Chapter

1

Introduction to Languages and Notations

This chapter begins with a brief introduction to SDL; the language, its history, its main concepts and application areas.

Next follows an introduction to the MSC language (including High level MSC), the Object Model notation and State Chart notation, and the ASN.1 notation.

Telelogic Tau also comprises the TTCN suite. For the sake of completeness, we have included a brief introduction to the TTCN notation. More information can be found in chapter 1, *Introduction to Languages and Notations, in the TTCN Suite Getting Started*.

After reading this chapter, you may want to deepen your knowledge of SDL. In chapter 1, *Object Oriented Design Using SDL, in the SDL Suite Methodology Guidelines*, you will find information about how to take advantage of the SDL-92 language in an SDL suite environment.

Also, a list of recommended literature dealing with various language topics is enclosed at the end of this chapter; see "References" on page 19.

Benefits of a Specification Language

It is widely accepted that the key to successfully developing a system is to produce a thorough system specification and design. This task requires a suitable specification language, satisfying the following needs:

- A well-defined set of concepts.
- Unambiguous, clear, precise, and concise specifications.
- A basis for verifying specifications with respect to completeness and correctness.
- A basis for determining whether or not an implementation conforms to the specifications.
- A basis for determining the consistency of specifications relative to each other.
- Use of computer-based tools to create, maintain, verify, simulate and validate specifications.
- Computer support for generating applications without the need of the traditional coding phase.

The SDL suite fulfills all the demands outlined in the list above.

General about the SDL Language

SDL (Specification and Description Language) is a standard language for specifying and describing systems[1]. It has been developed and standardized by ITU-T in the recommendation Z.100.

The development of SDL started in 1972 after a period of research work. The first version of the language was issued in 1976 and it has been followed by new versions every fourth year. The latest versions expanded the language considerably, and today SDL is a “complete” language in all senses.

In the SDL suite, there is full support of SDL including some of the SDL-96 concepts. For more information about the SDL support in the SDL suite, see “Compatibility with ITU SDL” on page 9 in chapter 1, *Compatibility Notes, in the Release Guide*.

Modularity

An SDL specification/design (a system) consists of a number of interconnected modules (blocks). A block can recursively be divided into more blocks forming a hierarchy of blocks. The channels define the communication paths through which the blocks communicate with each other or with the environment. Each channel usually contains an unbounded FIFO queue that contain the signals that are transported on the channel. The behavior of the leaf blocks is described by one or more communicating processes. The processes are described by extended finite state machines.

Object Oriented Design

SDL furthermore supports object-oriented design by a type[2] concept that allows specialization and inheritance to be used for most of the SDL concepts, like blocks, processes, data types, etc. The obvious advantage is the possibility to design compact systems and to reuse components which in turn reduces the required effort to maintain a system.

1. No distinction is made in SDL between the terms “specification” and “description”, although they generally have different meanings in SDL applications.
2. SDL has adopted the term *type* which corresponds to the term *class* used in many of the OO notations and programming languages.

Graphical and Textual Notations

SDL gives a choice of two equivalent syntactic forms; a Graphical Representation (SDL/GR) and a textual Phrasal Representation (SDL/PR). The SDL suite supports both notations.

Application Areas

Currently, SDL is mainly known within the telecommunication industry, but it also has broader areas of application and is now gaining acceptance within the real-time software industry. The application areas may be summarized as follows:

- Type of system described by SDL: Real-time, interactive, distributed.
- Type of information provided by SDL: Behavior and structure.
- Level of abstraction supported by SDL: From system overview to functional detail.

More about SDL

Theoretical Model

The basic theoretical model of an SDL system consists of a set of extended finite state machines (FSM) that run in parallel. These machines are independent of each other and communicate with discrete signals.

An SDL system consists of the following components:

- Structure
  - hierarchical decomposition with system, block, process, and procedure as the main building blocks
- Communication
  - asynchronous signals with optional signal parameters
  - remote procedure calls for synchronous communication
- Behavior
  - processes
- Data
  - abstract data types that can be inherited, generalized and specialized
  - ASN.1 data types according to Z.105
- Type Concept
  - describing type hierarchies with inheritance, generalization and specialization

Structure

Figure 1 shows the four main hierarchical levels in SDL: system, block, process, and procedure.

