Specification and Description Language (SDL)

Definition

Specification and description language (SDL) is an object-oriented, formal language defined by The International Telecommunications Union–Telecommunications Standardization Sector (ITU–T) (formerly Comité Consultatif International Telegraphique et Telephonique [CCITT]) as recommendation Z.100. The language is intended for the specification of complex, event-driven, real-time, and interactive applications involving many concurrent activities that communicate using discrete signals.

Overview

This tutorial discusses the applications and reasons for the use of specification and description language (SDL). Over the last decade, the size of produced software has increased dramatically. More and more systems are multiprocess and distributed, and they execute in a heterogeneous environment. It is increasingly accepted within a steadily growing range of industrial segments that the best way to meet the needs of these systems is through formal methods. Furthermore, as the international market grows, equipment from different manufacturers must be able to communicate with each other. Therefore, the formal methods should be internationally standardized. Telecommunications software engineers have developed such methods and tools for the development of complex real-time software. SDL is an object-oriented formal language defined by the ITU–T for specification of complex, real-time applications. The strength of SDL is its ability to describe the structure, behavior, and data of a system.

Topics

1. Benefits of a Specification Language
2. History
3. SDL Characteristics
4. Theoretical Model and Structure
5. Sharing, Reuse, and Maintenance
6. Openness, Portability, Scalability, and Distributed Applications

7. Graphical and Textual Notations and Applications Areas

   Self-Test

   Correct Answers

   Glossary

1. 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 thorough and accurate basis for analyzing specifications
- a basis for determining whether or not an implementation conforms to the specifications
- a basis for determining the consistency of specifications
- computer support for generating applications without the need for the traditional coding phase

SDL has been defined to meet these demands. It is a graphical specification language that is both formal and object-oriented. The language is able to describe the structure, behavior, and data of real-time and distributed communicating systems with a mathematical rigor that eliminates ambiguities and guarantees system integrity. It has a graphic syntax that is extremely intuitive. Even nonconstructors quickly obtain an overview of a system's structure and behavior. The most important characteristic of SDL is its formality. The semantics behind each symbol and concept are precisely defined. Above all, the great strength of SDL lies in describing large real-time systems.

2. History

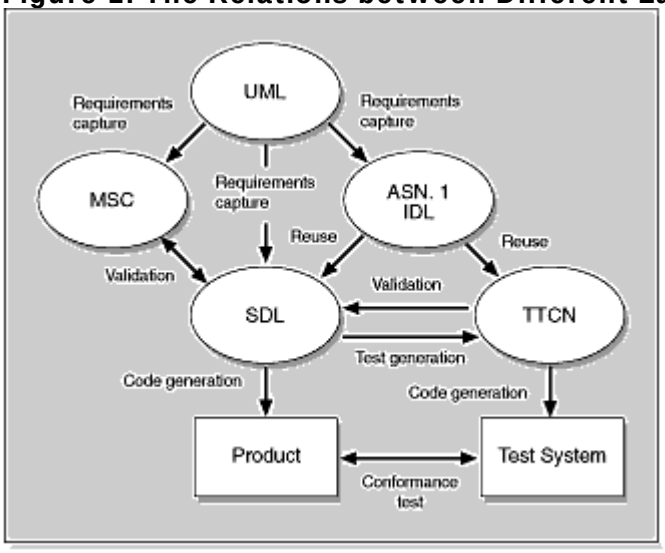
The development of SDL started in 1972. A 15-member study group within CCITT representing several countries and large telecom companies like Bellcore, Ericsson, and Motorola began research on a standard specification language for the telecommunications industry. The first version of the language was issued in 1976, followed by new versions in 1980, 1984, 1988, 1992, and 1996. The latest

versions expanded the language considerably and simplified interfacing. Today SDL is a complete language in all senses.

