message* (arrow), *action* (rectangle) and *condition* (hexagon) [20].

Both conditions 'CCBS_Idle' and 'CCBS_Requested' are *global*: they refer to all instances contained in the MSC. In MSC'96 *global* is always interpreted on the level of the MSC document since the composition rules (HMSC static semantics rules) do not discern the instances. Thus, all of them are attached to the *global* condition. This differs from the composition rules stated in MSC'92 where conditions are always discriminated by the set of instances to which they are attached.

In case of MSC 'REQUEST' also the textual representation MSC/PR has been provided whereby the *event oriented form* which is new in MSC'96 has been employed. The MSC/PR contained in MSC'92 lists message sending and receiving events in association with an instance (*instance oriented form*). During the last ITU study period, a better readable notation was requested, in particular in cases where MSC/PR is not only used internally by tools, but also edited by humans. Accordingly, a new event oriented textual representation was elaborated where events are listed in form of a possible execution trace and not ordered with respect to instances. The event oriented textual syntax is closer to the graphical grammar than the instance oriented textual syntax. This has the advantage that details of the graphical representation are expressible more easily. In MSC'96, both the instance oriented and the event oriented syntax form are combined within one textual representation.

3.3 MSC 'REJECT'

The CCBS request may be rejected in case no compatible terminal exists at destination B. Another reason for a rejection is if the maximum number of requests against destination B already is queued. The rejection procedure is shown in Figure 4.

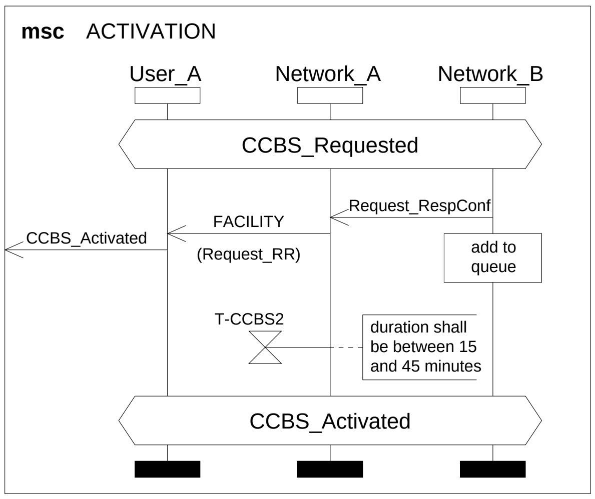


Figure 5

3.4 MSC 'ACTIVATION'

Figure 5 describes the acceptance of the CCBS service request. In this case the network registers the CCBS request (add request to queue) and the user is informed that the request was successful and the CCBS service now is activated. Furthermore the CCBS service duration timer T-CCBS2 is started.

MSC 'ACTIVATION' contains the setting of a timer T-CCBS2. The new timer symbols are considerably more intuitive than the graphical symbols used in MSC'92. Contrary to MSC'92, the individual timer constructs of MSC'96, i.e., *timer set*, *timer reset*, and *timeout*, may be split between different MSCs.

The setting of a timer is represented graphically by an hour glass symbol connected with the instance axis by a (bended) line symbol (see Figure 5). The timer reset is represented by a cross symbol ($\times$), again connected with the instance axis by a line symbol (cf. Figure 10). Time-out is described by a (bended) arrow which is connected to the hour glass symbol (see Figure 8 and 12).

3.5 HMSC 'Monitoring'

The reference 'MONITORING' in Figure 1 again refers to an HMSC which is shown in Figure 6. The HMSC starts with the condition 'CCBS_Activated'. The status check of user B (reference 'CHECK_STATUS_B') leads either to the result that B is free (reference 'REPLY_B_FREE') or that B is busy (reference 'REPLY_B_BUSY'). If B is busy the HMSC 'MONITORING' returns to the initial condition 'CCBS_Activated' and the 'CHECK_STATUS_B' procedure is repeated. After a successful status reply (reference 'REPLY_B_FREE'), the network repeats the status check for user B after a certain time interval (supervised by the *destination B idle guard timer*). This waiting period enables user B to initiate a call before any CCBS request is processed. Subsequently, the system gets into state 'CCBS_Await_Status', which means waiting