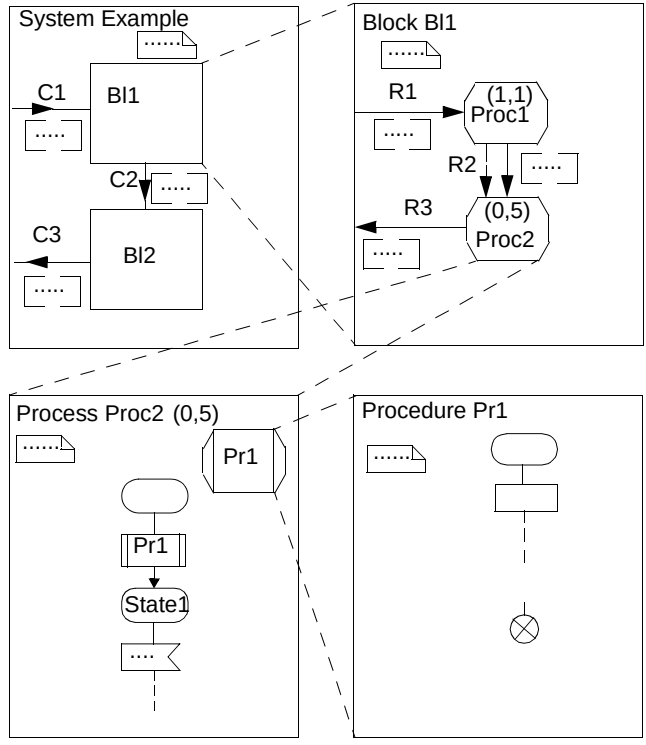

Figure 1: The architectural view of an SDL system

In addition, there is a service concept that can be used within processes. Procedures can be used in both processes and services.

Communication

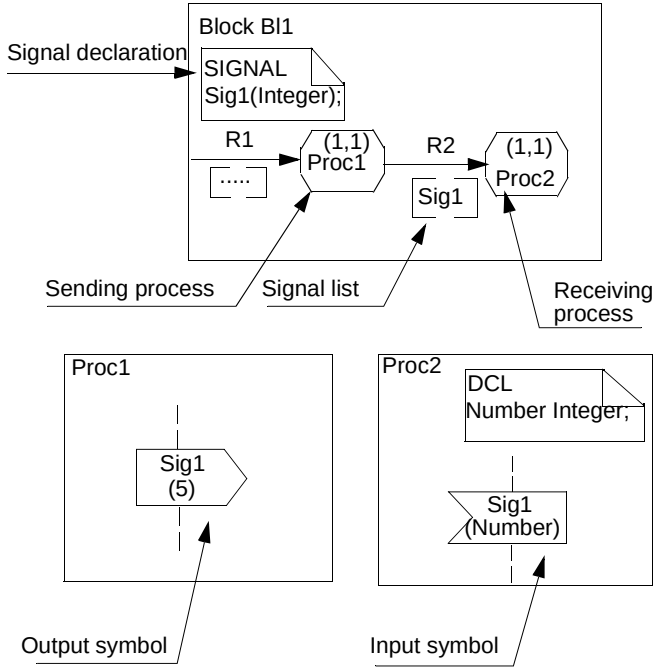

Figure 2: Sending signals between two processes

In SDL, there is no global data. This approach requires that information between processes, or between processes and the environment, must be sent with signals and optional signal parameters. Signals are sent asynchronously, that is, the sending process continues executing without waiting for an acknowledgment from the receiving process.

Synchronous communication is possible via a shorthand, remote procedure call. This shorthand is transformed to signal sending with an extra signal for the acknowledgment.

Behavior

The dynamic behavior in an SDL system is described in the processes. The system/block hierarchy is only a static description of the system structure. Processes in SDL can be created at system start, or created and terminated dynamically at runtime. More than one instance of a process can exist. Each instance has a unique process identifier (PId). This makes it possible to send signals to individual instances of a process. The concept of processes and process instances that work autonomously and concurrently makes SDL a true real-time language.