SDL and Other Languages

SDL is well suited to be the core of full-scale projects because of its abilities to interface with other languages. Such languages include other high-level notations for analysis such as object modeling technique (OMT)/unfied modeling language (UML) object models and mobile switching center (MSC) use-cases, as well as abstract system notation one (ASN.1) or common object request broker architecture (CORBA)/interface description language (IDL) data-type definitions. Furthermore, there are tools available that can generate executable code—for example, C/C++ or ITU high-level language (CHILL), directly from the SDL design. Tests can also be generated from the SDL specification by making a test suite in tree and tabular combined notion (TTCN). See *Figure 1* for the relations between these languages.

Figure 1. The Relations between Different Languages and SDL



Typically, the procedure from requirements analysis to product implementation and testing would involve the following steps:

- Collect the initial requirements in a text document.
- Study system analysis results in a number of OMT/UML object models and MSC use-cases depicting typical scenarios. The resultant classes are implemented in SDL as SDL block diagrams and SDL/ASN.1/IDL data-type definitions.

- Complete the SDL diagrams and ASN.1 or IDL specifications to a level where they can be simulated and checked for consistency with the system requirements analysis.

- Use verification and validation to determine whether required properties are correctly and completely implemented. The verification procedure also detects general errors like deadlocks, signal races, loss of signals, etc. When SDL design has proved consistent with the requirements, a code for the application can be generated.

- Make a test suite in TTCN. Tests can be generated from the SDL specification. In some cases, such tests are already available (e.g., from standardization bodies).

- Generate code to create an executable test suite that can be run in a test system.

- Run the executable tests and test the application in the target environment.

3. SDL Characteristics

SDL is a design and implementation language dedicated to advanced technical systems (i.e., real-time systems, distributed systems, and generic event-driven systems where parallel activities and communication are involved). Typical application areas are high- and low-level telecom systems, aerospace systems, and distributed or highly complex mission-critical systems.

SDL has a set of specialized characteristics that distinguishes it from other technologies:

- **standard**—SDL is a nonproprietary internationally standardized language (ITU–T standard Z.100 and Z.105).

- **formal**—SDL is a formal language ensuring precision, consistency, and clarity in the design that is crucial for mission-critical applications (e.g., most technical systems).

- **graphical and symbol-based**—SDL is a graphical and symbol-based language providing clarity and ease of use. An SDL design is both an implementation and its own documentation.

- **object-oriented (OO)**—SDL is a fully OO language supporting encapsulation, polymorphism, and dynamic binding. Moreover, SDL extends the traditional data-oriented OO class concept by customizing

it for technical applications and introducing OO concepts for active objects (e.g., systems, blocks, and state machines).

- **highly testable**—SDL has a high degree of testability as a result of its formalism for parallelism, interfaces, communication, and time. The quality and speed improvements are dramatic compared to traditional nonformal design techniques.

- **portable, scalable, and open**—SDL is portable, scalable, and open. SDL implementations are independent of cross compilers, operating systems, processors, interprocess communication mechanisms, and distribution methods. A single SDL implementation can be used for many different target architectures and configurations.

- **highly reusable**—SDL provides a high degree of reuse. Because of visual clarity, testability, OO concepts, clear interfaces, and abstraction mechanisms, SDL design has a much higher degree of reusability than any other type of design or implementation.

- **efficient**—The formalism and the level of abstraction that is provided by SDL make it possible to apply sophisticated optimization techniques for cross-compilation.

4. Theoretical Model and Structure

Theoretical Model

The basic theoretical model of an SDL system consists of a set of extended finite state machines (FSMs) 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—**system, block, process, and procedure hierarchy

- **communication**—signals with optional signal parameters and channels (or signal routes)

- **behavior**—processes

- **data**—abstract data types (ADT)

