SDL—CCITT Specification and Description Language

ANDERS ROCKSTRÖM AND ROBERTO SARACCO

Abstract—**The CCITT Specification and Description Language (SDL) became a CCITT recommendation in 1976, and a refined version was published by CCITT in 1980. This paper presents the background and history of SDL, and its relation to the other languages recommended by CCITT.**

SDL is mainly a language to specify and describe the logic of functional processes in a fashion independent of implementation techniques, and is based on a finite-state machine approach. The concepts and representation forms of SDL are briefly explained. A survey is given of today's applications and tools, and the expected future developments are discussed.

I. HISTORY

THE CCITT Specification and Description Language (SDL) has so far been under study within CCITT, Study Group XI, for three study periods (i.e., 3 × 4 = 12 years), and is presently being further refined upon and improved in the current study period. CCITT SG XI is a study group primarily dealing with producing recommendations in the area of telephony switching and signaling—therefore, it might need some explanation why the group undertook the study of SDL and the other CCITT languages.

Why SDL

The study was initiated in the late 1960's when the experience of the first stored program control (SPC) applications in telephony was first felt. As SPC was a new technology there were a number of problems attached to the first exchanges. These problems were often identified as "software problems." Whether this was a correct identification may, of course, be discussed at length. This led, however, to a tendency of claiming that the solution to the problems should be to use "high-level languages" when programming the exchanges, in contrast to the "low-level" languages used at that time. Particularly for SPC telephony applications, a telephony-oriented language was urged. A proposal that this should be studied within CCITT was first raised at the CCITT Plenary Assembly in Mar del Plata in 1968. The question raised was regarded as a switching problem, and the question was assigned to SG XI.

However, the results of the first discussions on this question were that the need of two different languages was recognized: one high-level and telephony-oriented programming language, and one language for specification and description of functions. This latter language, SDL, was not to be a programming language but a language able to describe functions in a fashion independent of implementation techniques.

Manuscript received May 14, 1981; revised December 23, 1981. This paper was presented in part at the National Telecommunications Conference, New Orleans, LA, December 1981.

A. Rockström is with the Central Administration of Swedish Telecommunications, S-123 86 Farsta, Sweden.

R. Saracco is with CSELT (Centro Studi e Laboratori Telecomunicazioni) S.p.A., 10148 Torino, Italy.

Progress

The first period of study, 1968-1972, was mainly devoted to discussions about what should be studied. The main result of this study was that the "CCITT language family" was extended to also contain a "Man-Machine Language" (MML).

In the next Study Period the basic SDL was outlined, the language becoming not exactly defined but the general principles were agreed upon. The result of this was the Recommendations Z101-Z104 approved at the CCITT Plenary of 1976.

During the following Study Period, 1977-1980, these recommendations were refined upon. A number of semantical problems were resolved, and the language became extensively used within CCITT and by manufacturers and administrations. The new CCITT Recommendations on SDL, Z101-Z104, were approved in the Plenary of 1980.

The present work on SDL is presented in Section V of this paper.

Relations to Other CCITT Languages

As briefly mentioned above, SDL has been developed in parallel with the other two CCITT languages; the high-level programming language CHILL and the man-machine language MML. During this development the desirability of a harmonized language family has been considered and recognized, i.e., the use of common concepts and terminology. This can be illustrated with the following quotation from the consideratas to the study questions of 1972:

> ". . . that there is an interrelation between the requirements of a command language and those of a high-level programming and methods of functional specification to be developed, and this relationship must be considered in their development. The establishment of a common standard terminology is of greatest importance. . . ."

This correlation between the languages has been obtained; for example, both CHILL and SDL contain a number of common concepts: *process*, *signal*, etc.

The relations between the languages and the actual system specified and described can be illustrated in the manner shown in Fig. 1.

Very briefly, Fig. 1 can be explained as follows. Both the specification and the description of a system can be made using SDL, and it is possible to make a "functional" comparison between these two descriptions. While SDL describes the functions of a system in a way independent of the implementation technique, CHILL can describe the same functions adding details depending on the implementation in software. The

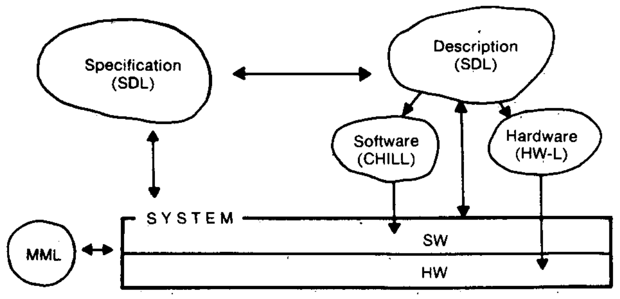


Fig. 1. The CCITT language family.

interaction between the implemented system and the operators of the system is performed using MML.

All these representations are found necessary and useful, and it is also found useful if the different representations do not use different concepts or terms to represent identical items. These relations and the scenario of use is further explained in Section IV of this paper.

II. BASIC SDL

As mentioned, one objective for the development of SDL was that the language should be able to describe telephony functions in a telephony-oriented fashion, that is easily understandable by telephony people. In order to do this a number of general SDL concepts have been defined. In this section these general concepts are explained, and in the next section the representation forms are presented.

