a t m o s p h e r e

Esprit Project 2565

Telecommunication System Development Based on Message Sequence Charts and SDL

Peter Graubmann, Ekkart Rudolph and Jens Grabowski

SIEMENS AG, ZFE IS SOF 1
Otto-Hahn-Ring 6, D-8000 München 83
Tel: +49 89 636 42243
E-mail: rudolph@ztivax.uucp

Abstract

For a long time Message Sequence Charts (MSCs) have been part of auxiliary diagrams within the SDL-recommendations, but only recently their standardization in graphical and textual form has been decided within the CCITT. MSC is a trace language which in graphical form admits a particularly intuitive representation of system runs in distributed systems. Within a software development process for (tele)communication systems MSCs are used primarily for the requirements definition describing the required system behaviour in form of traces. However, in practice the requirements definition by means of MSCs often was restricted to the specification of few selected system runs, the 'standard cases', since each MSC only describes a partial behaviour. In order to overcome this restriction, composition mechanisms are introduced by means of global and local conditions. Standard building blocks, so called Sequence Chart segments, are defined as means for structuring the composition and to obtain an overview about the specified system. The aim of this paper is to prove the usefulness of these concepts by applying then to the specification of the INRES-Service. Beyond that, the power of the composition rules is increased considerably by taking over the asterisk concept from SDL to MSC-conditions.

Contents

Telecommunication System Development Based on Message Sequence Charts and SDL

Peter Graubmann, Ekkart Rudolph
Siemens AG, ZFE IS SOF 1
Otto-Hahn-Ring 6
D-8000 München 83

Jens Grabowski
Universität Bern
Längassstrasse 51
CH-3012 Bern

Abstract: For a long time Message Sequence Charts (MSCs) have been part of auxiliary diagrams within the SDL-recommendations, but only recently their standardization in graphical and textual form has been decided within the CCITT. MSC is a trace language which in the graphical form admits a particularly intuitive representation of system runs in distributed systems. Within a software development process for (tele)communication systems MSCs are used primarily for the requirement definition describing the required system behaviour in form of traces. However, in practice the requirement definition by means of MSCs often was restricted to the specification of few selected system runs, the 'standard cases', since each MSC only describes a partial system behaviour. In order to overcome this restriction, composition mechanisms are introduced by means of global and local conditions. Standard building blocks, so called Sequence Chart segments, are defined as a means for structuring the composition and to obtain an overview about the specified system.
The main aim of this paper is to prove the usefulness of these concepts by applying them to the specification of the INRES-Service. Beyond that, the power of the composition rules is increased considerably by taking over the asterisk concept from SDL to MSC-conditions.

1. Introduction

Within the software life cycle increasing attention is paid to the stage of software design since the quality of all following stages is depending on it. In particular in the field of telecommunication this has been taken into account by the introduction of a special design language 'SDL' (*S*pecification and *D*escription *L*anguage). An SDL-design, however, is useful only if it is checked with respect to syntactical and semantical correctness. Apart from a general correctness proof (e.g. absence of deadlocks) the consistency of the SDL-specification with respect to a prescribed system behaviour has to be checked. A convenient way to describe the system behaviour is offered by system traces which are suitably presented in form of signal flow diagrams called Message Sequence Charts (MSCs). MSCs are a widespread means for the description and particularly graphical visualization of selected system runs within distributed systems, especially telecommunication systems. A Message Sequence Chart shows sequences of messages interchanged cation systems. A Message Sequence Chart shows sequences of messages interchanged between entities (such as SDL services, processes, blocks) and their environment (cf. figure 2.2). Formally MSCs describe the causal partial ordering of message events i.e. message sending, message reception and consumption.

MSCs have been used for a long time by CCITT Study Groups in their recommendations and within industry, according to different conventions and under various names such as Signal Sequence Chart, Message Flow Diagram and Arrow Diagram. Only recently the development of a standardized form has been tackled [11, 12]. The recommendation for the new CCITT–standard language MSC will be provided in 1992, the end of the present CCITT–study period.

The reason to standardize MSCs is to make it possible to provide tool support for them, to exchange MSCs between different tools and to ease the mapping to and from SDL specifications.

One part of the standardization work is to provide a clear definition of the meaning of a MSC. This is done by means of relating MSCs to SDL specifications (cf. figure 2.1), as follows:

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

According to the above definition, a MSC can be derived from an existing SDL system specification. However, a MSC is generally created before the system specification, and then serves as

- a statement for requirements for SDL specifications, or
- a basis for automatic generation of skeleton SDL specifications, or
- a basis for selection and specifications of test cases, or
- a semi–formal specification of communication, or
- an interface specification.

2. Message Sequence Charts

2.1 Basic Concepts

Message Sequence Charts show the signal flow between entities like processes. In general, MSCs are related to SDL–systems. Let us consider the MSC in figure 2.2 which describes a selected trace piece of the connection set–up in the INRES–Service specification [2]. It could equally be represented using SDL–diagrams with certain additions / modifications (cf. figure 2.1, dashes stand for not followed branches, bold arrows indicate the signal flow).

The diagram in figure 2.1 contains at least the same information as the MSC in figure 2.2. Obviously, however, the MSC is much more transparent, since it concentrates on the relevant information, namely the entities (INITIATOR, RESPONDER) and the signals involved in the selected trace piece (ICONreq, ICON, ICONind). Beyond that, what is even more important, the relation of MSCs to SDL–diagrams may be rather sophisticated. The entities very often are collections of (SDL–)processes on a higher level of abstraction such as blocks, since MSCs in general are created before the SDL–specification.