- **inheritance**—describing relations and specialization

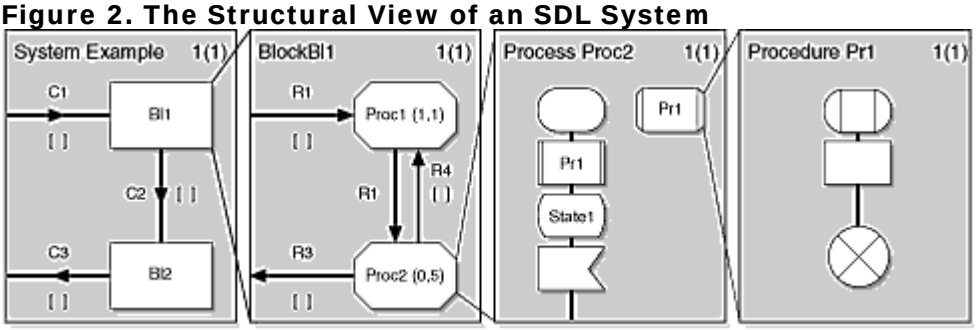
The following subsections introduce the basic concepts.

Structure

SDL comprises four main hierarchical levels:

- system
- blocks
- processes
- procedures

Figure 2. The Structural View of an SDL System



Dividing a system into a system, block, and process hierarchy is called partitioning a system. The objectives of partitioning include the following:

- hiding information (move details not important in an overview to lower levels)
- following natural functional subdivisions
- creating modules of intellectually manageable sizes
- creating a correspondence with actual software or hardware
- reusing already-existing specifications

Each SDL process type is defined as a nested hierarchical state machine. Each substate machine is implemented in a procedure. Procedures can be recursive; they are local to a process or they can be globally available depending on their scope. SDL also supports the remote procedures paradigm, which allows one to make a procedure call that executes in the context of another process.

SDL processes have separate memory spaces, (i.e., data is local to a process or procedure). This is a highly important aspect that dramatically reduces the number of deficiencies and increases robustness.

A set of processes can be logically grouped into a block (that is, subsystem). Blocks can be nested inside each other to recursively break down a system into smaller and maintainable encapsulated subsystems. These break-down mechanisms are important for large team development efforts, and SDL simplifies this by also providing clear interfaces between subsystems.

Static and Dynamic Structure

The static structure of a system is defined in terms of blocks and channels. A block is perceived as a module with the well-known black box model.

The dynamic structure is defined with the help of the process and the signal route concepts. A process is an independent device that reacts to stimuli in the form of signals (the process concept is described more fully in the Behavior subsection).

Communication

SDL does not use any global data. SDL has two basic communication mechanisms: asynchronous signals (and optional signal parameters) and synchronous remote procedure calls. Both mechanisms can carry parameters to interchange and synchronize information between SDL processes and with an SDL system and its environment (e.g., non–SDL applications or other SDL systems).

SDL defines clear interfaces between blocks and processes by means of a combined channel and signal route architecture. This communication architecture with formally clear signal interfaces simplifies large team development and ensures consistency between different parts of a system.

SDL defines time and timers in a clever and abstract manner. Time is an important aspect in all real-time systems but also in most distributed systems. An SDL process can set timers that expire within certain time periods to implement time-outs when exceptions occur but also to measure and control response times from other processes and systems.

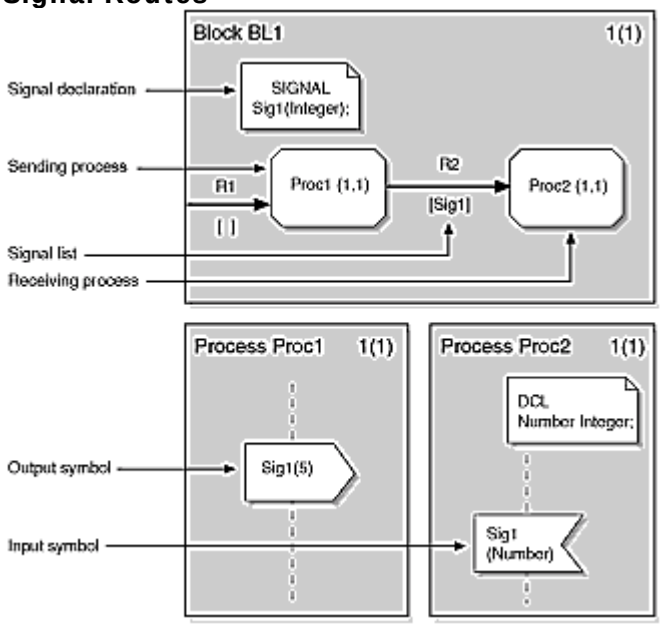
When an SDL timer expires, the process that started the timer receives a notification (signal) in the same way as it receives any other signal. Actually an expired timer is treated in exactly the same way as a signal. SDL time is abstract in the sense that it can be efficiently mapped to the time of the target system, be it an operating system timer or hardware timer. This makes it possible to simulate time in SDL models before the target system is available.