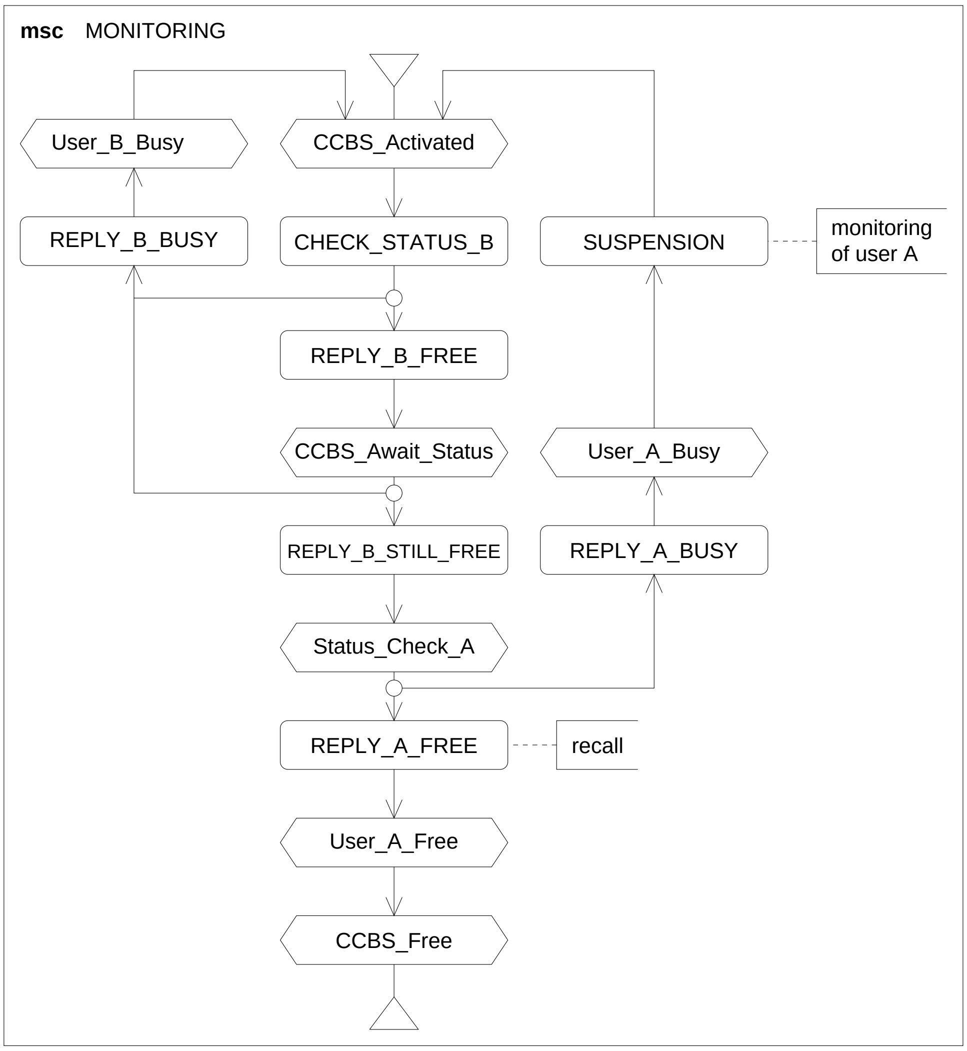


Figure 6

for another status reply from user B. In case, after expiration of the guard timer this second status request finds user B busy again (reference 'REPLY_B_BUSY'), the HMSC monitoring starts again from the beginning. However, if this second status request recognises B as free again (reference 'REPLY_B_STILL_FREE') the monitoring of user A is started and the system gets into the state 'Status_Check_A'. If user A is found to be free (reference 'REPLY_A_FREE') the CCBS recall of user A is initiated. Accordingly, the system state changes into 'CCBS_Free' which means waiting for a reply from user A. In case of user A being busy (reference 'REPLY_A_BUSY') the CCBS recall is suspended until user A is found to be free again (reference 'SUSPENSION') and the HMSC 'MONITORING' starts anew from the beginning.

The HMSC 'MONITORING' contains an end symbol (triangle). Note, that the HMSC contains adjacent conditions, namely 'User_B_Busy' and 'CCBS_Activated', and, 'User_A_Free' and 'CCBS_Free'. Such adjacent conditions may be introduced to provide some additional information about the system state, e.g., if user B is in the state 'User_B_Busy' the HMSC returns to the initial condition 'CCBS_Activated'. We have also some cases where the composition is not guarded by conditions, since MSC 'CHECK_STATUS_B' has no final and 'REPLY_B_BUSY' no initial condition. Also do references 'REPLY_B_FREE' and 'REPLY_B_STILL_FREE' not contain initial conditions. In MSC'92 such a composition was not allowed which forced users to employ conditions in a very strict manner.

The present MSC'96 supports the delayed choice outside of MSC references in HMSCs. Alternatives defined within a referenced MSC cannot be continued differently outside of the reference. Consequently, the MSC has to be split at the point where the decision is made. This refers particularly to the status request procedure for the users B and A in the MSCs 'CHECK_STATUS_B' and 'REPLY_B_STILL_FREE'. In both cases the split has to be made after sending the *status request* message. Accordingly, we have to distinguish between *busy* and *free* as status replies. This explains the modelling in HMSC 'MONITORING' where 'REPLY_B_FREE', 'REPLY_B_BUSY', 'REPLY_A_FREE' and 'REPLY_A_BUSY' appear as references to separate MSCs, i.e., they refer to Figure 8, 11, 10, and 12 respectively. In practice, exception handling becomes quite clumsy if all decisions have to be made outside of the references. In general, it leads to fairly small MSC pieces often containing one or two messages only. Therefore, this deficiency should be removed as soon as possible by an appropriate composition mechanism.

3.5.1 MSCs referred to by HMSC 'MONITORING'

HMSC 'MONITORING' (Figure 6) provides an overall view of the monitoring procedure for the users A and B. It refers to the concrete message exchange defined in the Figures 7, 8, 9, 10, 11, 12, and 13. These MSCs are now presented in more detail.

User B is monitored until it is not busy. As specified in MSC 'CHECK_STATUS_B' (Figure 7), that means a status request is sent by network B to a status request process. Within the subsequent final state 'CCBS_Await_Status', network B is waiting for a status reply.

If user B turns out to be free the network B starts the destination B idle guard timer T-CCBS4. This timer enables destination B to initiate a new call before the pending CCBS request is processed.

The corresponding MSC 'REPLY_B_FREE' (Figure 8) contains a local condition 'User_B_Free' attached to instance 'Network_B' only. This condition is used to indicate a local state. In addition, this MSC contains a timer expiration construct for T-CCBS4, i.e, a combination of timer setting and time-out (the latter is represented by an arrow pointing to the instance).

