Putting Extended Sequence Charts to Practice

(Final Version)

Jens Grabowski, Dr. Ekkart Rudolph
Siemens AG, ZFE F 2 SOF 1
Otto-Hahn-Ring 6
D-8000 München 83
Fed. Rep. of Germany

Introduction

Within the SDL-User-Guidelines [1] only a short section is devoted to Sequence Charts (SC) as one of the auxiliary diagrams. However within an integrated tool set for the design of real time systems, particularly telecommunication systems, SC's may very well play an important role. The power of SC's shows up even more if the pure message form (standard-form) is extended by some additional concepts, mainly coming from SDL-process diagrams[4]. Eventually one trace in an SDL-system can be described completely by an extended SC.
The main importance of SC's, providing a graphically transparent description of system behaviour, lies in the stage of requirement definition. All later stages in system design have to be shown to be consistent with the SC's, set up for requirement definition. Thus SC's also provide a basis for test cases of the later implemented system. Beyond that, the stepwise refinement of SC's offers a realistic chance for a systematic computer aided and user guided construction of SDL-process diagrams from SC's.
A discussion of the role of SC's within the whole software lifecycle is presented in chapter 1. In chapter 2 several variants of SC's are introduced providing a stepwise refinement of the pure message form. Chapter 3 is devoted to the semantics of SC's particularly to the (time)ordering of SC-events. Finally in chapter 4 the realization of the presented methodology within an integrated set of tools for software design is elucidated.

1. Sequence Charts in System Engineering

1.1 Requirement Engineering

Static Aspects: System Partitioning by SDL-Block-Diagrams

The static or structural aspects of requirement definition show the functional partitioning of the system. In practice the partitioning is influenced also by the hardware components being involved. For the description of these static features SDL provides the hierarchical concept of block interaction diagrams.

Dynamic Aspects

The dynamic aspects of requirement definition are concerned with the (correct) system behaviour, i.e. the system reaction with respect to input signals. The required system behaviour can be described by traces. If the structural properties of the system, i.e. the partitioning into blocks and processes is clarified already, then (extended) SC's offer an appropriate graphical representation of such traces, which also presents the time (causal) ordering of events in an intuitive manner (chapter 3).
If the system partitioning still has to be derived from the dynamic behaviour, before the employment of SC's other trace representations have to be chosen, e.g. Mazurkiewicz traces [5].

1.2 Construction of SDL-Process Diagrams Based on a Stepwise Refinement of Sequence-Charts

The construction approach we have in mind is highly interactive and is based on sufficiently refined SC's: we assume SC's in the state- or state-input-form representing the 'correct' system behaviour. On this stage of design it seems to be reasonable to support the combination of traces (more exactly the projections of traces onto instance axes) to SDL-process diagrams (PD). The obtained PD's have to be supplemented by non-standard behaviour.

1.3 Consistency Check between SDL-Process Diagrams and Sequence Charts

A consistency check between PD's and SC's is necessary, since an SDL-system construction from SC's in practice is not fully automatic and the SDL-specification of a software system often is enhanced afterwards due to further requests by the customer.
Within such a consistency check essentially the symbol sequences along each instance axis of the SC have to be found in the corresponding PD's. That way also a refinement of existing SC's can be gained.

1.4 Sequence Chart-Generation from SDL-Process Diagrams

To remain in agreement with a possible enhancement of the SDL-system, for future consistency checks (1.3) and test case generation (1.6) the set of representative SC's has to be updated by generating new SC's from PD's.

1.5 Documentation (SC)

Although SDL-diagrams describe the dynamic behaviour of the complete specified system, for documentational purposes the representation is sufficiently transparent only with respect to single processes. Even within each process the SDL-description in general is very compact, such that other representations showing selected traces are desirable. For the documentation of process communication within a system configuration which may be complicated by several instances of the same process type, auxiliary diagrams are indispensable. Extended SC's with an optional degree of refinement combine both, a description of individual traces within one process (along one instance axis), and a description of process communication by message arrows.

1.6 Test Case Representation

SC's provide the basis for a black box test of the implemented system:
Signals coming from the environment provide input data.
Signals sent to the environment present the system reaction.