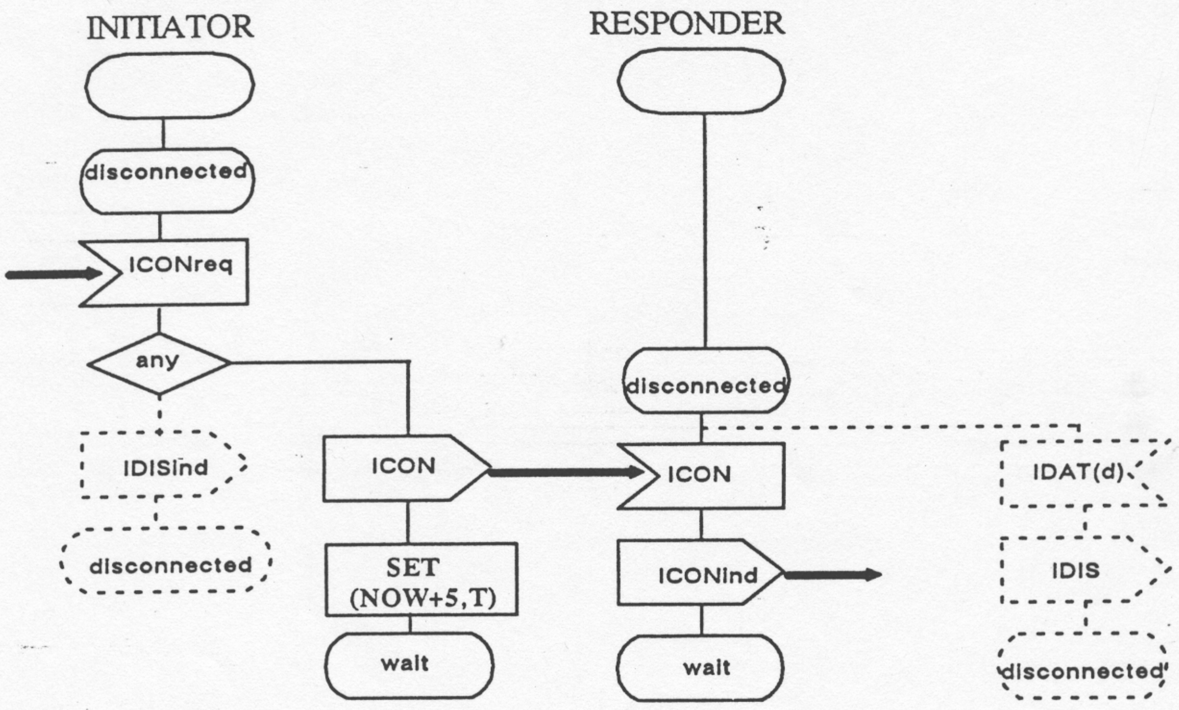


Figure 2.1: (Nonstandard) combined SDL – signal flow diagram

Nevertheless the correspondence between figure 2.1 and figure 2.2 may serve to give a good intuitive idea about the meaning of a Message Sequence Chart. It also demonstrates that a Message Sequence Chart describing one possible scenario can also be looked at as an SDL–skeleton [1,7].

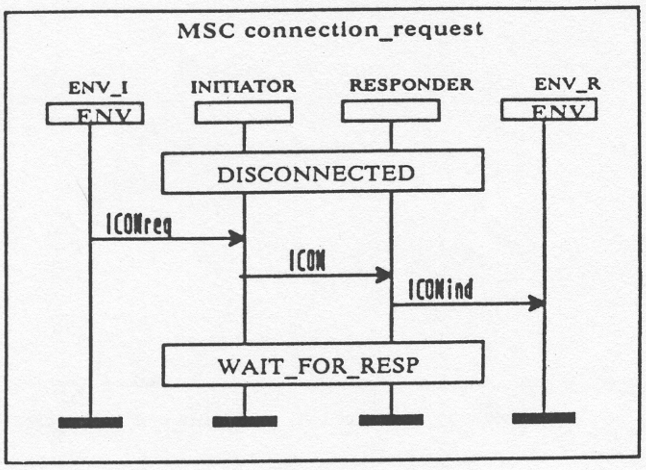


Figure 2.2: Message Sequence Chart

The most basic language constructs of Message Sequence Charts are entities, i.e. instances of certain SDL–types or environments, and messages describing the signal events. In the graphical form the instances and environments are represented by vertical lines or alternatively by columns. The signal flow is presented by horizontal arrows with a possible bend to admit signal overtaking or crossing. The message arrow denotes the signal consumption, the opposite end (message origin) the signal sending.
One MSC in general describes a small section of a 'complete system run'. Therefore the MSC has to be characterized by the specification of the initial and final conditions and possibly by intermediate conditions. Graphically a condition is characterized by a rectangle containing the condition name (cf. 'DISCONNECTED' in fig. 2.2).
In addition in MSCs the actions triggered by signal consumption and timeouts may be indicated.

2.2 Additional concepts

Sequence Charts in the pure message form may be extended by some additional concepts, mainly coming from SDL–process diagrams [5,6]. Eventually one trace in an SDL–system can be described completely by an *Extended Sequence Chart* (cf. fig. 2.3 which is the due extension of fig. 2.2). By means of the inclusion of an SDL–input–symbol both message reception and message consumption may be represented.
It should be noted that there is an essential difference between SDL–states used in Extended Sequence Charts (cf. fig. 2.3) and conditions used in MSCs : Whereas SDL–states only appear at the beginning of a state transition, MSC–conditions may appear everywhere between. Thus e.g. conditions may be used also to indicate the result of a decision. Therefore both concepts are graphically well distinguished.

For system engineering it is important that Extended Sequence Charts admit a stepwise refinement of the system specification. Thus within the software development process Extended Sequence Charts are used apart from requirement definition for system construction, analysis, simulation, animation and test case definition [6,8]. Both Message Sequence Charts and Extended Sequence Charts already are a central part of SDL-toolsets within industrial practice [6].

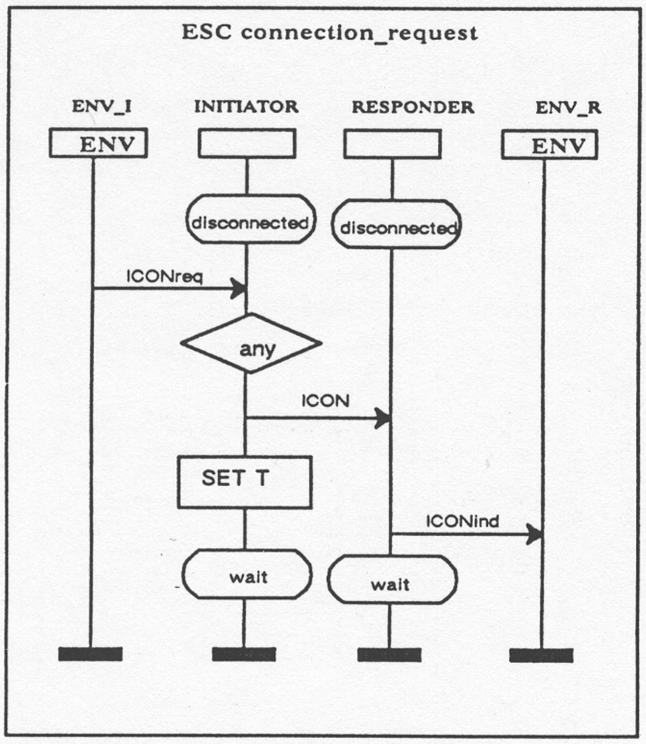


Figure 2.3: Extended Sequence Chart

3. Methods for system design using Message Sequence Charts and SDL

MSCs are commonly used as a requirement language to describe the purpose of a system in form of trace examples. As basis for implementation however, a different view is preferable. Such a view is offered by SDL which describes the system from the point of view of communicating processes. Within a software development process for telecommunication systems both MSC and SDL are used in parallel, supplementing each other and being correlated in many ways [6]. In the sequel a systematic MSC-methodology based on composition mechanisms is developed. The conditions introduced for this purpose may be employed also for the derivation of SDL-skeletons [1,7].