When the destination B guard timer T-CCBS4 expires and B is still not busy, network A is informed (cf. MSC 'REPLY_B_STILL_FREE', Figure 9). Subsequently, user A is monitored by sending a status request to user A. The status check timer T-CCBS1 is set supervising the maximum response time for user A. MSC 'RE-

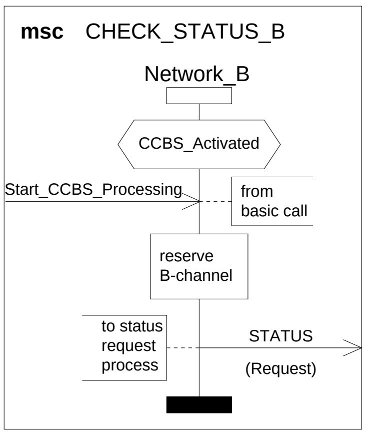


Figure 7

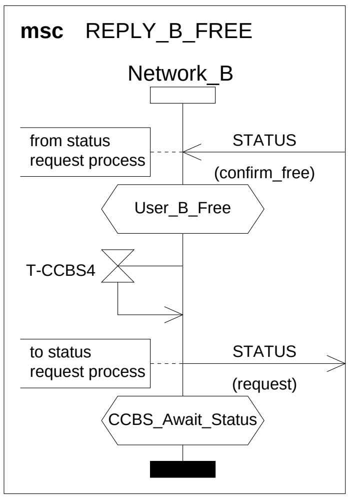


Figure 8

PLY_B_STILL_FREE' shows the example of an intermediate (i.e., neither initial nor final) condition: 'User_B_Free' which is non-global but also non-local, i.e., attached to more than one instance.

If user A is found not busy (cf. MSC 'REPLY_A_FREE', Figure 10) user A is recalled with an indication that it is a CCBS recall. At the same time the CCBS recall timer TCCBS-3 is started. MSC 'REPLY_A_FREE' shows a timer reset for the status check timer T-CSSB1 which is graphically represented by a cross symbol connected to the instance by a line symbol.

If the destination guard timer T-CCBS4 (which was set in MSC 'REPLY_B_FREE') expires and user B is found busy again (cf. MSC 'REPLY_B_BUSY', Figure 11), processing of the destination B-CCBS stops. The network monitors again destination B, i.e., the monitoring procedure continues with MSC 'CHECK_STATUS_B'.

If user A is found busy at the time of a recall then it is notified and the CCBS request is suspended. The corresponding MSC 'REPLY_A_BUSY' is shown in Fig. 12.

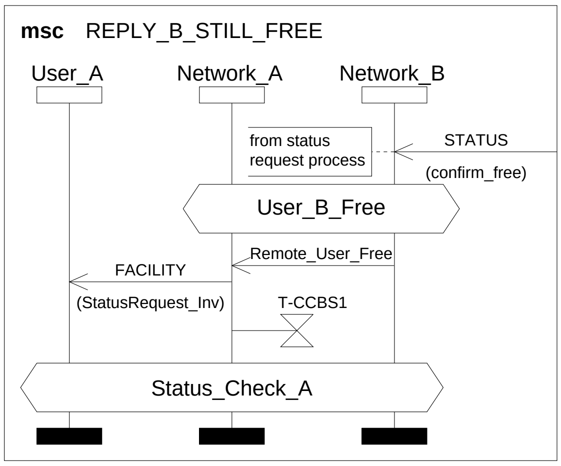


Figure 9

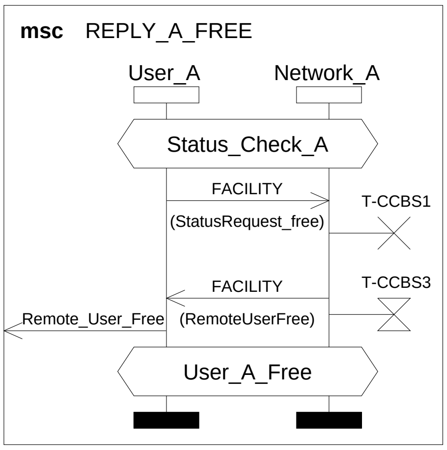


Figure 10

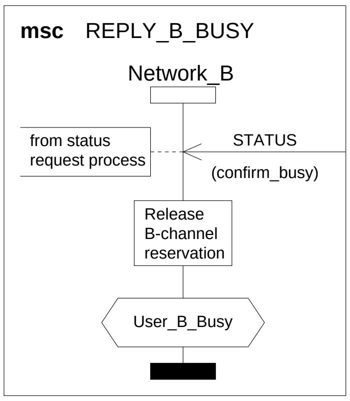


Figure 11

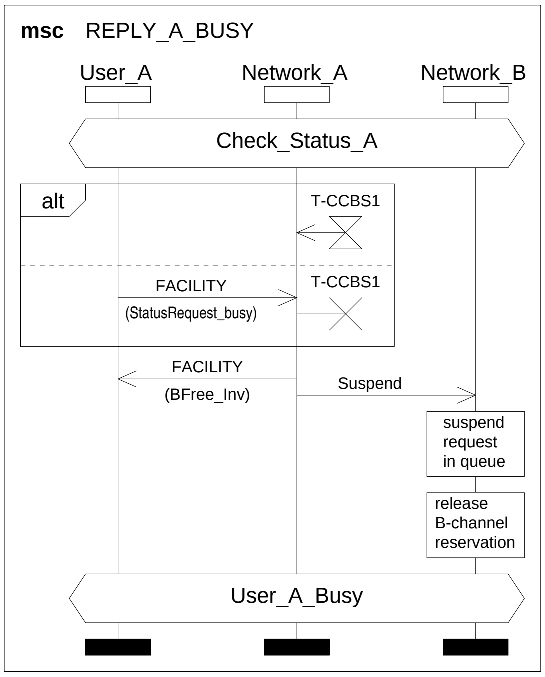


Figure 12

The MSC contains an *inline operator expression* referring to alternative composition. Graphically, the inline expression is described by a rectangle with dashed horizontal lines as separators. The operator keyword is placed in the left upper corner.

Inline operator expressions in MSC'96 allow the five operator keywords *alt*, *par*, *loop*, *opt*, *exc* which denote alternative composition, parallel composition, iteration, optional region and exception, respectively. MSC references may contain corresponding textual operator expressions which in addition include the sequential operator using the keyword *seq*.

As shown in MSC 'SUSPENSION' (Figure 13), user A being found busy, is monitored until it becomes free. Then, user A's CCBS request shall become not suspended and the monitoring procedure shall start again.