Telephony Functions

SDL aims mainly at describing the logic of processes in telephony, within exchanges, between exchanges, and the general interworking with the environment. This means that SDL also will cover a wide range of general real-time applications outside the telephony area as well.

The characteristics of telephony functions are as follows.

1) The great number of concurrent performances of both equal and different processes. For example:

- all the calls progressing concurrently in an exchange
- the different signaling processes a call can make use of when interworking with another exchange.

2) The progress, for each of these performances, is triggered by events external to the process itself. For example:

- the progress of a call may be initiated by a subscriber lifting off the receiver (off-hook).

3) These triggering events will normally also carry some information, besides the information that the triggering conveys, to the process. For example:

- when a subscriber triggers the call process with a dialed number.

All the characteristics above are independent of the technique the process happens to be implemented in. The same functional processes are performed by operators and electromechanical systems as well as by SPC systems. There are, of course, numerous ways of describing these types of processes. A commonly used approach is to regard the processes as communicating finite-state machines; this is the approach SDL is based upon. This approach is further elaborated and a brief explanation of the concepts used in SDL is found below.

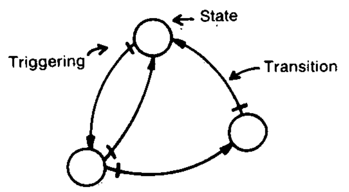


Fig. 2. A finite-state machine represented as a directed closed graph.

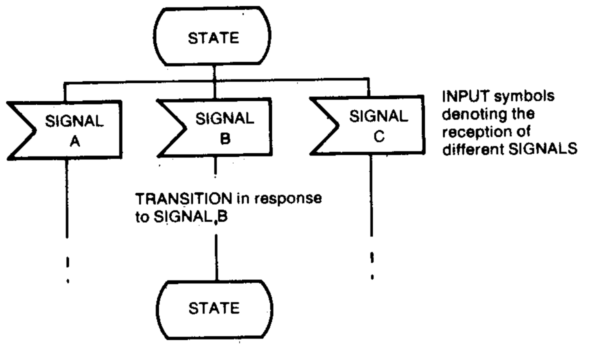

STATE
SIGNAL A
SIGNAL B
SIGNAL C
INPUT symbols denoting the reception of different SIGNALS
TRANSITION in response to SIGNAL B
STATE

Fig. 3. Graphical SDL representation.

Dynamic Behavior

The object in SDL that is actually performing the logical function is called a *process*. Each process is regarded as a finite-state machine that can be represented by a closed directed graph as in Fig. 2.

The process is further assigned the property that every state can be reached from any other state by a suitable series of transitions. This means that "dead states" are considered semantically incorrect in that there is no starting or terminating state. The communication between processes is defined to be performed only via sending and receiving signals. The concept *signal* servers both as a primitive for synchronization (triggering) and as a vehicle for conveying information between processes.

An SDL process is defined as an object that is in either a state awaiting an *input* (the reception of a *signal*) or is in a transition. A *state* is defined as a condition in which the actions of a process are suspended awaiting an input. A *transition* is defined to be a sequence of actions which occurs when a process changes from one state to another in response to an input.

Fig. 3 is an example of the relation between these concepts, using the graphical version of SDL syntax. For each state it is defined which inputs are allowed to cause transitions. In a transition there are three types of actions allowed:

- *output* which is an action generating a signal which in turn will act as an *input* elsewhere;
- *decision* which is an action asking a question to which the answer can be obtained at that instant and which chooses one of several paths to continue the transition;
- *task* which is an action within a transition which is neither a decision nor an output.

In addition to this there is one more concept defined within a process, that is, the concept of *save*. This concept has been defined in order to clarify the semantics of interprocess communication. In order to explain the concept, this semantic has to be expanded somewhat.

When a signal is sent, via an output or from the environment, it is always addressed to one particular process. When the signal reaches this process, it will be retained for later reception. The process is able to receive signals only when in a state. For each state it is defined which signals are allowed to be received, and which transitions they should cause if received. When the process is in a state, then one of the signals waiting to be received will be made available for the process for reception. If there are no signals waiting the process will just be suspended until a signal arrives. The selection of a signal is based on the order of arrival. If the selected signal is allowed to cause a transition, it will do so. If the signal is not allowed to do so it will just be discarded, and the next signal will be made available.

The *save* concept is used to "save" signals from being discarded, the semantics being that a signal referred to in a save-construct will not be made available but will stay waiting to be received until a new state is entered. The discarding is regarded as an implied transition resulting in the same state as before.

Structural Concepts

SDL, as defined in the current recommendation, deals mainly with processes and their dynamic behavior and representation. Concerning the structure of a system, today's recommendation merely states that a functional specification (FS)/description (FD) can be partitioned into functional block specifications (FBS)/descriptions (FBD). A functional block is defined to contain one or more processes. See Fig. 4.

How this partitioning is done and how it is represented is not yet defined. This is one of the main topics for the present Study Period; see further Section V.

III. REPRESENTATION FORMS

As mentioned, there are several ways of representing an SDL process. See Fig. 5.