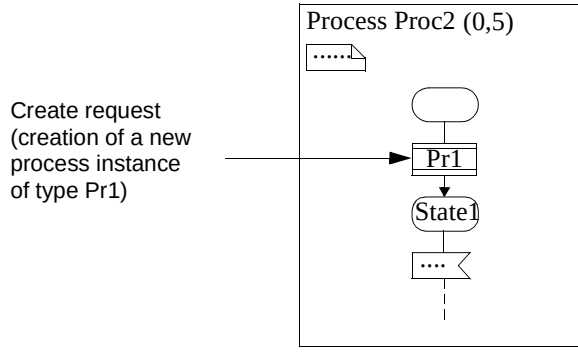

Figure 3: Creation of a new process instance at runtime

Data

The abstract data types concept used within SDL is very well suited to a specification language. An abstract data type is a data type with no specified data structure. Instead, it specifies a set of values, a set of operations allowed on the data type and a set of equations that the operations must fulfil. This approach makes it very simple to map an SDL data type to data types used in other high-level languages.

Alternatively, ASN.1 types can be used in SDL. This is useful when specifying or implementing telecommunication applications that make use of ASN.1. ITU-T Recommendation Z.105 defines how ASN.1 is used in combination with SDL. For more information on ASN.1, see “ASN.1 – Abstract Syntax Notation One” on page 17.

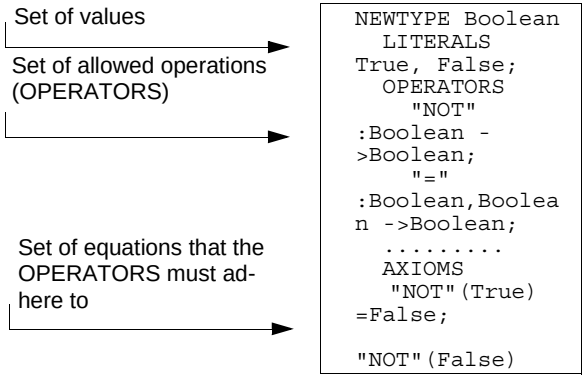

Figure 4: Abstract Data Type example

Type Concept

The object-oriented concepts of SDL give you powerful tools for structuring and reuse. The concept is based on type definitions. All structural building blocks can be typed. Type definitions can be placed anywhere in the system, and also in packages outside the system.

One of the major benefits of using an object oriented language is the possibility to create new objects by adding new properties to existing objects, or to redefine properties of existing objects. This is what is commonly referred to as specialization.

In SDL, specialization of types can be accomplished in two ways:

- A subtype may add properties not defined in the supertype. One may, for example, add new transitions to a process type, add new processes to a block type, etc.
- A subtype may redefine virtual types and virtual transitions defined in the supertype. It is possible to redefine the contents of a transition in a process type, to redefine the contents/structure of a block type, etc.

The Message Sequence Chart Language

History

During the last years, ITU has made a considerable effort in standardizing a formal language which defines Message Sequence Charts (MSC). In the summer of 1992, a first version of the MSC recommendation Z.120 was published.

As defined in the recommendation Z.120, the MSC language offers a powerful complement to SDL in describing the dynamic behavior of an SDL system. Its graphical representation is well suited for presenting a complex dynamic behavior in a clear and unambiguous way which is easy to understand.

There is an extended version of the MSC standard, called MSC'96, as defined in the current Z.120. In the SDL suite, there is support for the most important MSC'96 extensions. See "Compatibility with ITU MSC" on page 13 in chapter 1, *Compatibility Notes, in the Release Guide* for more information.

Plain MSC

An MSC describes one or more traces from one node to another node of an abstract communication tree generated from an SDL specification.

Basically, the information interchange is carried out by sending *messages* from one *instance* to another (see Figure 5). In an SDL specification, those messages would coincide with the signals which are sent from one process and consumed in another process. The instances would correspond to any part of the specification (an SDL system, a block or a process).

An MSC can reference another MSC using an *MSC reference* symbol. MSC references can for example be used to have one MSC describing an initialization sequence and then reference this MSC from a number of other MSCs.