MSC 'SUSPENSION' shows a nested inline expression. The alternative inline expression is embedded in an inline expression containing a loop operator. The loop operator denotes iteration whereby the range may be specified in parentheses. In MSC 'SUSPENSION', the range <0,inf> denotes that the loop will be executed a finite, but unbound number of times, zero times execution included.

3.5.2 Alternative modellings of MSC 'REPLY_A_BUSY'

The MSCs 'REPLY_A_BUSY', 'REPLY_A_BUSY_ALT1' and 'REPLY_A_BUSY_ALT2' (Figures 12, 14, 15) describe alternatives for modelling the special situation where user A is busy. All three contain an inline operator expression referring to alternative composition.

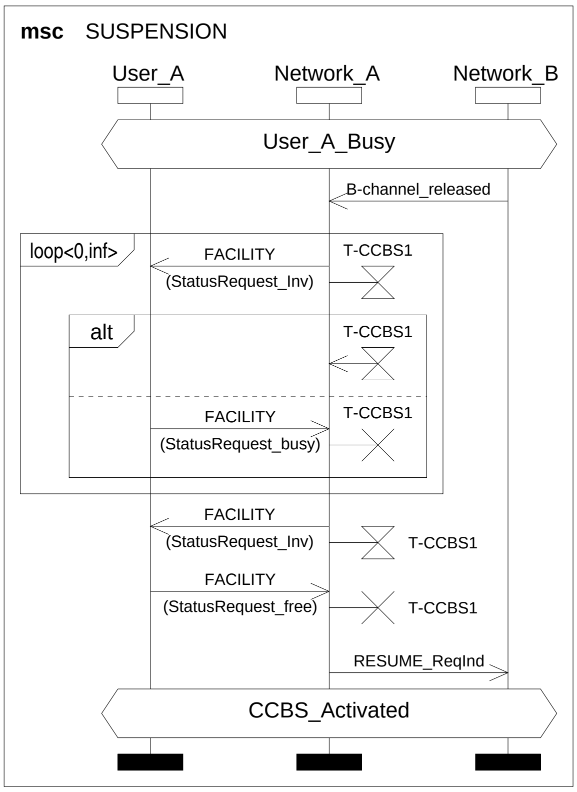


Figure 13

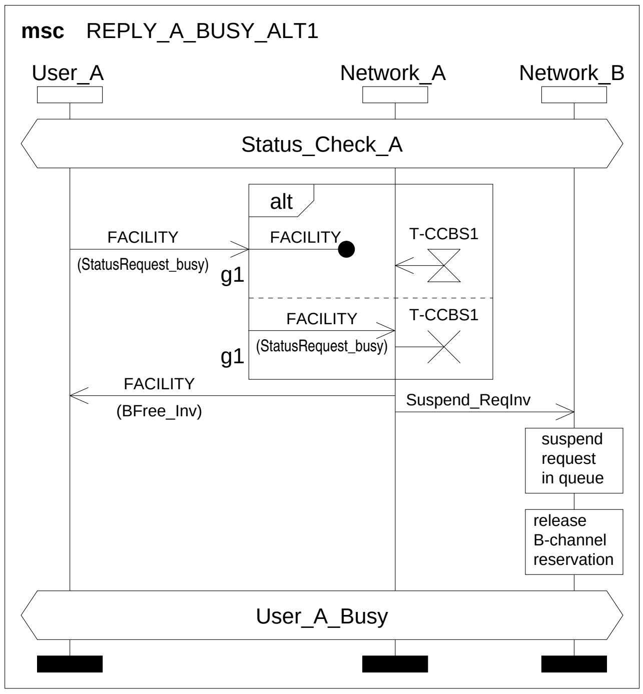


Figure 14

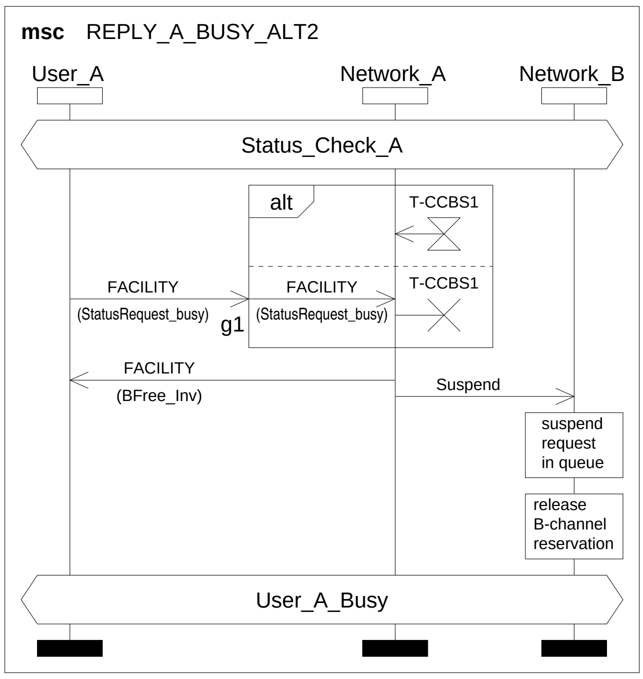


Figure 15

In Figure 14 and 15 messages may enter an inline expression via *gates*. E.g., in MSC 'REPLY_A_BUSY_ALT1' the message 'FACILITY(StatusRequest_busy)' is entering the inline expression via gate 'g1'.

Gates are used to define connection points for messages and order relations with respect to the interior and exterior of MSC references and inline expressions. Gates on inline expressions are merely transit points on the frame of the inline expression. If the gate is not continued outside the frame, the following implicit rules apply:

1. If there are other gates with the same name of the same inline expression, the continuation given for one of the gates holds for all.
2. If there are no other gates with the same name and no continuation exists, an implicit continuation to the next enclosing frame (either MSC frame or inline expression frame) is assumed.

A message gate name can be defined explicitly by a name associated with the gate on the frame or implicitly by the direction of the message through the gate and the message name.

