

# The Evolution of SDL-2000

RICK REED



*Rick Reed's (53) work in software support systems (see page 20) led to his participation in the ITU group on CHILL 1977–80. His work on software methods from 1982 onwards led to his participation in the ITU SDL group while still in ITU-T SG11. He was associate rapporteur for data 1984–1988 and for methodology 1992–1996, and rapporteur for the whole of SDL 1996–2000. For the 2001–2004 study period he is chairman of the Modelling Languages Working Party, which includes SDL and MSC. Rick Reed has been head of the UK delegation since 1988. He has been involved in several projects at ETSI concerning SDL. He was a founding member of the SDL Forum Society, its first Treasurer, and as Secretary he is now actively organising the 10th SDL Forum for 26–29 June, 2001.*

*rickreed@tseng.co.uk*

In November 1999 SDL-2000 became the latest international Recommendation of the ITU-T in force for specification and description of telecommunications systems and standards replacing the previous version. SDL is a language that has evolved to meet changing years over a period of more than a quarter of a century. This article recounts the history of SDL-2000, tracking the previous versions from 1976. An account is given of the latest changes to the language, followed by the author's opinions of the direction of evolution for the future.

## 1 History of SDL

In 1968, “stored program control” (SPC) systems were just coming into use for telecommunications. CCITT (replaced by ITU-T in 1993) decided that its Study Group 11, which dealt with signalling and switching, should assess the impact of SPC on telecommunications standards. At the end of the 4 year study period in 1972, the result was to launch studies on languages for human machine interaction (called at that time “man-machine language” – MML), specification and description, and programming.

### 1.1 SDL-84 Evolution

The study continued on the specification and description language (SDL) and the first Recommendation (23 pages containing some symbols for “state transition diagrams”, definition of concepts and a few examples) was published in the CCITT Orange book in 1976. It was assumed that the rules for use and semantics were intuitively understood, as many organisations were using different forms of state transition diagrams. The 1980 CCITT Yellow book had the first real description of today's SDL in 72 pages. This was still very informal but recognised that the need to find errors at the specification stage and to provide good tool support, the SDL drawings needed their meaning to be defined more formally than in the 1976 Recommendation.

The metamorphosis from a graphical drawing technique with loose semantics to a formal description technique took place in the following four years, so that the 1984 CCITT Red Book contained an interpretation model that treated the drawings as the basis of mathematical graphs. Additional language features were added, so that the drawings represented process graphs (with several concurrent process instances, which might be created or stopped dynamically), and such that branching in the graphs and passing information by signals between processes depended on data. The basic language of states and transitions triggered by signals between

processes did not change, but the 1984 version enriched the language with many features. Data and a textual form were added to the language. User guidelines were published.

### 1.2 SDL-88

The increasing importance of software was reflected in the creation of Study Group 10 (Languages for telecommunication applications) which between 1984 and 1988 improved the formal basis for SDL. An added incentive was given by work on support tools. The semantics were defined in an abstract way independent of whether the concrete language was the textual Phrase Representation (PR) form, or the Graphical Representation (GR) form. The infrastructure (abstract syntax, textual and graphical grammars, semantics and models) of the current SDL-2000 [1] and previous SDL-92 [2] Recommendations were established. Work was accelerated to complete the formalisation in two years by early 1987. The SDL-88 Recommendation [3] in the CCITT Blue book was about 200 pages (not including the formal definition in VDM or the user guidelines). During this period the data definitions part was aligned with the evolving ISO standard LOTOS [4] using the same algebraic model. This required few changes to the syntax of the language, and gave data a sound mathematical basis.

SDL-88 is the foundation of all the subsequent versions. Articles [5], [6], [7] or [8] provide tutorial material.

### 1.3 SDL-92

At the start of the next period, 1988 to 1992, there was a requirement to keep the language stable, but it was also realised that there were major user benefits if the language could allow for re-usable objects. The main evolution in this period was the addition of object features as an extension. A **block** (or **process**) **type** can be used to define a different **block** (or **process** respectively) at each place where it is used.

Remote procedure calls and non-determinism were also added to meet user needs. The SDL-92 Recommendation [2] was about 10 % larger than the SDL-88 standard [3]. Significant work was also done in the period on methodology, which is a difficult area to handle within the framework of standardisation since many aspects are organisation dependent. The result was issued as methodology guidelines to replace the user guidelines of the Blue book. There is an overview of SDL-92 in [9].