The reference symbol may not only refer to an MSC but can also contain MSC reference expressions that reference more than one MSC. This construct gives a very compact MSC representation and it also provides an excellent means for reusability of certain MSCs.

By using *inline operator expressions*, several MSC scenarios can be composed in a single diagram. The same structures of events can be expressed as with MSC reference symbols.

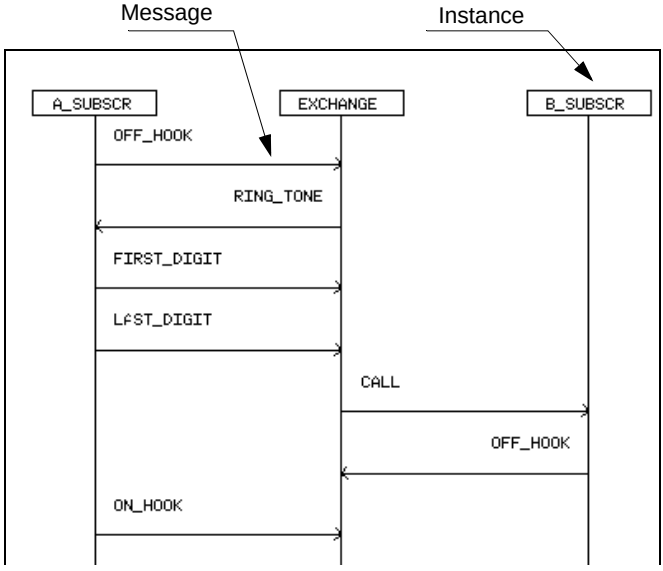

Figure 5: An example of a simple Message Sequence Chart

High-Level MSC

A high-level MSC (HMSC) provides a means to graphically define how a set of MSCs can be combined. Contrary to plain MSCs, instances and messages are not shown within an HMSC, but it focus completely on the composition aspects. HMSCs can be hierarchically structured, i.e. it is possible to refine HMSCs by other HMSCs. The power of the MSC language is considerably improved with the new concepts introduced with HMSCs. It is e.g. much easier to specify a main scenario together with all accompanying exceptions.

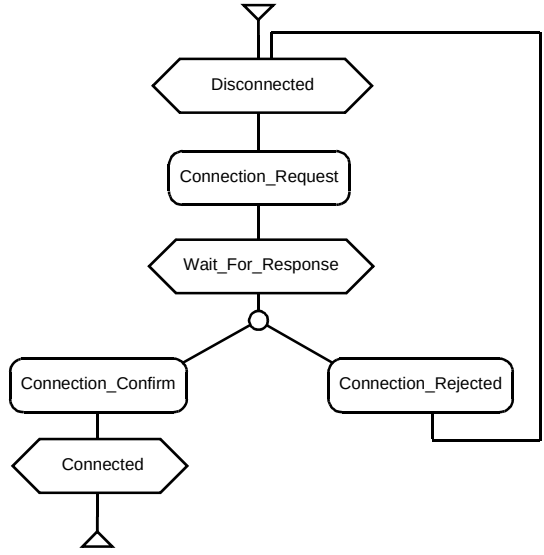

Figure 6: Example of an HMSC

Graphical and Textual Notations

The MSC language supports two notations which are equivalent. Besides the graphical notation (MSC-GR), a textual notation (MSC-PR) is standardized since the autumn of 1994.

Application Areas

Among the various application areas, we have selected the following:

- Producing documents with the purpose of defining the requirements of a system.
- Facilitating the design phase, by identifying and documenting a multitude of dynamic cases before starting designing with SDL.
- Presenting the execution of a simulation as a graphical output which is easy to understand and which can later on be verified against a reference. Message Sequence Charts can be verified against an SDL system using the SDL suite.
- Presenting the execution trace of an SDL system during an interactive simulation and generation of reports.

Object Model Notation

The object model notation used in the SDL suite is an adaptation of the notations used in OMT (Object Modeling Technique) and UML (Unified Modeling Language). The OMT/UML notation is a commonly accepted graphical notation that is used for drawing diagrams that describe objects and the relations between them.