3.1 MSC-composition and decomposition

Since one MSC only describes a partial system behaviour, MSCs are primarily used for the description of selected normal cases. The shortcoming of MSCs in system engineering, namely the often restricted application to only a few 'standard' cases, has been pointed out several times in the literature [1]. In order to overcome this drawback, composition mechanisms have been introduced within the CCITT MSC–standardization document (cf. figure 3.1) [11]. For each MSC initial, intermediate and final conditions may be introduced defining possible continuations by means of condition name identification. In industrial practice the use of global conditions referring to global system states is most common. E.g. in figure 3.1 'DISCONNECTED' and 'WAIT_FOR_RESP' are global conditions. Opposite to the global condition, the local condition is attached to just one instance. In general, however, conditions may refer to an arbitrary subset of instances defined within the MSC in order to gain more flexible composition rules and also to support the constructional approach [11]. Examples for local conditions are provided in chapter 4. MSCs can be composed by (name)–identification of final and initial conditions. The other way round MSCs can be decomposed at intermediate conditions.

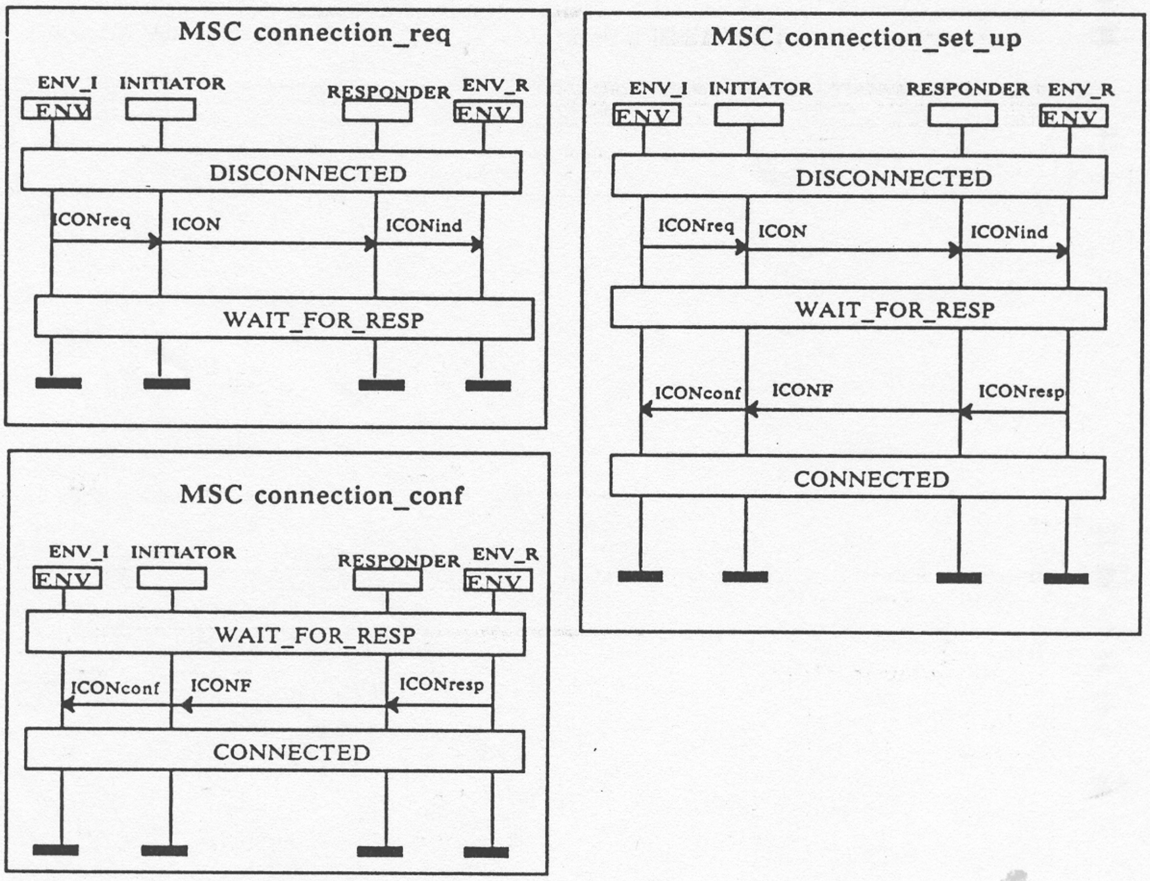


Figure 3.1: Message Sequence Chart–composition

3.2 Structured MSC–composition using Sequence Chart segments

A set of MSCs developed in an early stage of design, employing conditions for composition, may still lack sufficient transparency. Such a collection of MSCs in general can be rather unstructured: The chosen MSC–pieces may be too small to provide a good overview about the system. But also in the opposite case if the MSCs describe rather extensive system runs they often contain many large MSC–fragments in common (due to subcycles), a redundancy which could be avoided by an appropriate decomposition. In both cases the description does not give hints about its completeness.. Although composition rules are defined formally and hence all resulting system runs can be derived there is no sufficient overview about all of them.

In order to overcome these shortcomings standard building blocks, so called MSC–segments, are defined out of the given set of MSCs using possible decompositions and compositions. The resulting set of MSC–segments describes the same system behaviour as the original set of MSCs, taking into account the composition rules: in this sense the set of MSC–segments is equivalent to the original set of MSCs.

The set of these building blocks is maximal in the sense that all possible traces can be constructed out of its elements and minimal in the sense that no element can be composed from others in this set.