#### 1.4 SDL Combined with ASN.1

After 1993, ITU encouraged publication of Recommendations at any time. Work was progressed rapidly on using SDL in combination with ASN.1 in Z.105 [10].

ASN.1 was widely used for defining data structures on protocol interfaces: that is protocol data units conveyed by messages between systems: in standards and sometimes in real systems.

ASN.1 allows data values to be defined, but does not define any operators between those values. For example, ASN.1 defines the Integer values 1 and 2, but does not define “+” or any way of writing expressions and therefore cannot be used to describe behaviour. SDL can be used to describe structure and behaviour, and in particular can define operators for data. Z.105 brings these two worlds together, and an overview was published in [11].

Z.105 was approved in March 1995 [12], and as far as possible was an extension of SDL-92.

Z.105 allowed data used within SDL processes and in signals to be defined using a subset of the ASN.1 notation. This then also implied that ASN.1 encoding can be used, at least for signals to and from the SDL system. This is an advantage over SDL, because SDL does not define encoding of data.

Another advantage is that ASN.1 has been widely used to define protocol data units conveyed by messages between systems: in standards and sometimes in real systems.

#### 1.5 SDL-96

Apart from the development of Z.105, stability was the major objective in the 1992 to 1996 period, and allowed tools to catch up with the Recommendation.

A common graphical interface between tools was standardized in Z.106 [13], and a new methodology framework document was produced as Supplement 1 to Z.100 [14], based to

some extent on work done (but never published) by ETSI. Some extensions to SDL-92 were agreed to meet user needs, generally relaxing rules of SDL-92 to make it easier to use and some corrections were made to the language definition where it was unclear, ambiguous or inconsistent. These were collected into the 35 pages of Addendum 1 to Z.100 [15] which together with SDL-92 defines what became known as “SDL-96”. The texts of Z.100, Z.106, Addendum 1, and Methodology Supplement were approved for publication in 1996.

SDL-96 was essentially a superset of SDL-92 which in turn was a superset of SDL-88, so that valid SDL-88 was still valid in SDL-96, except in a few obscure cases that were unlikely to arise in practice. SDL-96 can therefore be considered as SDL-92 with some corrections and a few extensions. The extensions simplified SDL by harmonizing concepts and improved the expressive power of SDL. There were no changes to the underlying models.

##### 1.5.1 Harmonized Communication

Remote procedure definitions, remote variable definitions all defined communication primitives, but in SDL-92 remote procedures and remote variables could not be used:

- to show the communication on channels and gates;
- for communication with the environment of the system.

The use of remote procedures and remote variables was harmonized with signals, such that whenever a signal can be mentioned, a **remote** procedure or **remote** variable can also be mentioned (with a few exceptions that do not make sense). If a **remote** procedure or **remote** variable is mentioned on a gate or communication path of a diagram then the diagram need not include an **imported** specification. To use **remote** procedures and variables for communication with the environment, they must be mentioned on a channel connected to the environment.

A related improvement was to allow a signal list identifier (in brackets) in input.

The possible clash of names between signals, **remote** procedures and **remote** variables is resolved by taking first a visible signal with the name. A procedure (or variable) with the same name as a signal must be preceded by **procedure** (or **remote** respectively).

### 1.5.2 External Behaviour

SDL-96 enabled the use of procedures and operators defined outside the SDL-description by adding the keyword **external** at the end of the operator or procedure heading to show that the body of the procedure is defined external to the SDL.

### 1.5.3 Simplified Paths

Channels have names that are used in **output via**, and to connect a path outside a diagram when the inner diagram is drawn separately (that is it is **referenced** – the term “linked” is used in this article). When an inner diagram is the instantiation of a **type**, the outer path is connected to the inner gate name and the outer path name is not needed. SDL-96 allows the names to be dropped when not needed, and similarly signal lists do not have to be repeated on both the outer and inner path.

In SDL-96 the use of channel and signallist names in diagrams can be restricted to just those places where they are needed (that is if they are referred to in some part of the description), or if they serve a useful explanatory function.