The three MSCs describe slightly different situations: in all three cases either the timer T-CCBS1 waiting for an answer from user A expires (alternative 1) or the *status busy*-message 'FACILITY(StatusRequest_busy)' arrives (alternative 2).

In MSC 'REPLY_A_BUSY_ALT1' the *status busy*-message is lost in case of the first alternative (graphically represented by a black hole). A lost message in this case may also mean that it is discarded after the timer has expired. A more appropriate

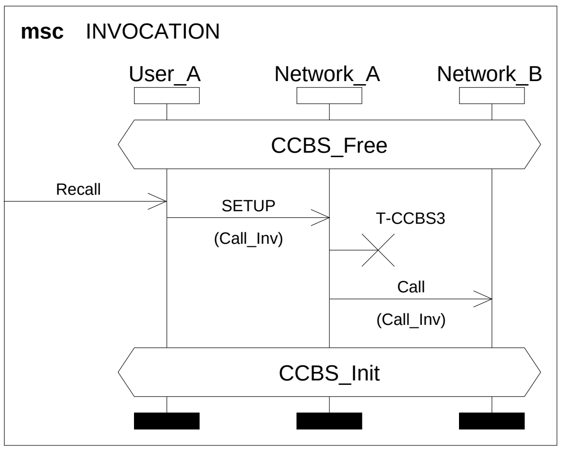


Figure 16

representation would be to attach the black hole to the instance which is not allowed, however, in the present MSC'96.

In MSC 'REPLY_A_BUSY_ALT2' the *status busy*-message is sent in both cases but if the first alternative, i.e., the expiring of the timer holds, the message cannot enter the inline expression since there is no corresponding gate. This kind of modelling is not excluded in MSC'96 but should be used with care. The static semantics rules in Z.120 are likely to be changed to rule out those situations.

In MSC 'REPLY_A_BUSY' the *status busy*-message is neither sent nor consumed in case of alternative 1. Note, that in this case the inline expression is attached to two instances 'User_A' and 'Network_A'.

3.6 MSCs 'INVOCATION', 'RELEASE', 'CANCEL', and 'DEACTIVATION'

Let us return to the CCBS example. The HMSC 'MONITORING' describes part of the overall CCBS service behaviour specified in the HMSC 'CCBS_SERVICE' (Figure 1). We already explained the HMSC 'MONITORING', the MSCs 'REQUEST', 'REJECT', and 'ACTIVATION' which all are referred to in 'CCBS_SERVICE'. In this section we describe the remaining MSCs, namely 'INVOCATION', 'RELEASE', 'CANCEL', and 'DEACTIVATION'.

3.6.1 MSC 'INVOCATION'

If user A accepts the recall before the CCBS recall timer expires (cf. MSC 'INVOCATION', Figure 16) the network initiates the CCBS call to destination B.

3.6.2 MSC 'RELEASE'

If the CCBS queue has been processed then processing is complete, the resources reserved for the CCBS supplementary service can be released (Figure 17), and the

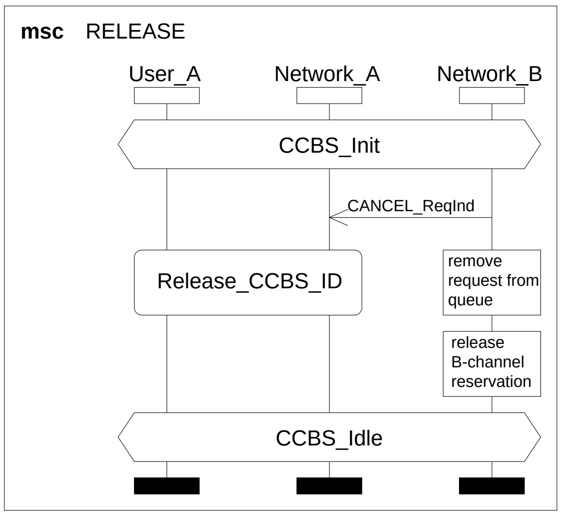


Figure 17

CCBS service returns into its initial state 'CCBS_Idle' (see also Figure 1).

Within MSC 'RELEASE', an MSC reference 'RELEASE_CCBS_ID' is specified. The corresponding MSC is shown in Figure 18. The MSC 'RELEASE_CCBS_ID' contains an inline expression with an *opt* operator. *opt* is an unary operator with one operand only. The *opt* operator denotes an alternative where the second operand is the empty MSC. In case of MSC 'RELEASE_CCBS_ID' the sending of the message 'CANCEL_ReqInd' is depending on a deactivation caused by user A. This is modelled in this tutorial by means of an optional region with a comment which strictly speaking demands a *guard*, instead. However, since guards need formal data definitions and therefore have not been included in MSC'96 we have chosen this kind of modelling.

3.6.3 MSC 'CANCEL'

If user A rejects the CCBS recall or the T-CSBS3 recall timer expires then the CCBS is deactivated. This is shown in Figure 19. The employment of the reference 'RELEASE_CCBS_ID' is an example for the reuse of MSCs by means of this language construct.

3.6.4 MSC 'DEACTIVATION'

In most situations (cf. Figure 1) the user can deactivate the CCBS service by sending a deactivation request. Upon successful deactivation, the corresponding CCBS request is discarded and user A is informed that the deactivation was successful. The corresponding MSC 'DEACTIVATION' is shown in Figure 20. Within this MSC, the MSC 'RELEASE_CCBS_ID' is referred to once again.