Class

The most important concept in an object model is the class definition. A class is a description of a group of similar objects that share the properties defined by the class. The properties of a class are described with attributes and operations. The object model notation for a class is exemplified in Figure 7, where the second class definition also shows how to define attributes and operations.

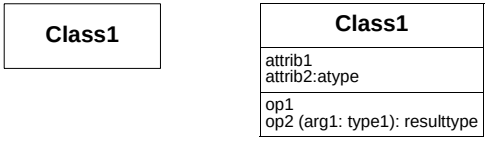

Figure 7: A collapsed class symbol and a class symbol with attributes and operations

Classes may inherit attributes and operations from other classes, known as specialization and generalization. The object model notation for this is shown in Figure 8.

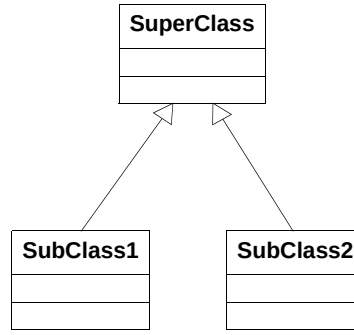

Figure 8: inheritance between classes

Relations and Multiplicity

Classes may be physically or logically related to each other. This is shown in the object model by means of associations as shown in Figure 9. An association may have a name and/or the endpoints of the association may be labeled by the role of this endpoint.

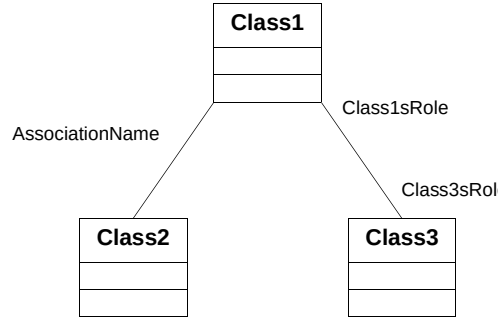

Figure 9: Associations between classes

Aggregation is special kind of association, indicating a “consists of” relation. It has its own notation as shown in Figure 10.

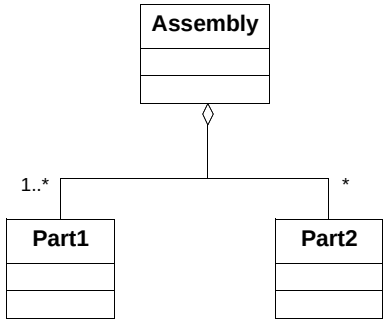

Figure 10: Aggregation

The endpoints of associations and aggregations may have a multiplicity according to the following:

- No multiplicity (exactly one)
- `*` (zero or more)
- `0,1` (zero or one)
- `1..*` (one or more)
- `1..3,6,10..*` (several intervals: 1, 2, 3, 6, 10 or more)

Objects

Besides class definitions, object models may also contain objects (instances) and their relations. The relation that exists between objects are links, which corresponds to associations for classes. The object symbol has one field containing the object name and a reference to the class ("name:class"), and an attribute field where constant or default values can be assigned to the object attributes. See Figure 11.

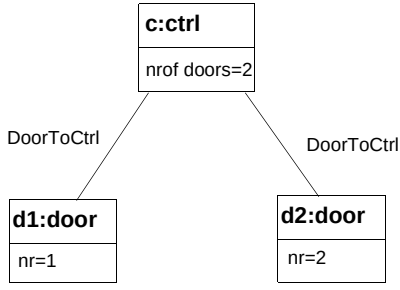

Figure 11: Objects related by links

State Chart Notation

The state chart notation used in the SDL suite is a subset of the notations used in OMT (Object Modeling Technique) and UML (Unified Modeling Language).

A state chart model is suitable to use together with class and object models. The descriptions of the behavior of a class in a class diagram is collected into a state chart which describes the dynamic view of the model by means of *states* and *transitions* between states.

State

A state symbol describes the name of the class, state variables, and internal activities. State variables highlight attributes of the class which is used or in some way affected by the behavior described in the state chart. Internal activities are taking place upon entering the state, while in the state and when exiting the state. Activities are described by specifying events and associated actions. Figure 12 shows a collapsed state and a state with state variables and events.