### 1.5.4 Paths to Self

In SDL-92 it was not allowed to draw a communication path from one process instance-set back to the same one, or from a block-set and back to itself. Therefore, it was not possible to denote the communication between one member of the set and another. This restriction was removed in SDL-96. So that the construct is not ambiguous, it must be a one-way communication path, but two or more distinct one-way paths are allowed so by this means two-way communication is possible.

### 1.5.5 Agents as Systems

SDL-96 allows a service, process, or block to be considered as a system with the surrounding constructs (that is for a service the process, block and system) implied. This simplifies the specification of simple systems, and makes it easier to consider a larger system as a number of communicating smaller system

### 1.5.6 Extended Use of Packages

The SDL-92 restriction that packages can only be attached to packages or the system diagram was removed. Packages can be attached to any diagram. This has the advantage that the visibility of the items in the package can be restricted to just those diagrams where they are used.

### 1.5.7 State Expression

An additional imperative operator, STATE, was added, which returns the name of the current state as a charstring expression.

### 1.5.8 Nullary Operators

These are operators without arguments. The implicit ordering rules of SDL for ordering literals do not apply to nullary operators, therefore they are useful for defining sorts of data with the **ordering** property. Otherwise, they are the same as data literal values.

### 1.5.9 Common Interchange Format (CIF)

SDL has had both a textual phrase representation (SDL/PR) and a graphical representation (SDL/GR) since 1984. The two forms have a large textual part that is common to both, but graphical constructs also have an equivalent textual representation. Originally, the objective was to provide a form like a programming language to enable SDL to be more easily processed and analysed by computers. At that time processing of graphics was slow and suitable equipment was expensive. SDL/PR was designed to be machine processed, and therefore although it is readable, its grammar is not very “user-friendly”.

SDL/PR found other roles as enabling SDL to be interchanged between tools, and as an intermediate language within tools supporting SDL/GR. However, if SDL is transferred from one tool to another using SDL/PR, the diagrams are often unrecognisable because graphical layout information is lost.

The objective of the CIF is to enable SDL to be transferred between tools and be “recognisably the same”. CIF is an extension of SDL/PR that contains the additional graphical information in comments of SDL/PR.

### 1.5.10 Methodology

This is not part of the Z.100 standard: the 1992 guidelines are in an appendix and are therefore *informative* (as compared to an *Annex* such as Annex F Formal definition which is normative). The methodology associated with Z.100 can only suggest approaches, but SDL can be used validly in many other ways. The work done in the 1992–96 period on methodology is published in Supplement 1 to Z.100 – “SDL+ methodology: use of MSC and SDL (with ASN.1)”.

The SDL+ methodology did not replace the 1992 guidelines, but provides a more detailed framework that can be elaborated for a particular use to develop a formal specification in SDL. It covers to some extent MSC and use of ASN.1, and can be applied from the level of requirement capture. It suggests the use of the OMT object model notation for forming the initial classes. An overview, published in [16], was presented at the SDL’95 Forum.

## 1.6 MSC and UML

The Message Sequence Chart (MSC) language was first standardized in Z.120 in 1992 as MSC-92, although they had been suggested since 1984 as auxiliary diagrams to be used with SDL. MSC tools now exist that are linked to SDL tools.

MSC has continued to evolve, and a variant has appeared as part of the UML set of notations in OMG.

Ivar Jacobson of Rational, who is well known for use cases and promoting UML, participated in the SDL group in the early 1980s and acknowledges the common origins for ideas in his work and in SDL. However, he ceased to participate in the group before the creation of the MSC standard and the development of SDL as it is today. In fact, SDL-2000 and MSC-2000 could easily be taken as UML profiles by OMG.

## 1.7 Recommendations in Force and Availability

The standards in force at the end of 1999 related to SDL were:

- SDL: Z.100 (11/99) [1] and Supplement 1 (10/96); [15];
- SDL combined with ASN.1 modules: Z.105 (11/99); [17];
- Common interchange format: Z.106 [13];
- SDL with embedded ASN.1: Z.107 [18];
- SDL combined with UML: Z.109 [19];
- Use of FDT's: Z.110 (11/99) [20].

The 11/99 standards were available as the English drafts immediately to members of the Study Group and the SDL Forum Society, and in pre-publication form via the ITU subscription service (ITU-T Recommendations Online) mid-December 1999. The published English version may differ in page numbers and minor changes in wording, but the technical content should be the same.