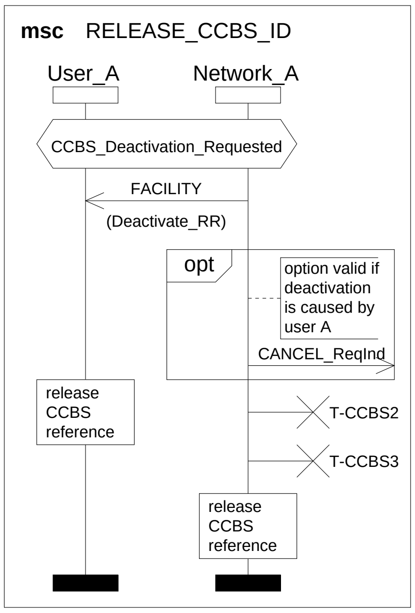


Figure 18

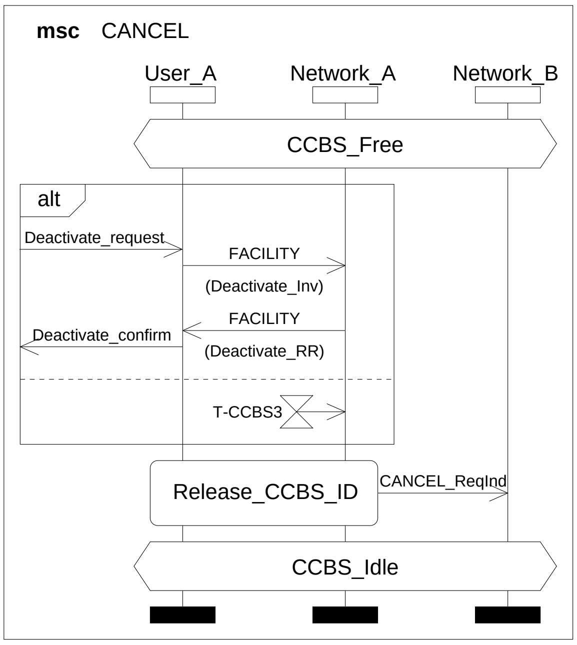


Figure 19

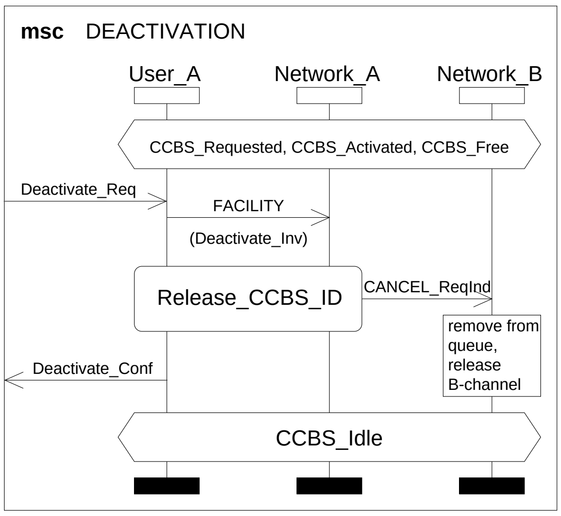


Figure 20

4 Further constructs of the MSC language

The CCBS specification in Section 3 is guided by the European Telecommunication Standard 300 359-1 [6]. Therefore, our example specification is restricted to the CCBS specific events. We stuck to this example in order to obtain an MSC specification transparent and compact enough for a tutorial presentation. In addition, this specification suits quite well to illustrate one of the intentions of MSC'96, namely to provide a complete specification of certain system features. How far this goal has been reached will be discussed in the conclusion.

However, this CCBS example is not sufficient to explain all existing MSC language constructs. In this section, the remaining ones are described. The MSC examples, given here, and the corresponding descriptions again refer to the CCBS service but this time, they do not display standardised behaviour.

4.1 HMSC 'TIME_SUPERVISION'

The idle guard timer (T-CSBS4) enables network B to initiate a call before any CCBS request is processed. In the HMSC 'TIME_SUPERVISION' (Figure 21), a connection setup of user B is specified in parallel with the time supervision and status check by Network B.

The HMSC 'TIME_SUPERVISION' contains a parallel frame embodying two small HMSC pieces which according to the semantics of this construct are executed in parallel (free merge). The small HMSC pieces refer to the MSCs 'TIMER' and 'CONNECTION_SETUP' provided in Figure 22 and Figure 23.