For the definition of MSC–segments the close relationship between SDL and Petri Nets, on the one hand, and between Message Sequence Charts and Petri net processes, on the other hand [5], has been exploited. For clarification it should be noted that Petri net processes describe system runs in Petri nets [8], whereas MSCs describe system traces of SDL–systems. Standard building blocks and a corresponding structured composition method has been suggested already for Petri net processes by means of process segments [3]. Process segments are basic behaviour structures of Petri net processes from which all possible Petri net processes can be composed. These ideas can be carried over to MSCs with MSC–segments as basic behaviour structures [10].

Basic behaviour structures essentially are maximal trace pieces which are inseparable in the sense that they always appear as a whole without (cyclic) inserts. They can be characterized in the following way: They describe

(a) (inseparable) sequential trace segments,
(b) (inseparable) cyclic trace segments,
(c) sections of sequential and
(d) cyclic trace segments in which other trace segments are embedded.

This is illustrated in figure 3.2 where the reachability graph of a given system is indicated schematically (the numbered circles refer to global system states). In this example, parallelism is hidden in global state transitions (described by MSCs). A simple comparison of the trace segments in figure 3.3 with the reachability graph shows that all possible traces can be composed from the presented segments. This construction procedure can be represented in form of a graph (fig. 3.4).

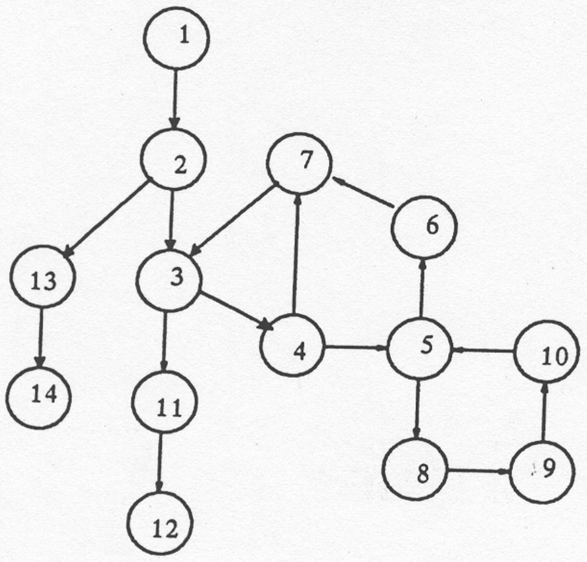
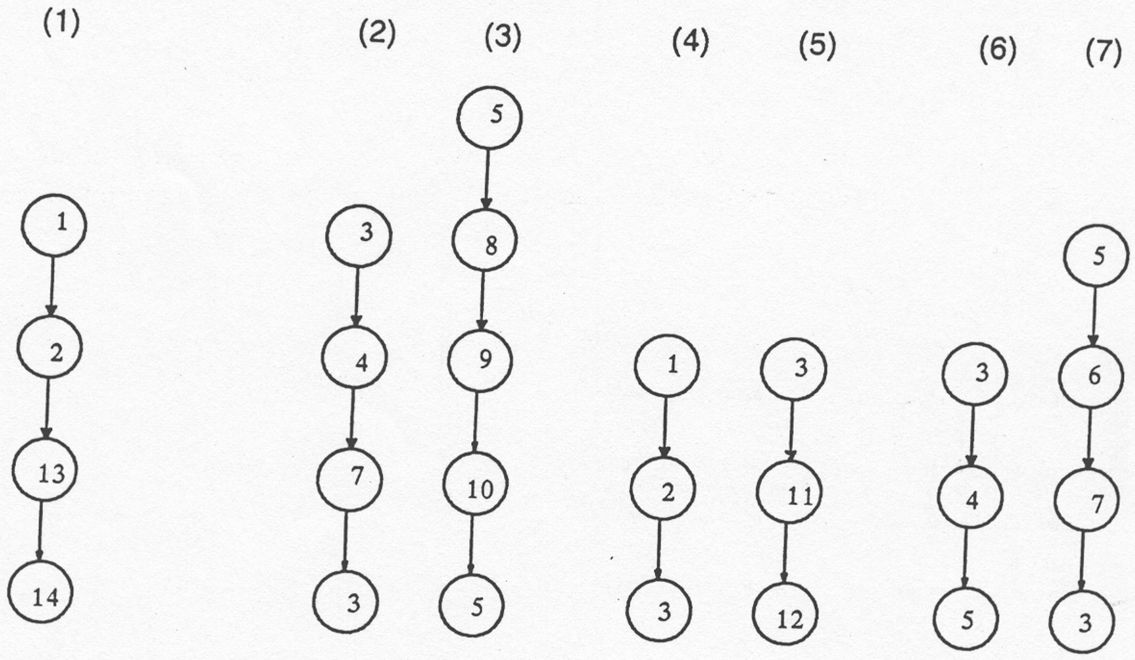


Figure 3.2: Reachability graph for a system



(a) charactestic inseparable sequential trace segment

(b) characteristic inseparable cyclic trace segment

(c) two sections of a sequential trace segment in which (2) (3) (6) and (7) are embedded

(d) two sections of a cyclic trace segment in which (3) is embedded

Figure 3.3: Trace segments for fig. 3.2

The figures 3.3 and 3.4 merely serve as an illustration of the mentioned ideas. For nontrivial systems the situation is not that simple.

Moreover, at the beginning of a system design the reachability graph which forms the basis for the generation of characteristic trace segments in the original definition of process periods [3] normally is not given. MSC–segments are defined in such a way that it is possible to generate them automatically from a given set of MSCs at each stage of system specification. Thus, MSC–segments represent standard building blocks which may be carried separately and in parallel with the specified MSC fragments.