Other aspects of the signaling concept in SDL are as follows:

- Signal and process priorities are not within the scope of SDL. These issues are left instead to the implementation phase where the user with special directives can assign signal and process priorities.

- An SDL signal can only be sent to one specific process instance at a time. To enable broadcasting the user can include a package with some general-purpose functions that will provide a broadcasting mechanism in the implementation.

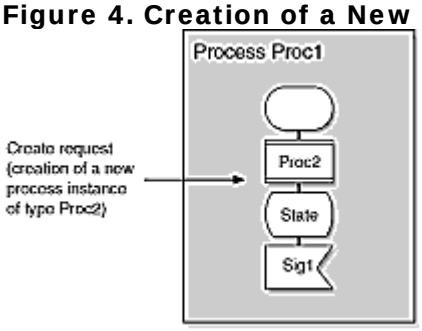
Figure 3. Signals between Two Processes Travel through Channels between Blocks and from One Process to Another via Signal Routes



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 at run time. 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 4. Creation of a New Process Instance at Runtime

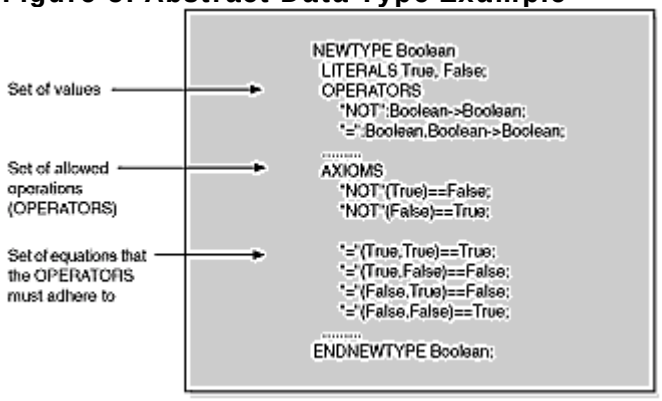


Data

SDL accepts two ways of describing data, abstract data type (ADT) and ASN.1. The integration of ASN.1 enables sharing of data between languages, as well as the reuse of existing data structures.

The ADT 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, and a set of equations that the operations must fulfill. This approach makes it simple to map an SDL data type to data types used in other high-level languages.

Figure 5. Abstract Data Type Example



The set of predefined sorts in SDL makes it possible to work with data in SDL in a traditional way. Variables of standard sorts, such as the following, can be declared:

- integer
- real
- natural

- boolean
- character
- duration
- time
- charstring
- PId
- complex data sorts (these can be created with Array and Struct as depicted in *Figure 6*)