Within a test editor the test cases have to be enhanced by internal data transported by signals.
SC's in the state-input-form also admit the generation of non standard test cases (3.3).
Test cases for white box tests can be generated by the extraction of internal signals.
Other representations of test cases have been provided in [7].

2. Different Variants of Sequence Charts

2.0 System Representation by SDL-Process Diagrams and Sequence Charts

SDL-process diagrams and SC's can be looked at as two different kinds of system representation which are complementary in many respects. SDL provides a comprehensive description of system behaviour within one SDL-process (type), whereas the communication between several processes is represented in a fairly indirect manner (a more formal description of process communication can be gained by means of equivalent Petri Net representations [2]).
A complete system configuration is built out of one or several instances of process types.
SDL-process diagrams provide the starting point for code generation.
Contrary to that, SC's describe traces within a specific system configuration in form of signal flow diagrams. Thus SC's illustrate the process communication for special cases.
A refinement of SC's is possible within the state- and state-input-form. In realistic systems a large set of SC's would be necessary for a complete description of system behaviour (in practice one has to restrict oneself to a representative subset).
In the following, three variants of SC's are introduced which refer to different stages of system development. Apart from slight modifications the symbols of the standard-form correspond to the SC's described in the SDL-User-Guidelines [1].

2.1 Standard-Form

An SC in the standard-form (figure 2.1, 2.2) consists of process instances (obligatory), environments (optional), messages (obligatory), dialogs (optional), end (obligatory) - and interruption-symbols (optional). An SC may contain one or several environments, graphically distinguished from the process instance-symbol. A dialog-symbol is a shorthand notation for a message and a message answer between two process instances whereby on an early stage of design the direction is left open. Thus the dialog symbol 'connection' in figure 2.2 (a) may be a substitute either for message 'seizure' from Subscriber_A to Subscriber_B and 'answer' from Subscriber_B to Subscriber_A (figure 2.2 b) or the same messages with exchanged sender/receiver (figure 2.2 c).
The end-symbol marks the end of a process instance axis.
A process instance axis may be interrupted by another process instance within the same column, in order to get a compact and transparent graphical layout. The interruption symbol marks the interruption of an instance axis (in figure 2.1 the instance axis of Subsriber B is interrupted by Subscriber C).

2.2 State-Form

The state-form 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 symbol which on this stage of design should be refined by messages. Beyond that an SC in the state-form (figure 2.3) consists of STATE's (obligatory), START's (optional), SC-decision_results

(optional), TASK's (optional), SC-creates (optional), COMMENT's (optional), STOP's (optional). Contrary to [4] the inclusion of TASK's (2.2, 2.3) seems to us to be meaningful since a fully refined SC should describe a complete trace in an SDL-system for documentation etc.
Apart from the SC-decision_result and the SC-create-symbol the additional elements are identical with SDL-symbols. The SC-decision_result is a DECISION-symbol with one chosen result (figure 2.3). For simple cases (branchings do not contain message_outputs) the decision_result-symbol may be generalized to a construct showing more then one results. The graphical layout for the SC-create-symbol (figure 2.4), combining an CREATE-symbol and a message arrow, differs from [4].
Along each environment axis the syntax form is identical with the standard-form. Along each process instance axis we use the following syntax rules (a message-input is denoted by the arrowhead of a message, a message-output by the origin):
In the state-form each message-input has to be preceded by a STATE-symbol. First symbol on each instance axis has to be a STATE or a START. The START or the message-input is followed by a transition consisting of TASK's, COMMENT's, CREATE's, message-outputs, decision_results. Last symbol on a process instance axis may be either a STATE or STOP, the latter denoting a process stop.