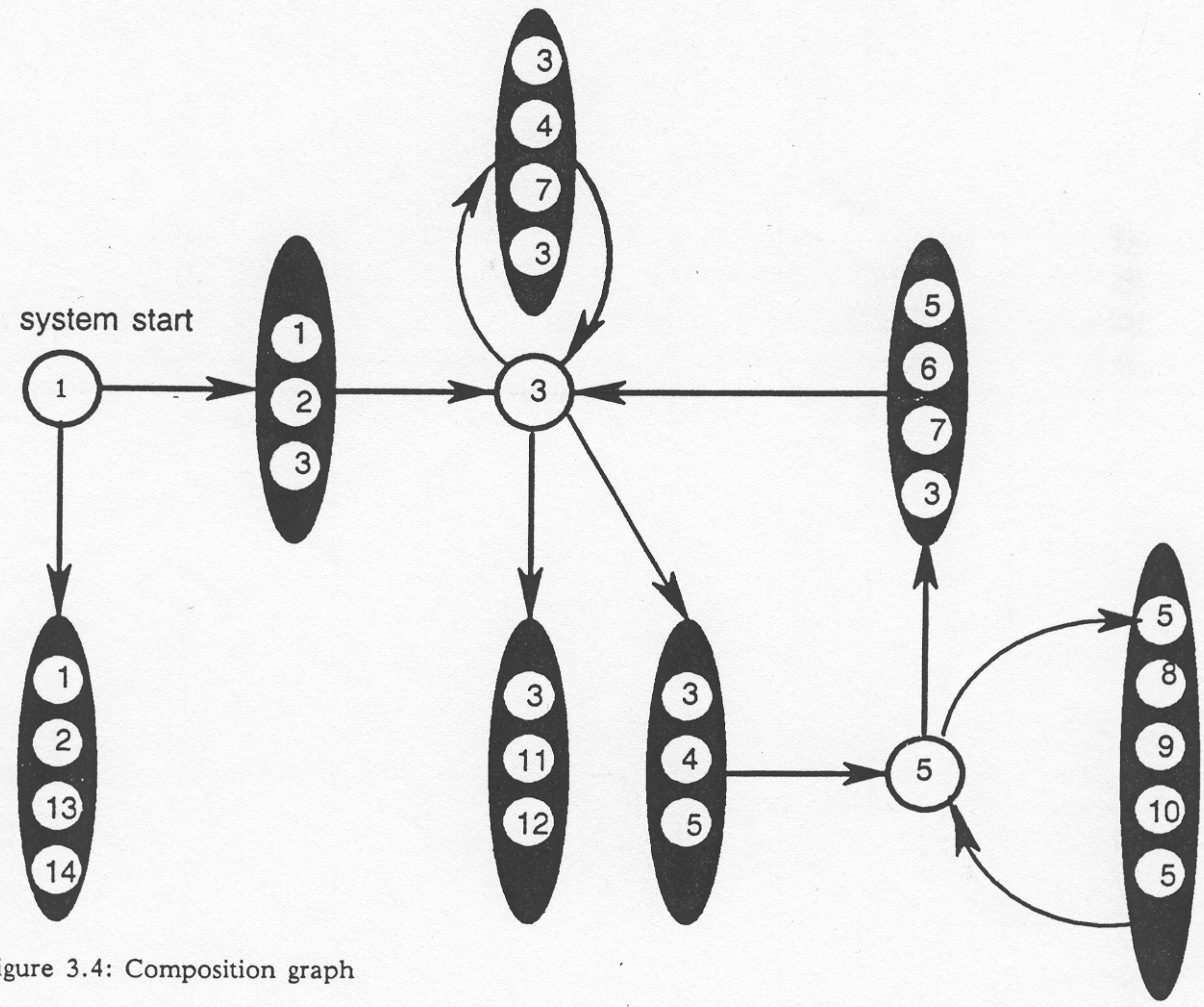


Figure 3.4: Composition graph

The set of generated MSC–segments represents the desired system behaviour. Beyond that, the computer aided composition of MSC–segments offers the possibility of an immediate simulation of the specified behaviour. Besides that, MSC–segments may be used for test case generation and test case selection. Viewing segments as the smallest units admits the abstraction from details of communication and the analysis of the essential behaviour structures of the system. MSC–segments are interesting also for system analysis providing an alternative to the reachability graph since they represent the concurrency aspects of the system in an obvious manner.

4. Example for MSC–composition and MSC–segments: INRES–Service:

As an example we use the requirement specification for the INRES–Service which shows a (simplified) connection set up and a following data transfer between INITIATOR and RESPONDER. Due to an unreliable medium between both the signal transfer may fail (cf. section 4.2 for failure cases) [2].
The MSCs specified in section 4.1 and 4.4 are closely related to the time sequence diagrams listed in [2]. However, compared with to time sequence diagrams they contain more information: besides actions and timeouts in particular conditions provide again the basis for composition mechanisms.

4.1 Standard cases

The first step is the specification of 'standard cases' (non–error cases). In order to obtain a description which can be enhanced easily by failure cases, local conditions are employed. However, if we would restrict ourselves to standard cases only, global conditions seem to be more transparent. For the INRES–Service example we obtain five standard cases (cf. fig. 4.1a – e).