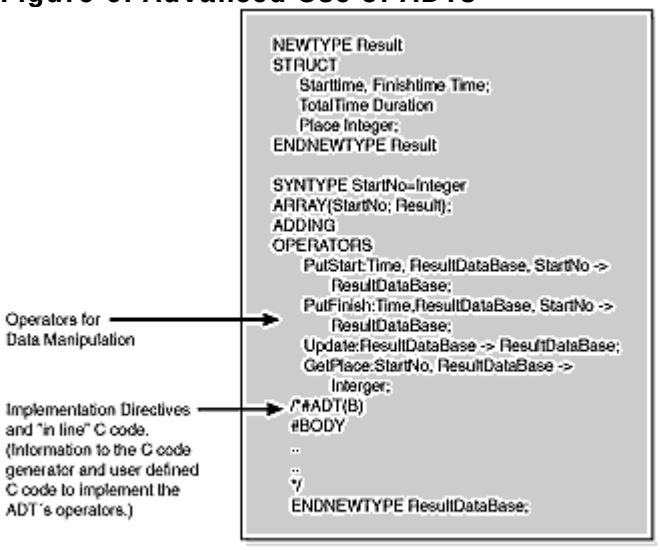
A description of a more advanced use of ADT follows, where the operator concept is used for hiding data manipulation.

How to Use Advanced ADTs

ADTs in SDL can be used for much more than representing data, such as for the following:

- hiding data manipulation
- hiding algorithmic parts of a specification
- creating an interface to external routines

Figure 6. Advanced Use of ADTs

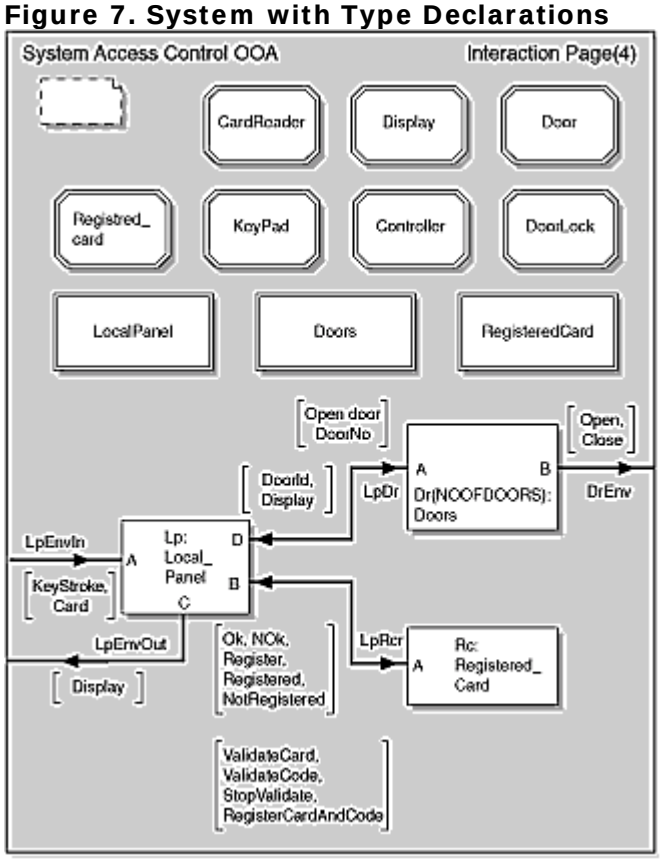


As *Figure 6* depicts, data manipulation is hidden in operators. The function of the operator update is to update the complete result database and recalculate place for all participants after new results. This is an example of a sorting-and-seeking algorithm that is much better to hide in operators than to express in ordinary graphical SDL. Still, the operator should be described using SDL diagrams.

Inheritance

The OO concepts of SDL give the user powerful tools for structuring and reuse. The concept is based on type declarations. Type declarations can be placed anywhere, either inside the system close to their context, or at system level. *Figure 7* shows an access control system with block and process types at system level. Type declarations can also be placed in packages outside the system, for sharing with other systems.

Figure 7. System with Type Declarations



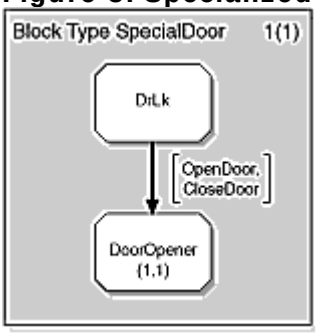
One of the major benefits of using an object-oriented language is the simple and intuitive way new objects can be created by adding new properties to existing objects or by redefining properties of existing objects. This is what is commonly referred to as specialization.