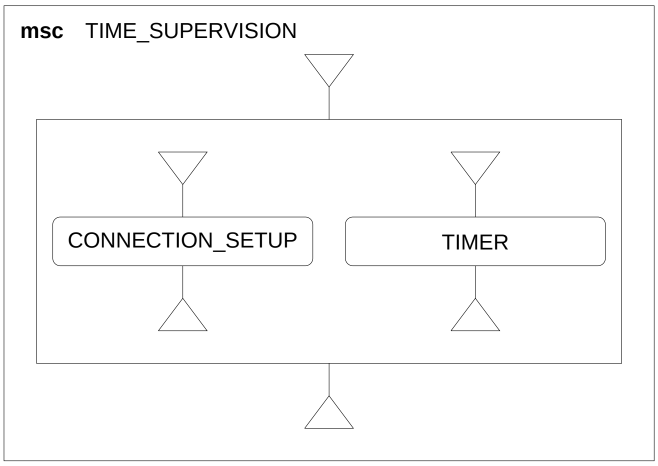


Figure 21

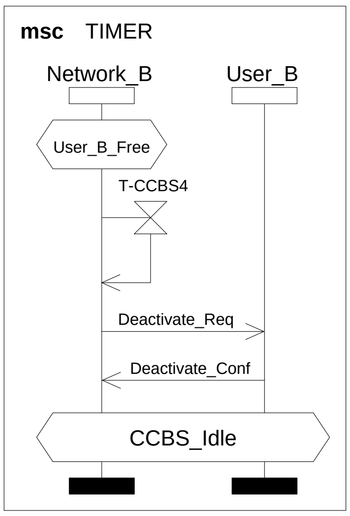


Figure 22

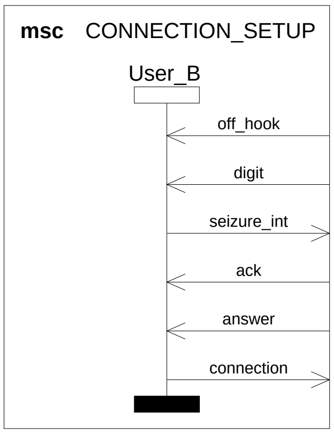


Figure 23

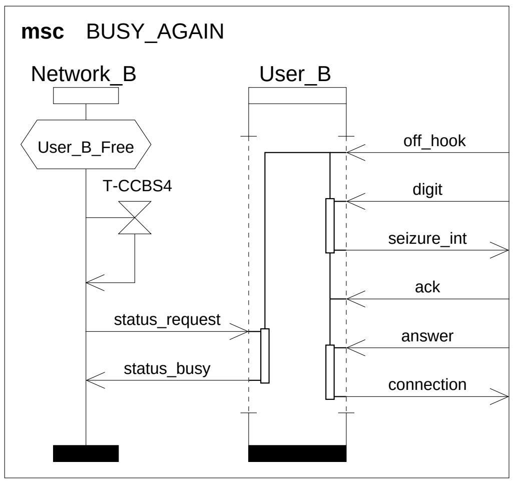


Figure 24

4.2 MSC 'BUSY_AGAIN'

If a connection setup procedure has been initiated before the status request is performed by Network B then the result is 'status_busy'. MSC 'BUSY_AGAIN' (Figure 24) provides an example for the modelling by means of generalised orderings graphically represented by means of *connections* between the message events. In this case, the connections are represented by line symbols with a staircase like shape. The connections are contained within a *coregion* in an instance in column form. The following partial ordering '<' is defined within MSC 'BUSY_AGAIN':

in off_hook < **in** digit < **out** seizure_int < **in** ack < **in** answer < **out** connection

in off_hook < **in** status_request < **out** status_busy

Going beyond MSC'96, a double vertical line taken over from object oriented Message Trace Diagrams (OMSC) [3, 4] shall denote a protected region which must not be interleaved by other events.

4.3 MSC 'BUSY_AFTER_FREE'

User B is not prevented from starting a setup after it has been found not busy (cf. MSC 'BUSY_AFTER_FREE', Figure 25). If destination B is again busy when the network attempts to make the CCBS call, then a special procedure has to be started which has been left out in the HMSC 'CCBS_SERVICE' (Figure 1).

4.4 MSC 'ABSTRACTION'

On an early stage of requirement specification one often abstracts from the internal message exchange while specifying the external behaviour only. On this level of abstraction, synchronisation constructs are demanded similarly to *Time Sequence Diagrams* [13] which impose a time ordering between events attached to different

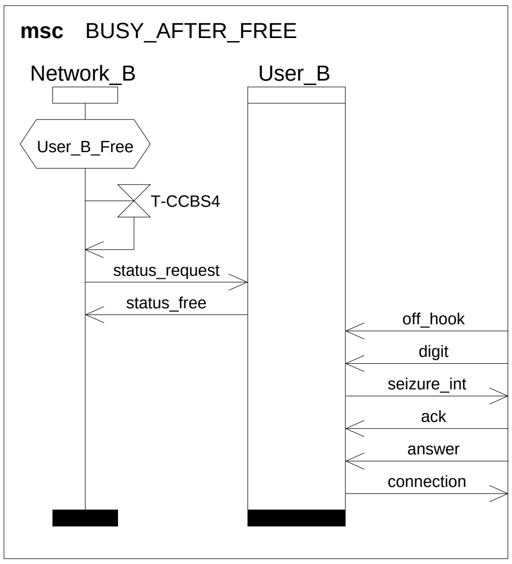


Figure 25

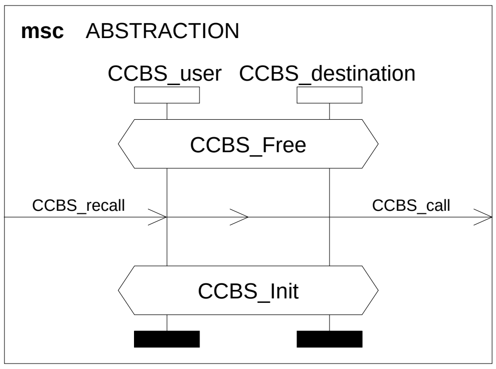


Figure 26

instances.[1] This kind of generalised ordering in MSC'96 is defined by means of connections graphically represented by a line symbol with an arrow symbol in its middle. Thus, in MSC 'ABSTRACTION' (Figure 26) defines the ordering:

in CCBS_recall < **out** CCBS_call

5 Conclusion and outlook

The modelling of the CCBS example by means of MSC'96 demonstrates the great suitability of the new language constructs for this purpose. In particular, HMSCs have shown to provide an overview about the functionality of such a protocol in a convincingly intuitive and transparent manner. This is quite important in practice because in SDL such a representation obviously is missing. Similarly to 'plain'

[1]Time Sequence Diagrams are frequently used for the specification of OSI services.

MSCs, HMSCs may supplement SDL specifications in many respects. Another very attractive feature of HMSCs is their hierarchical structuring, i.e., the possibility to refine HMSCs by other HMSCs which has been demonstrated within the CCBS example by the HMSC 'MONITORING' (Figure 6).

An obvious deficiency of HMSCs in MSC'96 is the lack of an appropriate exception handling within references. A choice made within a reference is not visible outside of the reference. As a consequence, all choices have to be made outside and that may lead to the definition of many small MSC(- pieces). Obviously one needs a means for guarding alternatives. There was already a proposal to use conditions again for guarding choices. For some reasons, this idea was not accepted within MSC'96. Certainly, formal data descriptions and corresponding parametrisation of references will be requested for this purpose during the next study period.

On the level of plain MSCs, inline operator expressions and MSC references have proven to provide an excellent means for a compact representation and for reusability. The inclusion of quite general gate concepts for inline expressions and references has contributed considerably to the power and expressiveness of MSC'96 [12].