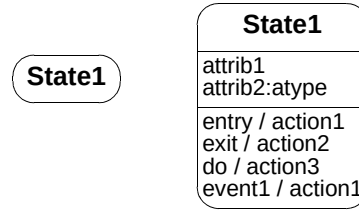

Figure 12: A collapsed state symbol and a state symbol with state variables and events

Transition

A transition symbol is an arrow which typically connects two state symbols. A transition is triggered by an event together with a condition, and a transition then executes an action; see Figure 13.

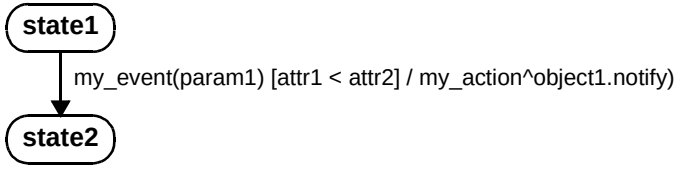

Figure 13: The transition from state1 to state2 is triggered by the event my_event and the condition that attr1 is less than attr2

Start and Termination Symbol

The start symbol denotes the starting point of a state machine described by a state chart and the termination symbol denotes the point of termination of a state machine. Figure 14 shows a simple state machine, describing the behavior of a door, including start and termination symbols.

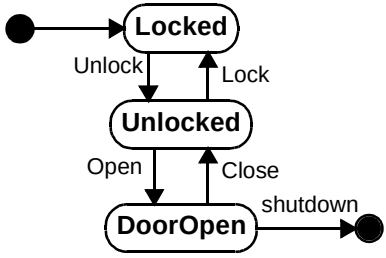

Figure 14: A simple state chart with a start symbol and a termination symbol

Substates

States may be refined into nested diagrams of sub-states, or hierarchical states. The state represents a simplification of more complex behavior expressed in the nested diagram.

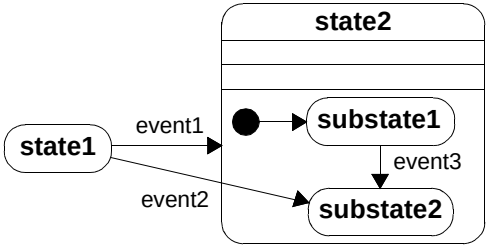

Figure 15: A state with substates

ASN.1 – Abstract Syntax Notation One

ASN.1 (ITU-T Recommendation X.680-683) is a generic notation standardized by ISO and ITU for the specification of data types and values. The general idea behind ASN.1 is to describe data type information independent of the transfer format.

The original use of ASN.1 has been the information description in high-level protocols like FTAM, CMIP, MHS, DS, VT, etc. Today it is also frequently used in numerous other telecommunication protocols and applications.

ASN.1 data types and values can be defined in modules that can be used in TTCN and in SDL. This makes it possible to use the data types of an application both in the SDL specification and in the TTCN test suite, which assures consistency between the information transferred in the system specification and the test specification.

The TTCN Notation

As the use of standards within the world of Information Technology and Telecommunications has increased tremendously during the last decade, so has the need for methods and tools that support the verification and validation of both the standards and their actual implementations.

This need has been addressed by ISO and ITU in the “Framework and Methodology for Conformance Testing of Implementations of OSI and ITU Protocols”. The framework has for some time had the status of an international standard as ISO/IEC 9646 (or X.290).

The standard introduces the concept of abstract test suites (consisting of abstract test cases). This is a description of a set of tests that should be executed for a system. The tests should be described using a black-box model, i.e. only control and observation using the available interfaces.

The abstract tests are to be described using a formal language rather than using informal natural language. As part of the standard the language TTCN is defined in order to describe the abstract tests.

TTCN – Tree and Tabular Combined Notation

With TTCN a test suite is specified. This is a collection of various test cases together with all the declarations and components it needs.

Each test case is described as an event tree. In this tree behaviors like "First we send A, then either B or C will be received, if it was B we will send D..." are described. The new version of TTCN allows several event trees to be running concurrently.