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

- A subtype might add properties not defined in the supertype. One can, for example, add new transitions to a process type, add new processes to a block type, etc. (see *Figure 8*).

- A subtype can 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.

Figure 8. Specialized Block Type



5. Sharing Information, Reuse, and Maintenance

Sharing Information—Packages

SDL packages are graphical SDL libraries that define data structures, signals, process types, block types, and system types that can be shared between SDL systems and projects. This facilitates maintenance and reuse aspects for large applications and allows for sharing information between many systems.

Reuse and Maintenance (Specialized Inheritance and Polymorphism)

Apart from supporting object-oriented data (i.e., object-oriented passive objects) SDL also supports all object-oriented features for active objects—for example, systems, blocks, and state machines down to the transition level. This extends the traditional passive class concept that is more oriented towards nontechnical applications and is found in UML, OMT, C++, and Java. SDL specializes it for technical applications (i.e., real-time systems, distributed systems, and event-driven systems, where there is a heavier focus on communication and active state-oriented objects).

6. Openness, Portability, Scalability, and Distributed Applications

Openness

The latest SDL standard (SDL–96) defines external procedures (i.e., procedures that are implemented outside an SDL system). These procedures can be implemented in languages other than SDL, such as C code.

With the ITU standard Z.105, SDL is combined with ASN.1. This extension to SDL allows a choice between declaring data according to the native SDL syntax or according to ASN.1. ASN.1 modules are treated as SDL packages and can, for example, be shared between an SDL design and a TTCN test suite.

Portability and Scalability

A key feature of SDL is its abstraction mechanisms for seamless portability between cross-compilers and operating systems. Moreover, the same abstraction mechanisms permit the user to choose how to map SDL processes to physical processes, IPC (interprocess communication) schemes, and time according to what is most efficient in each actual case. The same implementation can be used for many different configurations and different kernels, ranging from monotasking small systems to multiprocessor high-end systems.

Distributed Applications

The same abstraction mechanisms that make SDL implementations independent of cross-compilers, operating systems, and IPC and process mapping schemes also make an SDL system independent of distribution architecture and distribution method. This makes SDL the perfect language for modeling and implementing distributed systems. One SDL model supports many physical distribution configurations.

7. Graphical and Textual Notations and Application Areas

Graphical and Textual Notations

The SDL language supports two equivalent notations. In addition to the graphical notation (SDL–GR), the textual notation (SDL–PR) is standardized.

Application Areas

Although SDL evolved within telecommunications, it is becoming increasingly popular in other industries as well. Some examples of applications of SDL outside the telecommunication area include the following:

- satellite communications
- aeronautical standardization

- medical equipment
- railway control system
- communication protocols in cars

Self-Test

1. What distinguishes SDL from other graphical specification languages?

    a. SDL is both natural and intuitive.

    b. SDL is formal, object-oriented, and standardized.

    c. SDL is modern.

    d. SDL is used internationally.

2. Why is SDL particularly well-suited for the development of telecommunications applications?

    a. SDL is able to describe real-time and distributed communicating systems.

    b. SDL contains special telecommunications protocols.

    c. SDL is well established.

    d. SDL is owned by 15 large telecommunications companies.

3. In what way is object-orientation extended in SDL as compared to other object-oriented languages?

    a. SDL introduces polymorphism.

    b. SDL introduces data-oriented object-oriented classes.

    c. SDL introduces testability.

    d. SDL introduces object-oriented concepts for active objects.

4. How can time be simulated without access to the target system?

    a. SDL time is abstract and can be mapped to the time of the target system.

    b. The user can include a package that will provide this mechanism.

c. The user can assign signal and process priorities with special directives.

d. SDL can set timers that measure response times.

5. Why is it useful to be able to break down a system into encapsulated subsystems?

   a. The subsystems can have different timers.

   b. The SDL program can run on smaller computers.

   c. It makes large team development easier.

   d. The subsystems can be of different sizes.