SDL as such is defined on a conceptual level, and the different manners of representation of those concepts all conform to the defined semantics. The reasons for having several syntaxes are demands from different usages and different tools used. Today the graphical representation of SDL is the best defined. Program-like versions of representation are under development. Other versions adapted to special usage might be developed later on. The graphical representation of SDL has been enriched also with a "pictorial option." These three ways of representation are briefly presented below.

Graphical Representation

An SDL graph consists of a number of graphical symbols, each representing one of the subconcepts of a process, connected by directed flowlines. The symbols used are shown in Fig. 6.

The state symbol should be completed with a text indicating the unique state name of the state represented. If several state symbols containing the same state name are found in one SDL graph then, by definition, they all represent the same state. This multiple appearance of a state in a diagram is used to simplify the graph and to make the documentation more easily understood. In the symbols representing inputs, outputs, and saves, a signal name should be contained to indicate which signal the symbols refer to. In the same manner the appropriate text is added to the task and decision symbols.

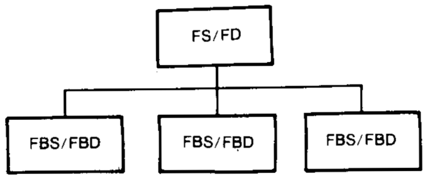


Fig. 4. Functional blocks.

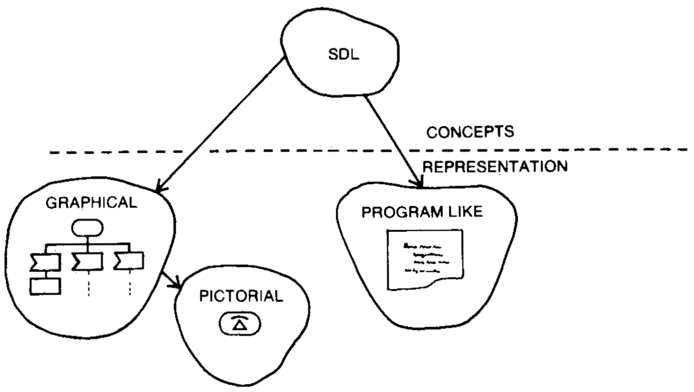


Fig. 5. One language—several representation forms.

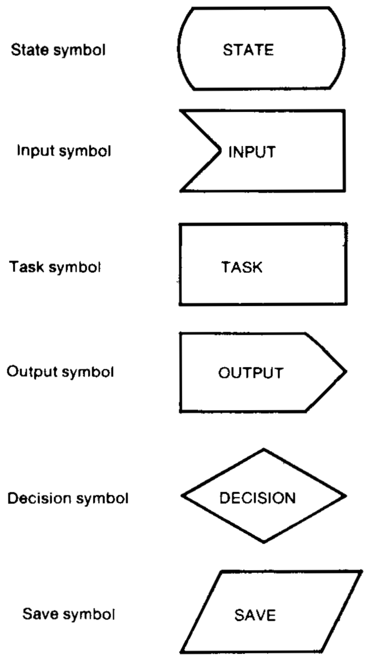


Fig. 6. SDL graphical symbols.

When connecting these symbols into a diagram, the following sequencing rules should be followed.

- A state symbol may only be followed by input symbols or by input and save symbols.
- Each input and save symbol follows one and only one symbol, which must be a state symbol.
- Each input symbol is followed by one and only one symbol which may be any symbol but an input or save symbol.