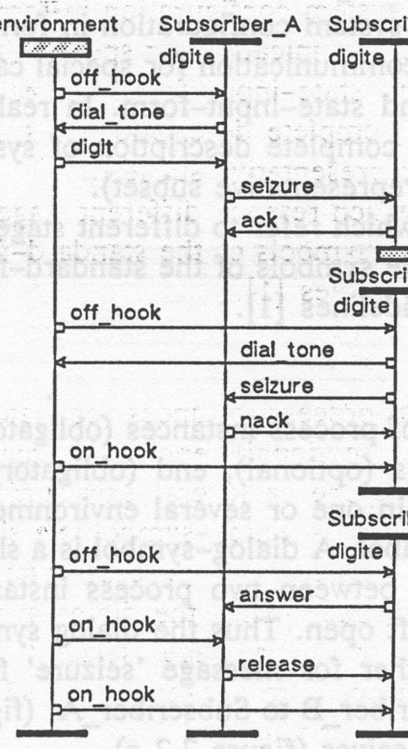

figure 2.1: Sequence Chart in the standard-form

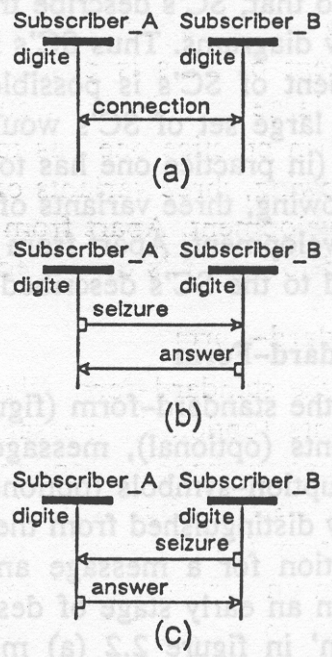

figure 2.2: Dialog-symbol (a) substituting (b) or (c)

2.3 State-Input-Form

Within the state-input-form the set of symbols used in the state-form is extended by an INPUT-symbol. As is explained and justified in detail in 3.3 an INPUT-symbol is used in addition to the message-symbol in order to distinguish between message reception (i.e. entry into the message queue of the process instance) and message consumption. In the SC-state-form a message-input is supposed to denote the message consumption (there is no separate indication of the message reception) whereas in the state-input-form a message-input denotes message reception and the INPUT-symbol denotes message consumption. Thus the state transition in the state-input-form is initiated by an INPUT-symbol which must be preceded by a corresponding message-input (figure 2.5). The other way round normally a message-input should have a corresponding INPUT-symbol as one of the successors. If this is not the case it would indicate the message consumption without transition i.e. a null transition.