The same concepts, however, may lead to the specification which are rather difficult to interpret. They may even contain deadlocks. This is a new situation, compared with MSC'92, where only deadlock-free MSCs could be specified. Certainly, the further elaboration of a corresponding formal semantics will promote the clarification of these language parts considerably. Nevertheless, in particular the gate concept combined with operator expressions and generalised ordering relations will remain a research topic for the next ITU- study period. In a sense, the inclusion of such far reaching concepts into the new standard may appear quite courageous. However, one has to keep in mind that standardisation is a highly interactive procedure which continuously needs feedback from users and tool makers. It also depends strongly on the effort provided by some few experts who again are depending on the support by their home organisations. This obviously implies that new parts of the language are not always completely settled, but still remain under development.

Whilst in MSC'96 very advanced concepts have been included which certainly need further elaboration, some important concepts have been left out because they seemed to be not sufficiently mature. The most prominent of these missing concepts are - apart from the above mentioned exception handling concepts - interruption and disruption operators and parallel composition concepts for HMSCs which include synchronisation mechanisms [19].

This wish list, of course, can be extended. The inclusion of formal data concepts has been mentioned already. A special language construct for the specification of synchronous communication mechanisms has still not been provided. In the context of object oriented modelling such a construct, essentially describing a remote procedure call presents the central communication mechanism.[2] There is of course a great demand for the inclusion of non-functional properties in MSC, most urgently

[2]However, a number of variants comes to mind immediately.

for performance evaluations [1]. All of these open items have been included in the working program for the next ITU- study period. An addendum to MSC'96 is planned for 1998 before the edition of the next MSC recommendation in 2000 (MSC'2000).

Compared with the extremely short time in which MSC'96 actually was produced, the result appears altogether to be surprisingly convincing and stable. The removal of deficiencies [16] but also the development of further language concepts like the incorporation of formal data concepts within MSC needs an intense input from users, tool makers and academic researchers. In this respect, the FORTE/PSTV'96 conference is the first opportunity to spread the *message* of *Message Sequence Charts* (MSC'96) to a broader community.

References

[1] R. Alur, G.J. Holzmann, D. Peled. *An Analyzer for Message Sequence Charts.* In: Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS95), Passau, March 1996.

[2] M. Andersson, J. Bergstrand. *Formalizing Use Cases with Message Sequence Charts.* Master Thesis, Lund Institute of Technology, 1995.

[3] G. Booch, J. Rumbaugh. *Unified Method for Object-Oriented Development.* Rational, 1996.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *A System of Patterns.* John Wiley & Sons, Chichester, 1996.

[5] V. Encontre, E. Delboule, P. Gavaud, P. Leblanc, A. Boussalem. *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.

[6] ETS 300 359-1. *Integrated Services Digital Network (ISDN); Completion of Calls to Busy Subscriber (CCBS) supplementary service; Digital Subscriber Signalling System No. one (DSS1) protocol; Part 1: Protocol specification.* European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France, Nov. 1995.

[7] J. Grabowski, P. Graubmann, E. Rudolph. *The Standardization of Message Sequence Charts.* In: Proceedings of the IEEE Software Engineering Standards Symposium 1993. Sept. 1993.

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

[9] J. Grabowski, D. Hogrefe, R. Nahm. *Test Case Generation with Test Purpose Specification by MSCs.* In: SDL'93 - Using Objects (O. Faergemand and A. Sarma, editors). North-Holland, Oct. 1993.

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

[11] P. Graubmann, E. Rudolph, J. Grabowski. *Towards a Petri Net Based Semantics Definition for Message Sequence Charts.* In: SDL'93 - Using Objects (O. Faergemand and A. Sarma, editors). North-Holland, Oct. 1993.

[12] O. Haugen. *Using MSC-92 effectively.* In: SDL'95 - Proceedings of the 7.th SDL Forum in Oslo, Norway (R. Braek and A. Sarma, editors). North-Holland, Sep. 1995.

[13] ISO/IEC JTC 1/SC 21. *Information Technology - OSI Service Conventions.* Revised Text of CD 10731, ISO/IEC JTC 1/SC21 N 6341, January 1991.

[14] I. Jacobson. *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, 1992.

[15] G.C. Kessler. *ISDN* (second edition). McGraw-Hill Inc., New York, 1993.

[16] S. Loidl. *Interpretation und Werkzeugunterstützung von Message Sequence Charts (MSC'96)* (in German). Diploma thesis (in preparation), Technical University of Munich (Germany), November 1996.

[17] S. Mauw, M.A. Reniers. *An algebraic semantics of Basic Message Sequence Charts.* Computer Journal No. 37, 1994.

[18] S. Mauw. *The formalization of Message Sequence Charts* In: Computer Networks and ISDN Systems - SDL and MSC (Guest editor: O. Haugen). Volume 28 (1996), Number 12, June 1996.

[19] E. Rudolph, P. Graubmann, J Grabowski. *Message Sequence Chart: Composition Techniques versus OO-Techniques - 'Tema con Variazioni'.* In: SDL'95 - Proceedings of the 7.th SDL Forum in Oslo, Norway (R. Braek and A. Sarma, editors). North-Holland, Sep. 1995.

[20] E. Rudolph, P. Graubmann, J. Grabowski. *Tutorial on Message Sequence Charts.* In: Computer Networks and ISDN Systems - SDL and MSC (Guest editor: O. Haugen). Volume 28 (1996), Number 12, June 1996.

[21] Z.100 I (1993). *SDL Methodology Guidelines.* Appendix I to Z.100. ITU-T, Geneva, July 1993.

[22] Z.120 (1993). *Message Sequence Chart (MSC).* ITU-T, Geneva, Sep. 1994.

[23] Z.120 (1996). *Message Sequence Chart (MSC).* ITU-T, Geneva, April. 1996.

[24] Z.120 B (1995). *Message Sequence Chart Algebraic Semantics.* ITU-T, Geneva, 1995.