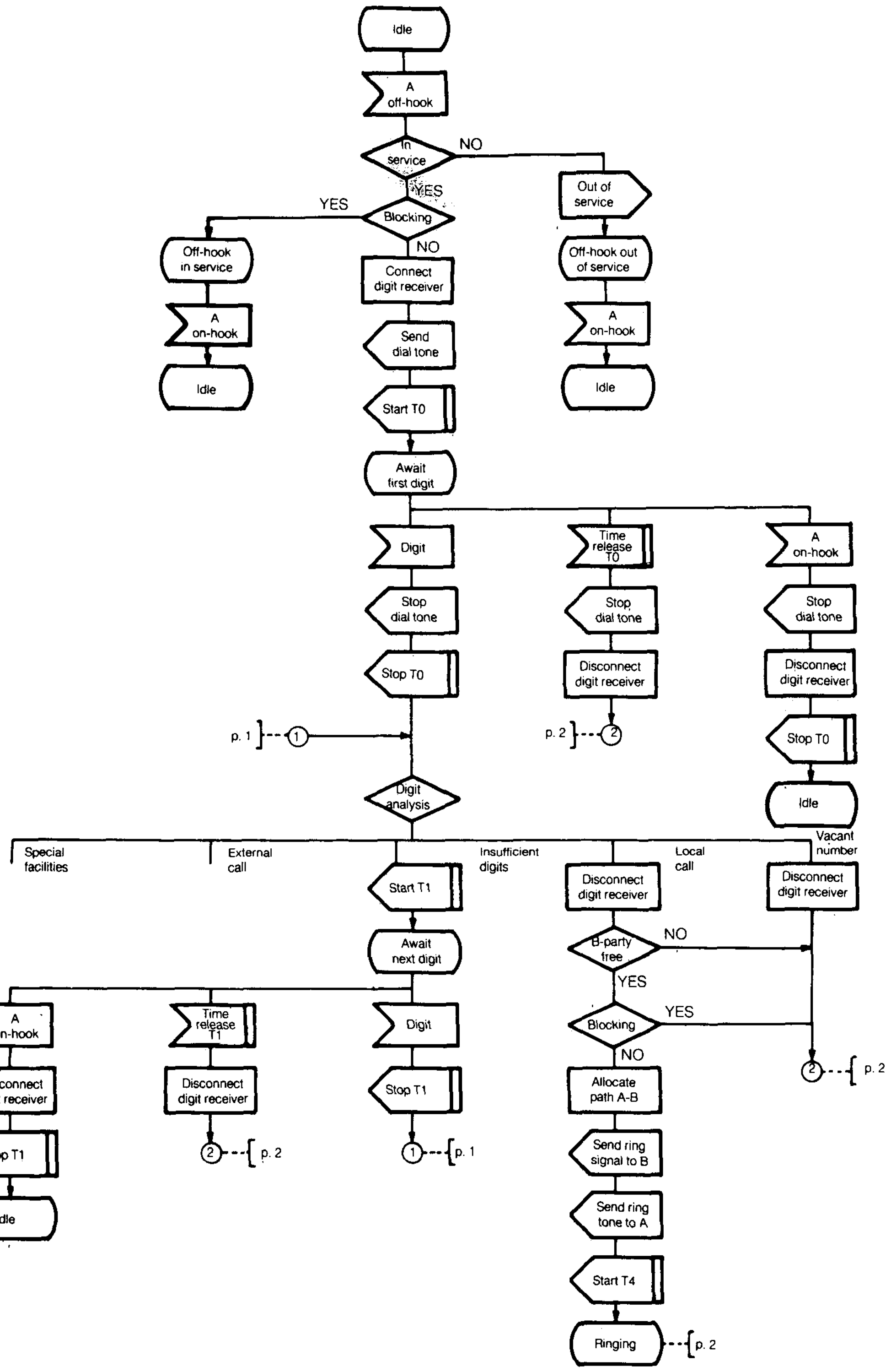


Fig. 7. Example of an SDL diagram, part of a call-handling process.

• Each task or output symbol is followed by one and only one symbol, which may be any symbol but an input or a save symbol.

• A decision symbol must be followed by two or more symbols, which may not be input or save symbols.

• A save symbol may not be followed by any symbol.

Fig. 7 is an example of a graphical representation of an SDL process. The figure shows a part of a call handling process and is taken from an example in the current recommendation.

Pictorial Option

A graphical representation of SDL with the pictorial option makes use of the same set of basic symbols as above; what is added is a standardized set of pictorial elements to be included in the state symbols to describe the state indicated. The pictorial elements standardized up until now are strictly telephony-oriented.

The basic idea behind this representation method is that

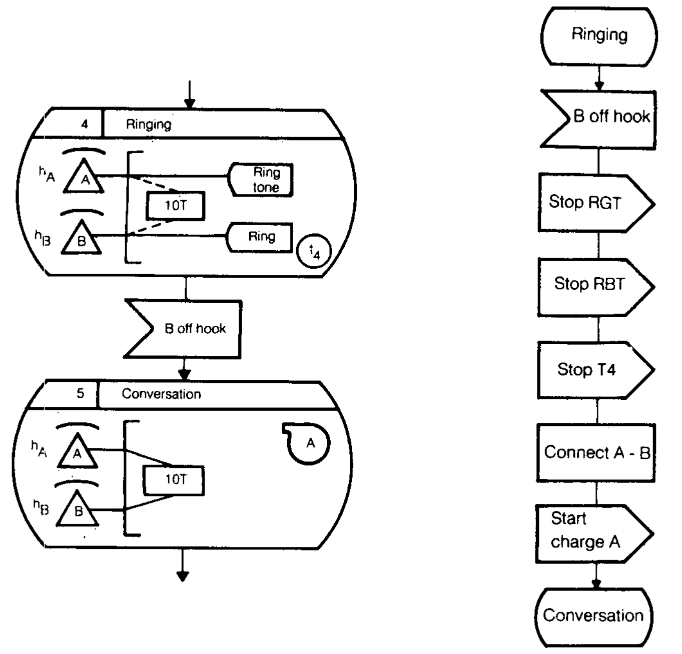


Fig. 8. Equivalent transitions, represented with and without pictorial elements.

the described process is regarded as a finite-state machine, progressing by changing states. Each state represents a specific status of the process, and the transitions describe changes from this status to another status. Thus, if the states are described accurately enough, the transitions can be implied, i.e., the actions of the implied transition should be such that they change the status of the process to fit the new state.

This is exemplifed in Fig. 8, where all the actions shown in the right-hand graph are implied by the difference of the state pictures in the left-hand graph. In this example all the actions are implied; this makes it difficult to study the interworking with other processes, as no outputs appear. However, the diagrams can be made more explicit using a combined form as shown in Fig. 9.

Program-Like Representation

The graphical representation as explained above is considered to be the version most easily understandable and usable for humans. However, there is also a need for a "program-like" representation form, especially for computer storage and manipulations.

During the last study period of CCITT a program-like version of SDL was discussed and almost standardized—there was not enough time to finalize the work. The version discussed for the program-like form is based on a construct very much like the corresponding constructs for process and signal handling in CHILL. However, the key words are adapted to the names commonly used in SDL environments.