6. The dynamic behavior in an SDL system is described in _____.

   a. the system/block hierarchy

   b. the channels

   c. the timers

   d. the processes

7. Procedures outside an SDL system cannot be implemented in a language other than SDL.

   a. true

   b. false

8. Applications of SDL are confined to the telecommunications industry.

   a. true

   b. false

9. The graphical and textual notations of SDL are equivalent.

   a. true

   b. false

10. ADTs in SDL can be used for hiding data manipulation and algorithmic parts of a specification.

    a. true

b. false

Correct Answers

1. What distinguishes SDL from other graphical specification languages?

   a. SDL is both natural and intuitive.

   b. SDL is formal, object-oriented, and standardized.

   c. SDL is modern.

   d. SDL is used internationally.

   See Definition.

2. Why is SDL particularly well-suited for the development of telecommunications applications?

   a. SDL is able to describe real-time and distributed communicating systems.

   b. SDL contains special telecommunications protocols.

   c. SDL is well established.

   d. SDL is owned by 15 large telecommunications companies.

   See Topic 1.

3. In what way is object-orientation extended in SDL as compared to other object-oriented languages?

   a. SDL introduces polymorphism.

   b. SDL introduces data-oriented object-oriented classes.

   c. SDL introduces testability.

   d. SDL introduces object-oriented concepts for active objects.

   See Topic 3.

4. How can time be simulated without access to the target system?

   a. SDL time is abstract and can be mapped to the time of the target system.

b. The user can include a package that will provide this mechanism.

c. The user can assign signal and process priorities with special directives.

d. SDL can set timers that measure response times.

See Topic 4.

5. Why is it useful to be able to break down a system into encapsulated subsystems?

a. The subsystems can have different timers.

b. The SDL program can run on smaller computers.

c. It makes large team development easier.

d. The subsystems can be of different sizes.

See Topic 4.

6. The dynamic behavior in an SDL system is described in _____.

a. the system/block hierarchy

b. the channels

c. the timers

d. the processes

See Topic 4.

7. Procedures outside an SDL system cannot be implemented in a language other than SDL.

a. true

b. false

See Topic 4.

8. Applications of SDL are confined to the telecommunications industry.

a. true

b. false

See Topic 7.

9. The graphical and textual notations of SDL are equivalent.

    a. true

    b. false

    See Topic 7.

10. ADTs in SDL can be used for hiding data manipulation and algorithmic parts of a specification.

    a. true

    b. false

    See Topic 4.

Glossary

ADT
abstract data type

ASN.1
abstract syntax notation one; a standardized language for specification of data objects and structures in an implementation-independent fashion; the language is defined as part of ISO standard 8822

CCITT
Comité Consultatif International Telegraphique et Telephonique; the former name of ITU–T

CORBA
common object request broker architecture; a standard set by OMG that specifies ways for server and client objects to interact with each other

FSM
finite state machines

IDL
interface description language; a key component of CORBA; a textual language that enables designers to capture interfaces and data types of objects

IPC
interprocess communication

ITU
International Telecommunications Union; a UN agency with some 200 member nations; umbrella organization for telecommunications

OMG
object management group; a standardization body

ITU–T
the telecommunications standardization sector of ITU

OMT
object modeling technique; a notation for capturing requirements with object analysis

OO
object-oriented

SDL
specification and description language; a formal language defined by ITU–T as recommendation Z.100

SDL–96
SDL version 1996

SDL–GR
graphical notation for SDL

SDL–PR
textual notation for SDL

TTCN
tree and tabular combined notation; TTCN is a specialized subformal language for protocol conformance testing; standardized as ISO 9646-3 and used for specifying sequences of events comprising abstract test cases

UML
unified modeling language; a notation for capturing requirements with object analysis; UML is a synthesis of the popular techniques from Rumbaugh, Booch, and Jacobsson