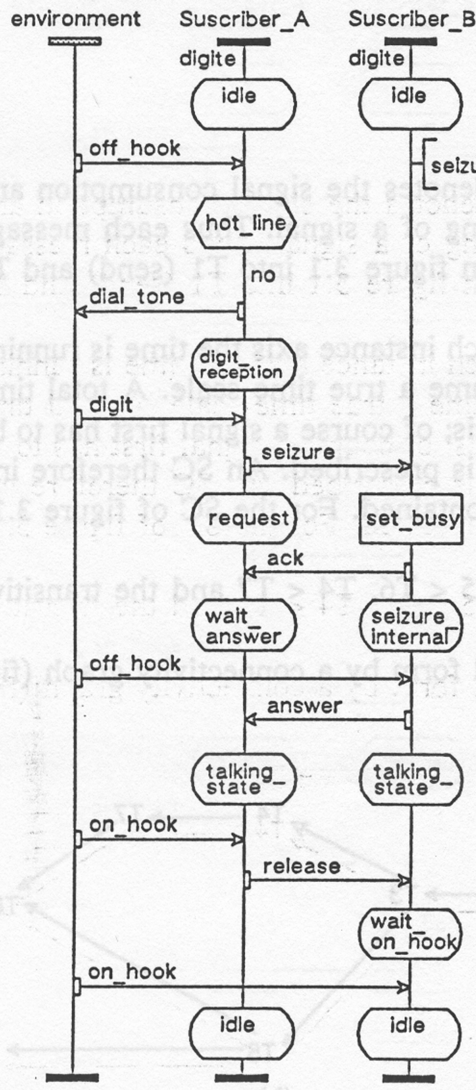

figure 2.3: Sequence Chart in state-form

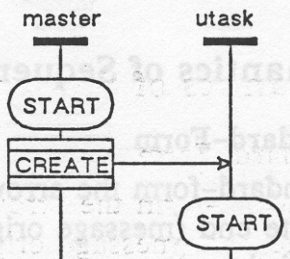

figure 2.4: SC-create-symbol

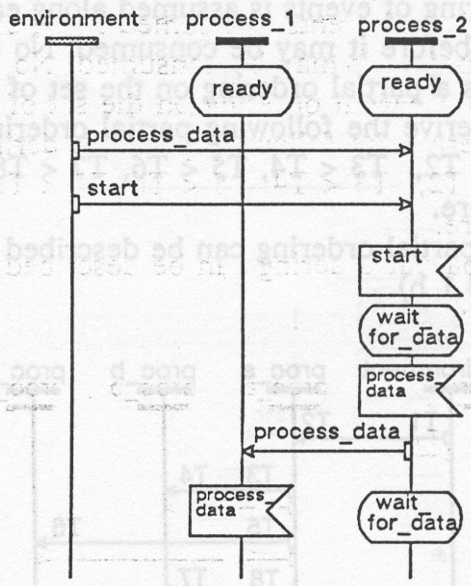

figure 2.5: Sequence Chart in state-input-form

2.4 Further Extensions

The described variants of SC's admit further extensions or modifications in different directions. On the one hand one may add further symbols and extend the syntax rules, on the other hand a modularisation concept may be included.
Firstly we agree with the authors of [4] not to use the following SDL-symbols within an SC: SAVE, PROCEDURE. The saving of a signal can be indicated within the state-input-form by assigning an ordering to the INPUT-symbols which differs from the ordering of corresponding message inputs (i.e. the ordering of message reception differs from the ordering of corresponding message consumption). The behaviour of PROCEDURE's should be shown in an expanded manner [4] since they may contain signals. ENABLING CONDITION and CONTINOUS SIGNAL might be included similarly to TASK's in 2.2, the latter leading to a transition without signal consumption.

Beyond the extension by SDL-symbols we intend to include constructs for the generalization of the time ordering on an instance axis and for modularisation [4] (macro brackets).
A further extension refers to the syntax rules. In practice the syntax of the state- or state-input-form may be too stringent. Therefore in a fourth form (mixed-form) also the use of STATE's is left optional.

3. Semantics of Sequence Charts

3.1 Standard-Form

In the standard-form the arrowhead of a message denotes the signal consumption and the opposite end (message origin) denotes the sending of a signal. Thus each message can be split into two events, e.g. the first message in figure 3.1 into T1 (send) and T2 (consumption).
No global time axis is assumed for one SC. Along each instance axis the time is running from the top to the bottom, however we do not assume a true time scale. A total time ordering of events is assumed along each instance axis; of course a signal first has to be sent before it may be consumed. No other ordering is prescribed. An SC therefore imposes a partial ordering on the set of events being contained. For the SC of figure 3.1a we derive the following partial ordering:
T1 < T2, T3 < T4, T5 < T6, T7 < T8, T2 < T3 < T5 < T6, T4 < T7 and the transitive closure.
The partial ordering can be described in the minimal form by a connectivity graph (figure 3.1 b).

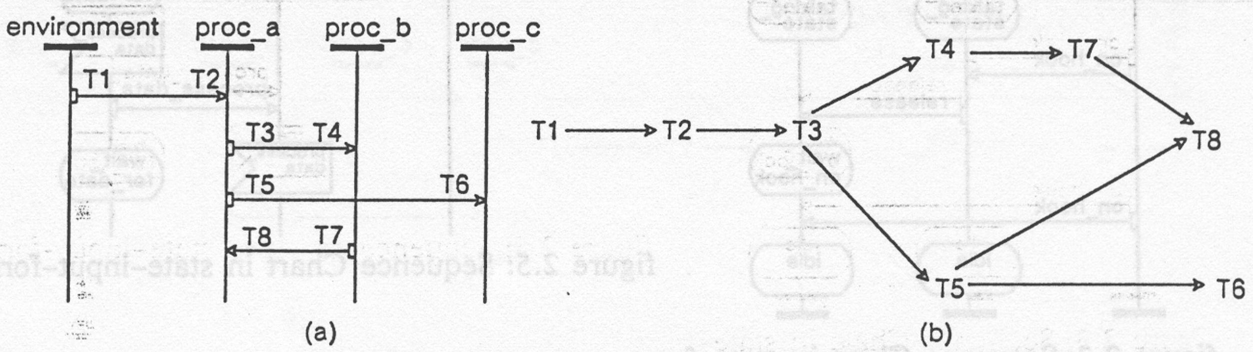

figure 3.1: Partial event ordering of a Sequence Chart in the standard form (Sequence Chart and corresponding connectivity graph)