Fig. 10 illustrates a part of a call handling process, represented both in a graphical version and in a program-like version.

As mentioned before, there exists a close relationship between CHILL and SDL, and the two languages are harmonized conceptually to each other. This means that it is also possible to represent an SDL process with CHILL code, and it is also under discussion to define a subset of CHILL as one program-like representation of SDL.

IV. APPLICATIONS AND TOOLS

SDL can be used for a wide range of applications. Its definition has been particularly tailored for the telephony field. Fig. 11 (taken from the SDL User's Guidelines) shows a range of possible uses of SDL in the context of purchase and supply of telecommunications switching systems.

In Fig. 11 the rectangles illustrate physical functional groups, whose precise names may vary from organization to organization, but whose activities would be typical of many administrations and manufacturers. Each of the flowlines respresents a set of documents passing from one functional group to another. SDL can be used as a part of each of these sets of documents.

Applications Within CCITT

As a first example let us take the flowline A3 of Fig. 11. The CCITT has already used SDL in specifying the interworking of international signaling systems (Rec. Q.601-Q.685), in the specification of maritime signaling systems (Rec. Q.60, Q.61 and Q.62), and in the specification of the new number 7 signaling system for digital integrated services network (Rec. Q.7). It is expected that all new CCITT telephone signaling systems will be specified using SDL.

Presently Study Group VII of CCITT is studying the application of SDL to data switching protocols.

Between Administrations and Manufacturers

Let us consider as a second example, the flowline A1. Several national administrations have already been using SDL for several years in producing in-house specifications for new facility requirements, such as new facilities for telephone subscribers. These facility specifications are often network independent in the sense that they could be implemented by adding hardware or software technology in any of several network locations, including at the subscriber's telephone set, at the local exchange, or at centralized regional processing centers, or in all of them. These specifications are usually implementation independent in the sense that they do not predetermine the manufacturer's choice of the hardware or software technology or the overall system design architecture. Different experiences have shown that SDL effectively describes the required facilities without prejudicing the manufacturer's means of implementing these facilities.

A facility specification is often converted into a network-dependent but implementation-independent systems specification, as shown by flowline A5, before being issued to a manufacturer. A large number of national administrations have already used SDL in systems specifications, and are planning to use it more extensively in the future.

After analysis of the specifications received, the manufacturer will propose an implementation description which satisfies the requirements contained in those specifications. Obviously, the implementation proposed is not always in a one-to-one relation with the specification; sometimes it presents

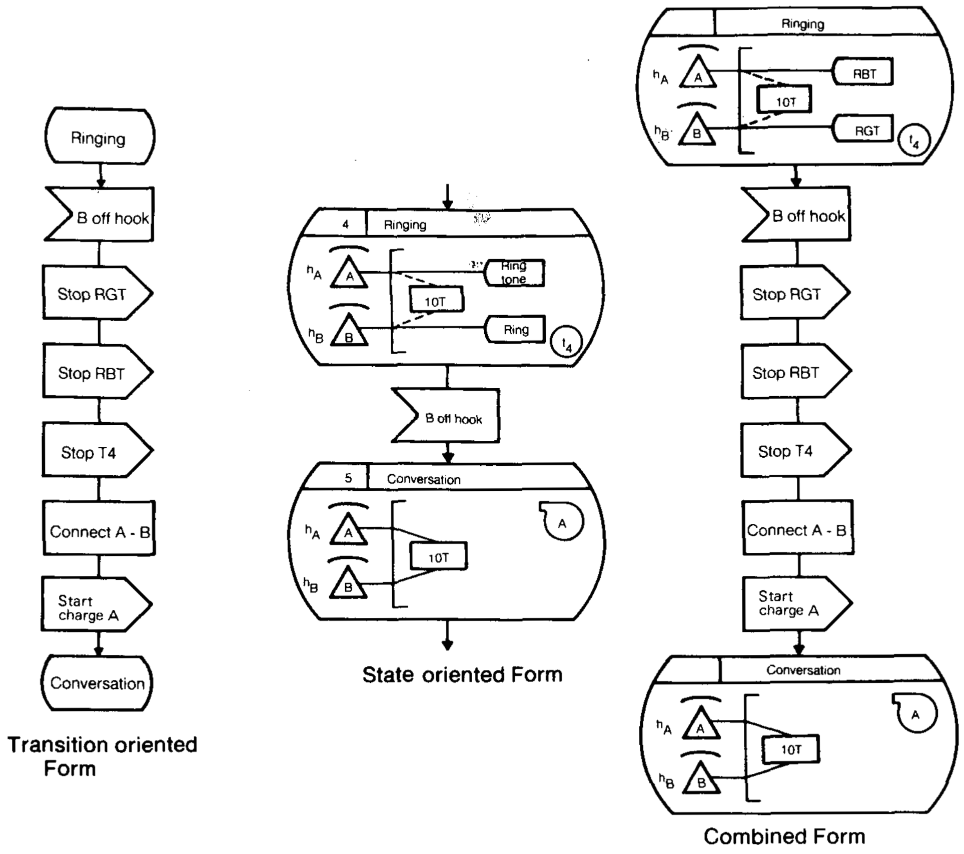
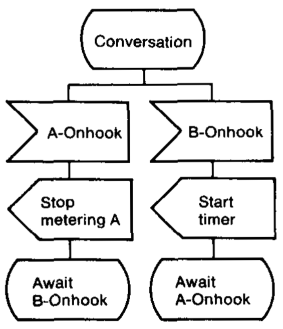