TTCN is abstract in the sense of being independent of the actual test systems. This means that a test suite in TTCN for one application (protocol, system...) can be used in any test environment for that application.

The use of TTCN has increased tremendously during the last years. This has been augmented by the significant amount of TTCN test suites released by various standardization bodies. TTCN is however not only used in standardization work. The language is very suitable for all kinds of functional testing for communicating systems. This has led to a wide usage also within the industry.

The specifications of the messages being sent and received can be defined using either the built-in form of TTCN or by using ASN.1.

Tool Support

Telelogic has been a firm supporter of SDL for a long time. We co-operate with ITU in the on-going work of improving the language and with ETSI in using SDL for defining protocol standards. We initiate and participate in international research programs on how to use the language in different application areas (such as the European Community programs RACE, ESPRIT and EUREKA). Our experience and know-how in these areas is put to practice when we develop software engineering tools that support the languages.

A tool for a specification language must be able to create, maintain, and analyze a specification. It is also fundamental that the tool can simulate, validate and generate application code to other high level languages.

The SDL suite can do all of this.

References

[1] ITU Recommendation Z.100:
Specification and Description Language (SDL)
1994, ITU, General Secretariat - Sales Section,
Places des Nations, CH-1211 Geneva 20

[2] Annex A, B, C1, C2 D, E, F1, F2 and F3 to Z.100, as above

[3] ITU Recommendation Z.120:
Message Sequence Charts (MSC)
1992, ITU General Secretariat - Sales Section
Place des Nations, CH-1211 Geneva 20

[4] Jan Ellsberger, Dieter Hogrefe, Amardeo Sarma:
SDL – Formal Object-oriented Language for Communicating Systems.
Prentice Hall Europe (1997)
ISBN 0-13-632886-5

[5] A. Olsen, O. Færgemand, B. Møller-Pedersen, R. Reed, J.R.W. Smith:
Systems Engineering Using SDL-92.
Elsevier (1994)
ISBN 0-444-89872-7

[6] Ferenc Belina, Dieter Hogrefe, Amardeo Sarma:
SDL with Applications from Protocol Specification.
Prentice Hall International (UK) Ltd. (1991)
ISBN 0-13-785890-6

[7] Belina, Hogrefe:
The CCITT Specification and Description Language SDL
Computer Networks and ISDN System.
North-Holland, Amsterdam (1988/1989)

[8] Bræk, Gorman, Haugen, Melby, Møller-Pedersen, Sanders:
TIMe - The Integrated Method
SINTEF 1998
`http://www.sintef.no/time`

[9] Færgemand, Marques (editors):
SDL 89: The language at work.
Proceedings of the Fourth SDL Forum,
North Holland, Amsterdam (1989)

[10] Færgemand, Reed (editors):
SDL 91: Evolving Methods.
Proceedings of the Fifth SDL Forum,
North-Holland, Amsterdam (1991)

[11] Færgemand, Sarma (editors):
SDL 93: Using Objects.
Proceedings of the Sixth SDL Forum,
North-Holland, Amsterdam (1993)

[12] Haugen, Møller-Pedersen:
Tutorial on object-oriented SDL.
SISU Project Report 91002
Norwegian Computer Center
PO Box 114, N-0314 Oslo 3, Norway

[13] Behcet Sarikaya:
Principles of Protocol Engineering and Conformance Testing.
Simon & Schuster International (1992)

[14] Sarraco, Smith, Reed:
Telecommunications system engineering using SDL.
North-Holland, Amsterdam (1989)

[15] K.J. Turner (editor):
Using Formal Description Techniques -
An Introduction to Estelle, LOTOS and SDL.
John Wiley & Sons (1992)

[16] ITU Recommendation X.680-683
Abstract Syntax Notation One (ASN.1)
1994, ITU, General Secretariat- Sales Section,
Places des Nations, CH-1211 Geneva 20

[17] ITU Recommendation Z.105
SDL Combined with ASN.1 (SDL/ASN.1)
1995, ITU, General Secretariat- Sales Section,
Places des Nations, CH-1211 Geneva 20