3.2 State-Form

Within the state-form the message semantics of the standard-form is preserved. The partial ordering is analogous to the standard-form: total ordering of SC-events together with SDL-symbols along each axis, ordering between events on different axes merely via messages.

3.3 State-Input-Form

Proceeding to the state-input-form the message semantics is enhanced. A message arrowhead now denotes *signal reception* (the message origin again indicates the signal sending).The INPUT-symbol denotes the *message consumption*. In this way it is possible to

describe situations where the order of signal consumption is inverse to the order of signal reception which may be caused by signal saving, ENABLING CONDITION's or priority signals in services (a different illustration of priority signals is proposed in [1]). In addition non-standard situations of signal transfer may be described for test case generation.
Signal sending, signal reception, signal consumption and the following state transition are totally ordered. The reception of signals takes place in the order prescribed on each axis. Thus in figure 2.5 the signal 'process data' is received before 'start'. However the signal 'start' is consumed before the signal 'process data', which is possible if signal 'process data' for instance is saved in the corresponding SDL-process diagram.
In case where signals are consumed immediately after signal reception, the INPUT-symbol can be omitted and one arrives back at the state-form.

3.4 Extensions

Macro constructs (macro brackets) can be looked at as a contraction of process instances which are not yet specified on this stage of design. Macro constructs admit the description of the external behaviour. For a macro bracket at most a partial ordering of events may be prescribed.
Independence brackets may be used for a comprehensive representation of situations where the ordering of certain events along one instance axis is irrelevant.

4. Integrated Set of Tools

Graphic editors for SDL-block diagrams, SDL-process diagrams and Sequence Charts are part of an integrated set of tools for SDL-system design which are used for the specification of switching systems (HICOM), automation technology, and reverse engineering [3],[6]. Corresponding tool components for static analysis (syntax check, static consistency checks) are available.
In particular each of the variants of extended Sequence Charts (chapter 2) may be checked syntactically.This includes an analysis of signal exchange and signal consumption. Special features of the signal flow can be visualized, especially the signal processing due to one selected message-input. Beyond that, the causal structure of SC-events (successors, predecessors, concurrent events) can be analyzed and visualized. In that way also inconsistencies, e.g. acausalities leading to deadlocks, are detected.
The tool set includes code generators for the generation of C- and CHILL-programs from SDL-specifications.
Currently the development of tools for PD / SC-consistency check (1.3) and SC-generation from PD's (1.4) is in progress. In addition the Sequence Chart editor is employed for the visualization of program flow in the implemented system.

5. Literature

[1] CCITT Recommendation Z.100–Annex D
SDL User Guidelines, 1988

[2] Graubmann, P.; Rudolph, E.: A method and a Tool for the Validation of SDL–Diagrams, Second SDL Users and Implementers Forum, Helsinki, 1985

[3] Koßmann, H.: An Integrated Set of Tools for Software Design, SDL '87 State of the Art and Future Trends, (R. Saracco, P. A. J. Tilanus eds.), North Holland, Amsterdam, 1987

[4] Tilanus, P.A.J.; Dijkerman, E.: On the Combination of SDL and Message Sequence Charts, CCITT SDL Newsletter 12, 1988

[5] Graubmann, P.: The construction of EN systems from a given trace behaviour, Advances in Petri Nets 1988,(G. Rozenberg, ed.), Springer–Verlag, 1988

[6] Tempel, H.G.: A Set of Tools supporting the Software Design Based on SDL, 1st European Software Engineering Conference, Strasbourg 1987, Springer Verlag, 1987

[7] Hogrefe, D.: Automatic generation of test cases from SDL specifications, CCITT SDL Newsletter 12, 1988