Fig. 9.



```
Block Call-handling;
Process Local-call;

  State Conversation;
    Input A-onhook;
      Output Stop-metering A;
      Nextstate Await-B-onhook;
    Input B-onhook;
      Output Start-timer;
      Nextstate Await-A-onhook;
  Endstate;

Endprocess;
Endblock;
```

Fig. 10. Example of the program-like representation of SDL.

some alternative possibilities by which the requirements may be met. SDL serves as an excellent documentation technique for highlighting the proposed amendments to the systems specifications, without entering unnecessary details concerning the technology to be used in its implementation. Furthermore, an SDL specification finally approved by both the customer and the supplier may be part of the test specification to be used for systems acceptance testing.

SDL has also already been used by several telecommunications manufacturers as a documentation tool for in-house design and has been found to have considerable advantages as a documentation interface between different subsystem design teams. Flowline A7 to A10 in Fig. 11 shows this use of SDL. It may be noted that the documentation required for systems simulation (A9) and system testing (A10) derives directly from earlier design documentation (A7, A8) and that the final detailed system documentation (A11, A12) can be related to the systems specification.

To sum up, SDL is intended to be used for the specification of the functional behavior required and for description of the actual behavior of the system implemented. However, it has also been used during the design phase by many manufacturers for information interchange between different working teams and as a product documentation during the various steps of the product implementation.

The use of SDL in the design phase affords many advantages because of the unique documentation for the whole system evolution process (decreasing training problems and the possibility of misunderstanding deriving from the application of different documentation languages) and because some documents produced during the design phase can be used as they are for the product testing and documentation delivered to the customer (less cost in documentation production and avoidance of translation consistency problems, ensuring a product documentation completely corresponding to the product delivered)—see Fig. 12 below.

In addition, SDL can be used during the system life to help the training of personnel, the system operation, and the evaluation of possible system upgrading.

The SDL in itself may be used in a wider field of application apart from the telephony field. It can be used and has already been used to document operational procedures such as man-machine interaction procedures, or to describe manual actions to be performed by operators of SPC systems. The part which may need extensions as a result of a wider field of appli-

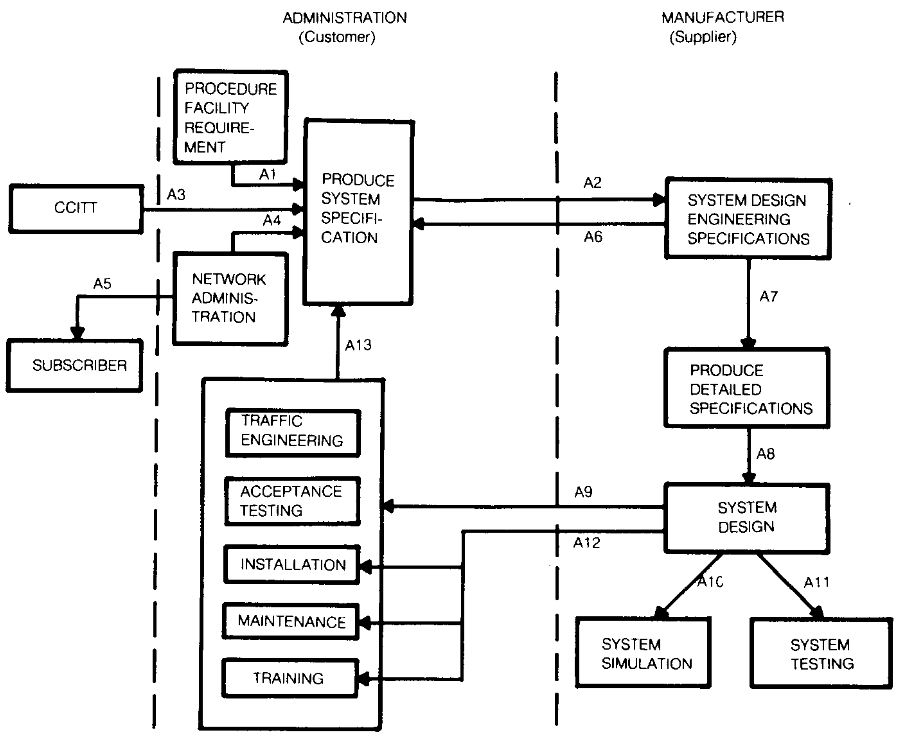


Fig. 11. One assumed scenario for the use of SDL.