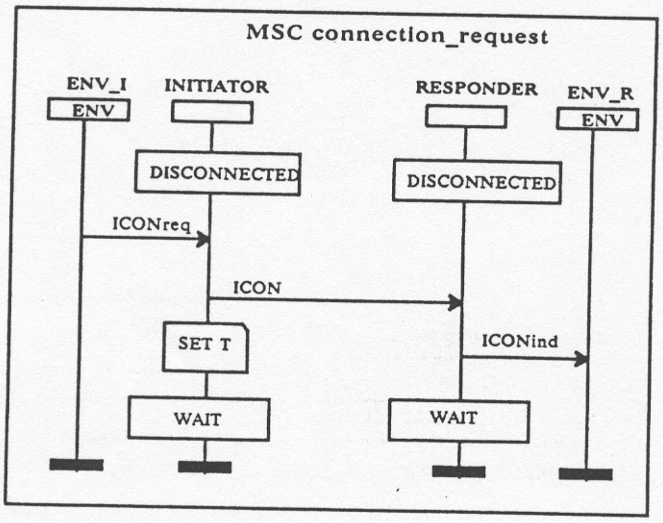


Figure 4.1a

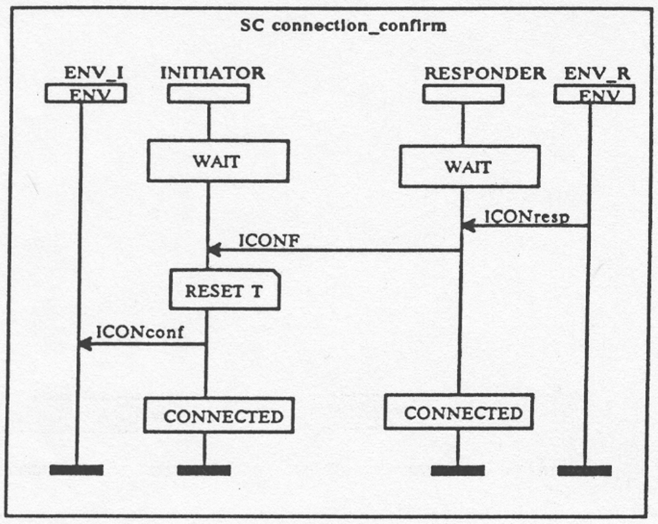


Figure 4.1b

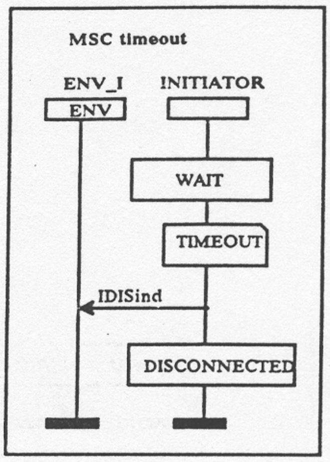


Figure 4.1c

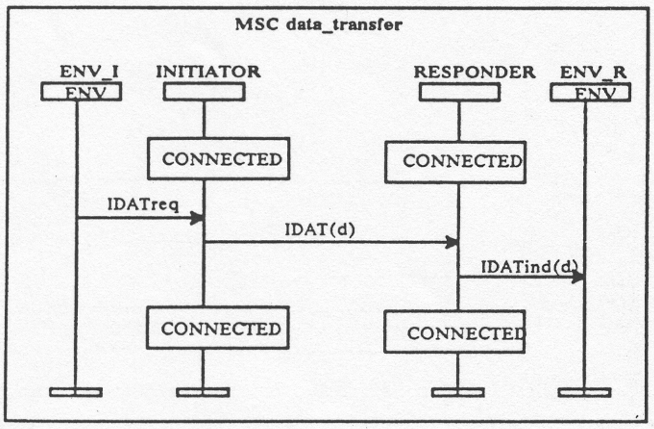


Figure 4.1d

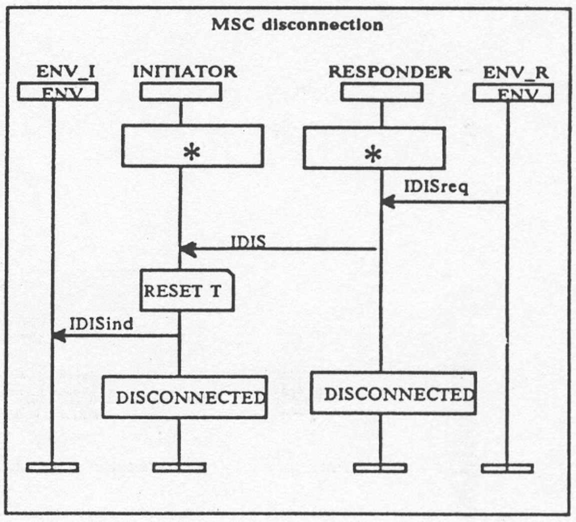


Figure 4.1e

4.2 Asterisk-Conditions

Within the MSC *disconnection* (cf. fig. 4.1e) we have taken over the asterisk-abstraction mechanism from SDL [2] to MSC-conditions. The asterisk-condition is a short-hand-notation to denote that for possible compositions the asterisk-condition can take on any name out of the set of specified condition-names (only conditions referring to the same set of instances are considered). E.g. for the MSCs specified in section 4.1 and their possible combinations the asterisk conditions may take on the following condition-names (in pairs for INITIATOR/RESPONDER): WAIT/WAIT, CONNECTED/CONNECTED, DISCONNECTED/DISCONNECTED, DISCONNECTED/WAIT. After including the failure cases of section 4.4. this set would be much enlarged. Analogously to SDL, exceptions may be specified for asterisk conditions. The asterisk construct is useful especially for the specification of system abortion, in a sense corresponding to the disabling–operator in LOTOS.

4.3 MSC–Segments

According to section 3.2 MSC–segments for the standard cases are constructed. This provides a first overview about the system (cf. fig. 4.2a – c)

For the construction of MSC–segments it should be noted that 'null–transitions' (signal–consumption causing no transition) are discarded.

We obtain five MSC–segments whereby MSC *data_transfer* (cf. fig. 4.2d) and MSC *disconnection* (cf. fig. 4.2e) already present segments.