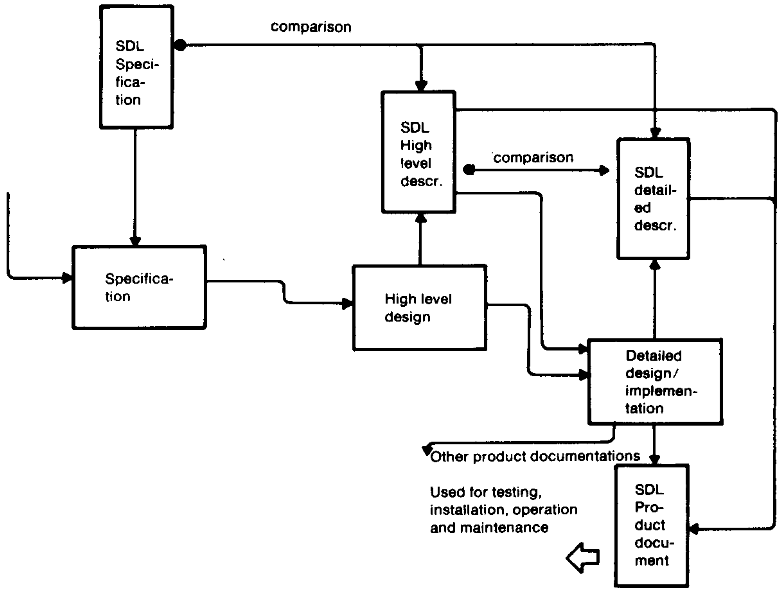


Fig. 12. The use of SDL in the design phase of a system.

cation is the pictorial representation option. Presently this part contains only symbols defined purposely for telephony applications and most probably other symbols are required to represent other concepts in a different environment.

Tools

The outlined possible applications of SDL call for the development of tools to aid the creation of SDL documents, checking, handling, and updating. The presence of these tools may increase the advantages deriving from its use, resulting in a wider application of SDL. Conversely, a wider use of SDL will lead to the development of new tools in a synergic fashion.

The SDL has been defined and it is presently extended in such a way that formal consistency and completeness checks are possible. The formal (mathematical) description of SDL has as its main purpose the intent of providing an effective description of SDL concepts and rules to aid tool developers in the implementation of SDL document syntax and semantics analyzers. Present extensions to SDL on structural representation of systems architectures take into account the aspects of automatic checks among system representations on different levels of detail.

In addition to the primary SDL documents, the SDL documentation comprises other auxiliary documents to supplement the primary ones, increasing the intelligibility and easing the approach to systems documentation. These auxiliary documents may be automatically generated, starting from the primary documents through ad hoc developed tools.

Tools may also allow the generation of a form representation [graphical (GR) or program-like (PR)] starting from the equivalent one (PR or GR). By these means it is possible to store SDL documentation using as example the more compact and computer-suitable PR form, having as output the GR form easily readable for users.

SDL documents may be used to simulate the behavior of the represented function or system by means of simulators using SDL representation as the simulation base.

Some parts of the SDL primary documents of a given actual system may be automatically generated starting from the implementation itself. This may be the case for software parts where ad hoc developed tools can be used to extract the SDL functional representation of the behavior of a function implemented in software directly and automatically from the program statements. Following these lines, some tools are presently under study to extract SDL representation from programs written in CHILL. The other way around, namely, generating CHILL code implementing the function described in SDL, is also under study in a first delimited context where the tools may help in generating CHILL code, but interaction with a software engineer is required to fill those gaps necessary to generate a program code from the SDL function representation.

Fig. 13 presents the SDL documents and their relation from the tool development point of view. It is foreseen that a wide variety of tools will be developed in the future to integrate the SDL representation method with existing design aids in different manufacturers.

Tools developed based on SDL systems representation to aid the training, operation, and maintenance offer a great deal in reducing the immense hidden costs of these activities and may reduce the long learning periods before personnel may become fully efficient after the cutover of new SPC systems.

V. PRESENT AND FUTURE WORK

As outlined in the historical record, the present SDL recommendation covers the functional representation and the function cooperation by means of signal interchange. In order to cover all the aspects related to the application of SDL for describing switching systems according to different levels of detail, CCITT is presently working on extending the semantics of SDL to cover common procedures, data, and other representations of information interchanges, and other extensions to include structural concepts such as partitioning and levels of specification and description. In addition, work is under progress to refine thє PR form of SDL and, in study, consideration is given to correlation with the CHILL language. The correlation with CHILL to be studied is considered to be what guidelines are required to permit the automatic generation of CHILL code from SDL and SDL from CHILL code.

In parallel to these areas of study, consideration is given to evident facilities required in SDL in order to facilitate the development of tools for automatic consistency checking,

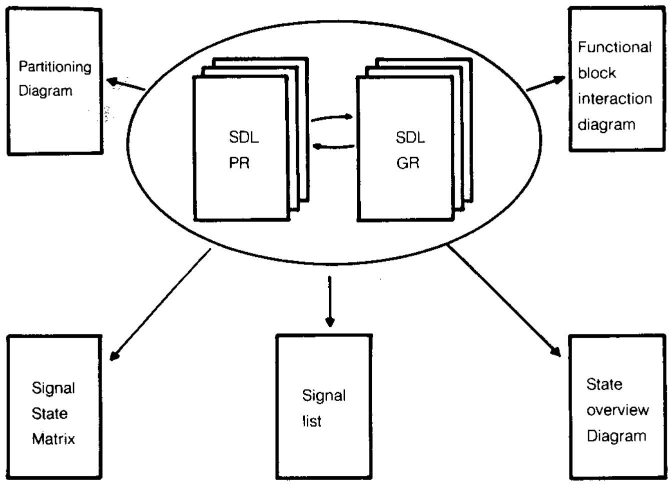


Fig. 13. Example of auxiliary documents that can be generated from the SDL representations.

automatic logic comparison of SDL diagrams, and automatic generation of other support information. Guidelines to help users in the application of SDL are under preparation; as soon as new concepts are introduced, guidelines on how to use those concepts will be prepared and made available.

Different priorities have been assigned the above-mentioned areas of work. In particular, the highest priority has been assigned to the extensions of structural concepts to cater to the systems partitioning and to the formal (mathematical) description of SDL.

According to the work program the formal description of SDL is expected to be defined by Spring 1982. By Spring 1983 it is expected to have a draft recommendation for the extensions proposed for the SDL-PR form, and also to have a complete set of user's guidelines. The final recommendations are expected by Autumn 1983.

The mentioned intention of obtaining close correlation means to extract SDL from CHILL code and vice versa will require more time.

The work presently undertaken at CCITT on the SDL benefits from a constant interaction among the groups studying the extensions to SDL and the users who are currently using SDL in specification and description, as well as with the tool implementers. An SDL Newsletter is published with the main purpose of informing SDL users about SDL support development and availability, user comments, areas of application, derived problems, and solutions accepted. It is therefore expected that the constant evolution of the SDL towards the solution of user problems will be smooth, maintaining upward compatibility so that all the documentation already using SDL can remain valid, but the subsequent availability of new concepts will provide a more powerful and more widely applicable language, increasing the derived benefits.

REFERENCES

[1] Recommendations Z 101–104, *CCITT Orange Book*, Int. Telecommun. Union, Geneva, Switzerland, 1977.
[2] Recommendations Z 101–104, *CCITT Yellow Book*, Int. Telecommun. Union, Geneva, Switzerland, 1982.

Anders Rockström was born in Sweden in 1946. He graduated from Technical College, Växjö, Sweden, in 1966.

In 1968 he joined the Swedish Telecommunications Administration, Farsta, where he has since been involved in the design of SPC telephone systems. Currently he is responsible for the software production of Swedish AXE 10 applications. In 1976 he was first involved in language development at CCITT (the International Telephone and Telegraph Consultative Committee), and in 1981 he was appointed Chairman for the sub-working-party developing SDL.

Roberto Saracco was born in Turin, Italy, in 1953. He graduated in information science in 1971 and received the Ph.D. degree in mathematics from the University of Turin, Turin, in 1975.

In 1971 he joined CSELT (Centro Studi e Laboratori Telecommunicazioni), Turin, where he has been involved in the design of software for SPC telephone exchanges.

Since 1975 Dr. Saracco has been involved in the design of software languages at CCITT (the International Telephone and Telegraph Consultative Committee). He is currently responsible for a research study on CAD for software projects.

CHILL—The Standard Language for Programming SPC Systems

KRISTEN REKDAL

Abstract—**SPC telecommunication systems have proved to contain some of the largest and most complex pieces of software ever constructed. To master this complexity requires the use of powerful tools. The CCITT HIgh-Level programming Language—CHILL—is one such tool that will contribute to improving the quality of SPC systems in the 1980's and beyond. At the CCITT Plenary Assembly in November 1980, CHILL was approved as CCITT Recommendation Z.200.**

The CHILL language is important because its widespread use will imply significant savings by both telecommunications manufacturers and administrations. The language is the outcome of a large effort which is not easily repeated. CHILL has reached the state of being a recommendation from a world standardization body and is already in practical use with several major manufacturers of telecommunication systems.

This paper surveys the history and objectives of CHILL and gives a short introduction to the language.

THE CHILL LANGUAGE DEVELOPMENT

Software Is Important in Telecommunication Systems

WITH the increasing use of stored program control (SPC) switching systems, programming is becoming an important part of telecommunications technology. This perspective was realized by CCITT already in the late 1960's when some initial investigations on SPC programming were made. In the 1973-1976 Study Period, specific work was started to create a family of languages for SPC use, CHILL, SDL, and MML. SDL is the CCITT Specification and Description Language [4]. MML is the CCITT Man-Machine Language [5].

Manuscript received May 8, 1981; revised December 22, 1981. This paper was presented in part at the National Telecommunications Conference, New Orleans, LA, Nov. 29-Dec. 3, 1981.

The author is with RUNIT—The Computing Centre at the University of Trondheim, N-7034 Trondheim-NTH, Norway.

Most of the problems to be faced in SPC programming were already then fairly well understood, and languages had been or were being developed to cope with these problems. One may therefore ask why CCITT should engage in making another programming language, adding to the already abundant flora of such languages. There were two major reasons for this:

- to provide one standard language covering all producers and users of SPC systems;
- to remedy weaknesses with existing languages when applied to SPC systems and to consolidate, in one language, features only found in a variety of other languages.

The standardization aspect was motivated by the fact that the telecommunications community is a very large one and distributed all over the world. Most producers and users of such equipment will somehow come into contact with software. Thus, the benefits of standardization also in the field of software are likely to be substantial.

CHILL has inherited most of its traits from other high-level programming languages, drawing upon the best of the language developments of the 1970's. CHILL is truly a state-of-the-art language now becoming available for practical use.

The CHILL Work Started in 1975

The work of CCITT in 1973 started with an evaluation of existing languages. The conclusion of this study in 1974 was,