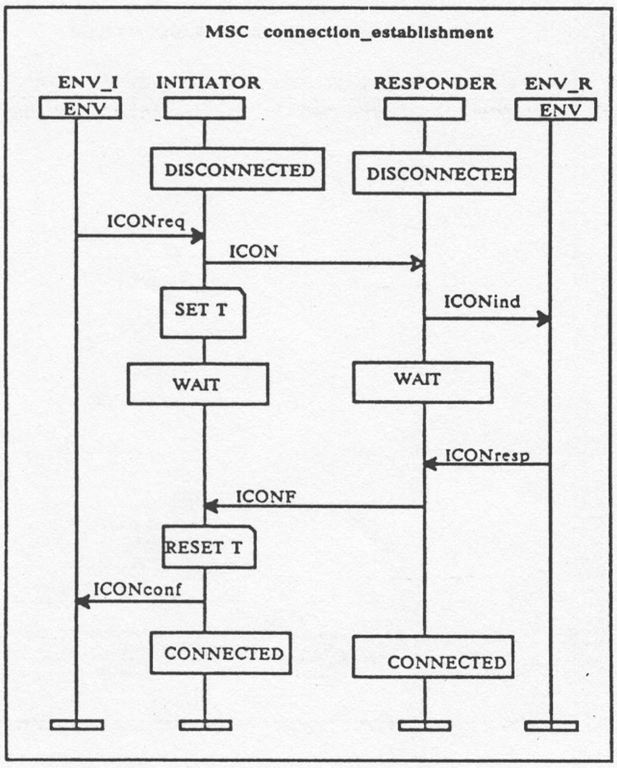


Figure 4.2a

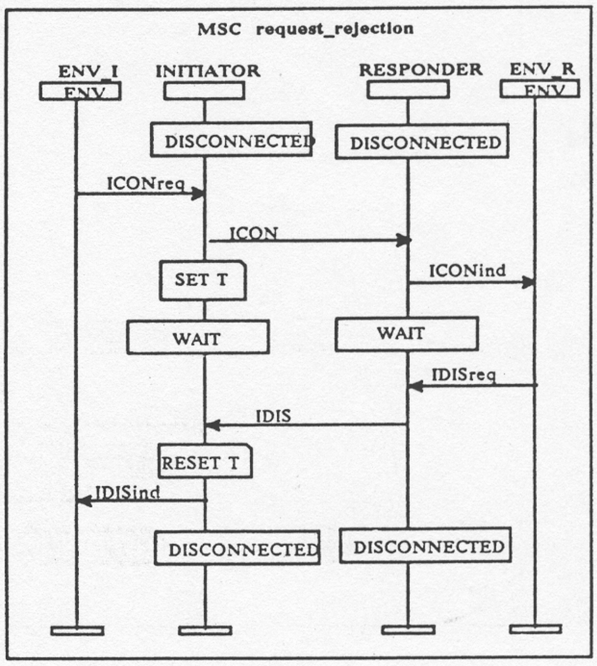


Figure 4.2b

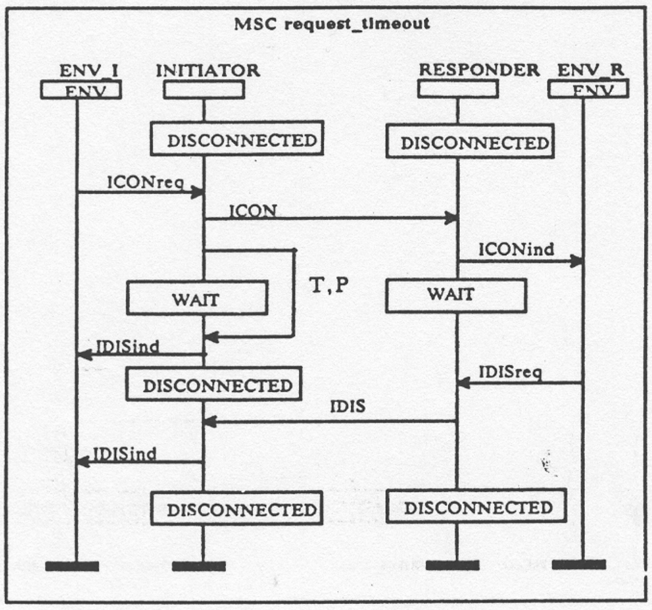


Figure 4.2c

In order to obtain an overview about all possible system runs the connection between the derived MSC–segments has to be shown. Such a description is provided in fig. 4.3 where the MSC–segments are represented as transitions within a graph .

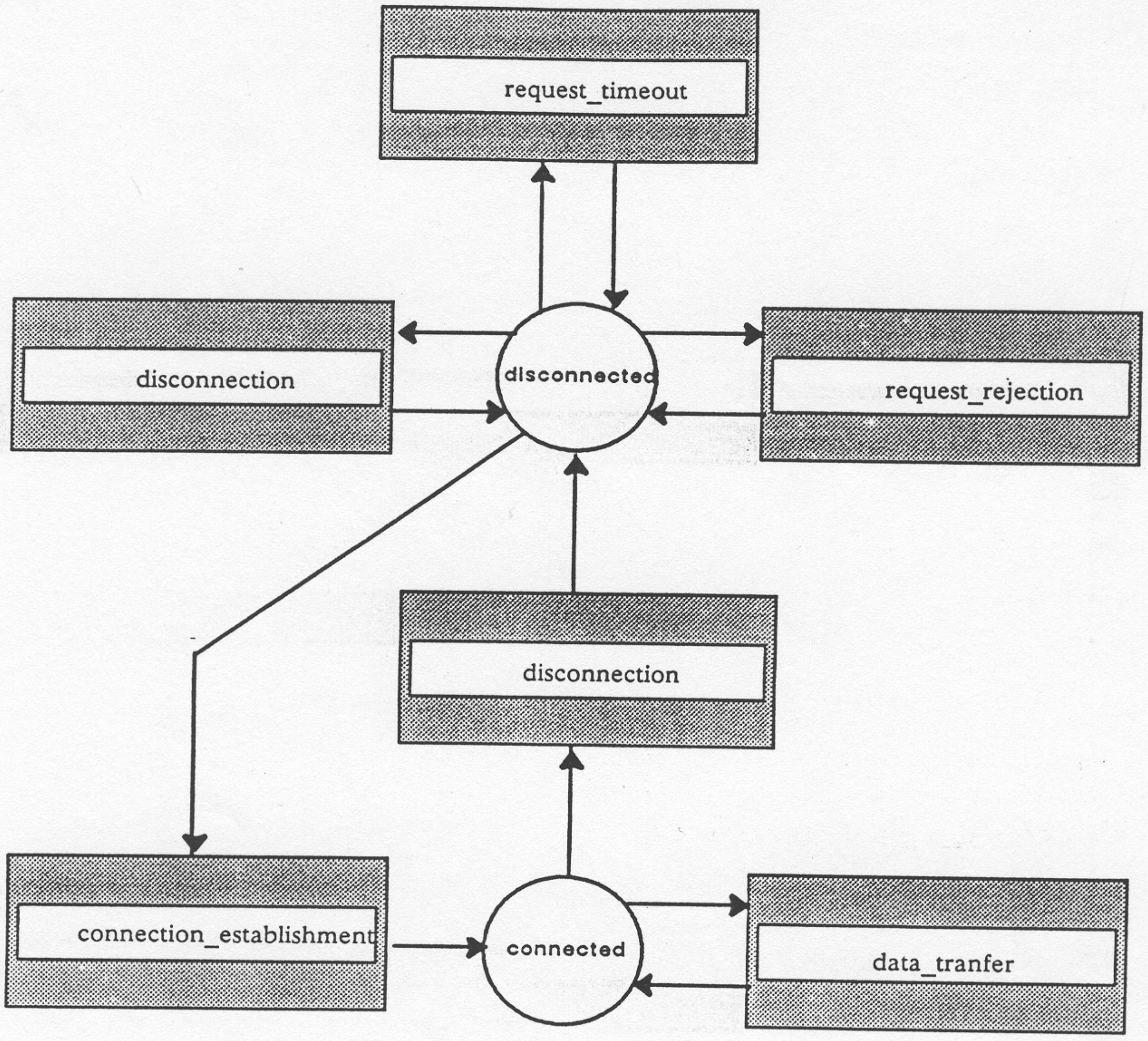

request_timeout
disconnection
disconnected
request_rejection
disconnection
connection_establishment
connected
data_tranfer

Figure 4.3

4.4 Failure cases

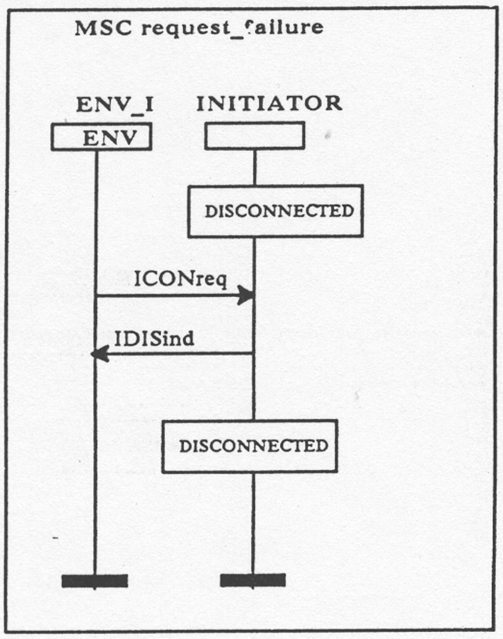


Figure 4.4a

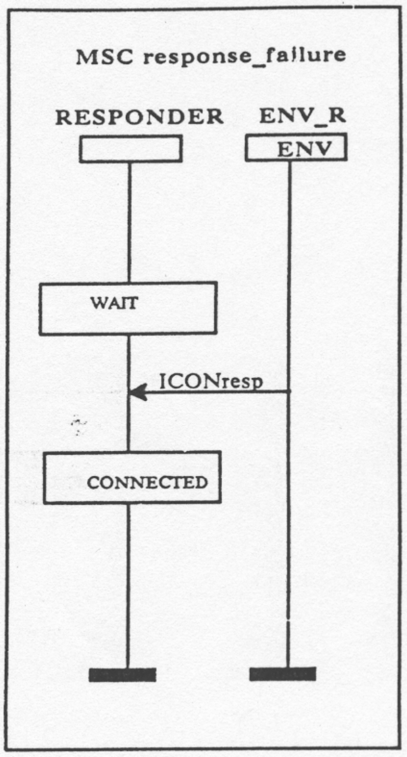


Figure 4.4b

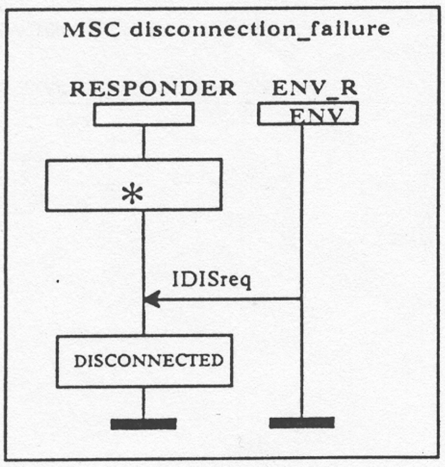


Figure 4.4c

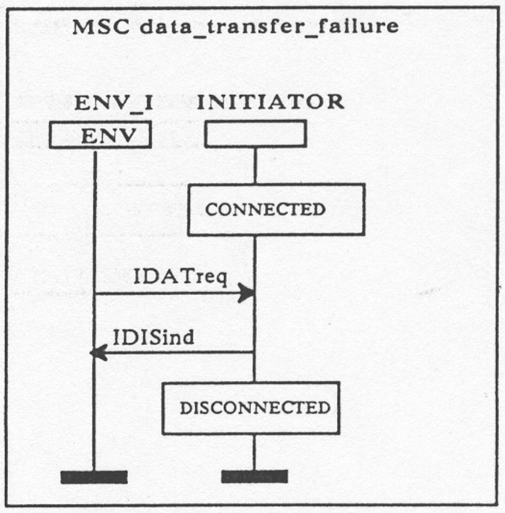


Figure 4.4d

Constructing the MSC–segments which we now obtain in addition to section 4.3, we first observe that MSC *request failure* (cf. fig. 4.4a) itself is a MSC–segment. MSC *response failure* (cf. fig. 4.4b) yields two additional segments which can be obtained from the

segments 'request–rejection' (cf. fig. 4.2b) and 'request–timeout' (cf. fig. 4.2c) by a corresponding insertion of 'response_failure' at condition 'WAIT' within the RESPONDER–instance. The MSC *data_tranfer_failure* (cf. fig. 4.4d) composed with a subsequent 'disconnection' (cf. fig. 4.1e) produces one segment.
The MSC *disconnection_failure* (cf. fig. 4.4c) leads to a lot of MSC–segments. In practice a separate treatment of such error exits, using further abstraction mechanisms, is recommendable.

5. Conclusion

For a comprehensive system specification by means of MSCs an extensive use of local conditions seems to be most appropriate. Global and semiglobal conditions are transparent shorthand notations which are more useful on a less detailed level, e.g. for the description of standard cases.
Including an MSC–asterisk–construct the described composition mechanisms make a far reaching comprehensive MSC–system specification feasible which also includes non-standard cases. Applying the idea of MSC–segments to the INRES–service in general they have proven to be very useful for a system overview and analysis. Whereas the MSC–pieces, originally defined, primarily provide a local system view, MSC–segments together with their transition graph give a global system view in a structured manner.

Literature:

[1] Belina F.: SDL Methodology Guidelines, CCITT experts meeting Rome, 1991

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

[3] Chong-Yi, Y.: Process Periods and System reconstruction, Lecture Notes in Computer Science 222, Advances in Petri Nets 1985, (G. Rozenberg ed.) Springer Verlag 1986

[4] Faergemand O., R. Reed R. (ed.): SDL '91 Evolving Methods, North-Holland 1991

[5] Grabowski J.: Statische und dynamische Analysen für SDL-Spezifikationen auf der Basis von Petri-Netzen und Sequence Charts, Diplomarbeit, 1990

[6] Grabowski J., Rudolph E.: Putting Extended Sequence Charts to Practice in "SDL'89 The language at work", Editors O. Faergemand & M. Marques, North Holland, 1989

[7] KDD contribution to WP X/3: The importance of State description in Sequence Charts, CCITT experts meeting Turin, 1990

[8] Nahm R.: Consistency Analysis of Message Sequence Charts and SDL-Systems, in [1]

[9] Reisig, W.:Petri Nets EATC Monographs on Theoretical Computer Sciences, Vol. 4 Springer Publ. Comp. 1985

[10] Rudolph E., Graubmann P., Grabowski J.: Towards an SDL-Design-Methodology Using Sequence Chart Segments, in [4]

[11] Rudolph E.: Message Sequence Chart, CCITT-Meeting Geneva, 1991

[12] Rudolph E.: Tutorial on Message Sequence Charts, 5 th SDL-Forum, Glasgow 1991