## 2 The SDL-2000 Standard

SDL is a living language, which means it is used frequently with new applications in new areas. If SDL were a natural language, new applications, features and vocabulary would arise, while some language constructs would become unused, unfamiliar or even unknown. Similarly SDL users have new needs, and have innovative ideas for changing SDL. However, SDL is a formal language that needs to be well defined and machine processable, so that all users and tools can

understand and manipulate designs in SDL. Changes to SDL need to be carefully managed. Formally, this was the task of ITU-T Study Group 10 under Q.6/10 in the period 1996 to 2000, but the work was done in close collaboration with the SDL Forum Society.

SDL is quite complex. This means that there was plenty of scope for clarifying the language definition so that it is not misinterpreted, and there were still some ambiguities, inconsistencies or just defects in the language (that is “bugs”) that needed correcting. The study therefore specifically included defect correction. Maintenance also includes the possibility to change the language, and for SDL the rules for maintenance (including the change procedure) were defined in Addendum 1 of Z.100. Similar rules are now part of SDL-2000.

The scope of the study question for SDL-2000 was (phrases copied direct from Q.6/10):

*... corrections, ease of use and ease of maintainability, new uses of SDL – especially in conjunction with other languages, and simplification – both by decommitting non-used features of SDL and by combining similar concepts of the language.*

SDL is mainly used for implementing systems. More focus was therefore given to constructs for using SDL for design and implementation.

## 2.1 SDL Simplification and Maintenance

There were differing opinions on the precise meaning and scope of “simplification” objective for the 1996–2000 study period: it could just be deletion of some language concepts, or it could extend to modelling blocks and processes (and the SDL-92 *services*) all as variants of some building block object. The definition was adopted that change meant “removal of unnecessary restrictions and differences in language concepts and perhaps unnecessary, unused features”. Simplification has encompassed both the above changes.

Z.100 for SDL-2000 consolidates Z.100 (03/93) [2] with Z.100 Addendum 1 (10/96) [15] and many features of Z.105 (03/95) [12] into one Recommendation [1]. This also incorporates the Master List of Changes maintained by the Q.6/10 experts' group according to the rules Recommended in [15].

As noted above, the maintenance rules allowed features to be removed and the following features were deleted:

- Alternative block partitioning (that is the possibility to give two different versions of a block, one where the block is described using processes and one where the block is described using a block substructure);
- Structural graphical macros (that is macros in block diagrams);
- Behavioural graphical macros;
- View/reveal;
- Channel substructure;
- Signal refinement;
- Axiomatic data type definition.
- Handle run time errors (which implies there are language defined errors);
- Compatibility with Z.130 Object Definition Language [21];
- Avoid any conflicts between exceptions in SDL and in other languages (such as IDL/ODL, Java);
- User defined exceptions and causing exceptions;
- Provide a mechanism to enable non-responding **remote** procedures to be trapped.

It was also recognized that there could be some improvement in the handling of timers.

Taking such a step is difficult because there is always a call for backward compatibility. A procedure for deleting features was defined and followed. Essentially, it was possible to remove these features because they were seldom used.

The new mechanism provides the ability to raise an exception, which can have a handler not considered to be part of the usual behaviour. This is an important feature for the use of SDL with the ITU ODL and with CORBA.

The service feature was not deleted from the language but was harmonized with blocks, processes and composite states, so the keyword `service` is no longer used.

### 2.2.2 Textual Algorithms

The mechanism allows algorithms to be written textually within graphical diagrams. Most users prefer SDL/GR most of the time, but in some situations, SDL/GR leads to additional diagrams, unnecessary complexity, unnecessary indirection and lack of conciseness. SDL/PR is not a good alternative, because it is verbose, tool oriented and may still require indirection. Textual algorithms introduce a user-oriented mechanism for combining text with SDL/GR. It was already supported by one tool before the SDL-2000 was approved.

## 2.2 New Features in Z.100

A long list of open items was considered for inclusion in SDL-2000, ranging from trivial items such as allowing the keyword `call` to be optional, to major changes such as the introduction of block identities similar to Pids.

The new features covered by Z.100 for SDL-2000 are:

- Exceptions;
- Textual algorithms;
- New data model;
- Nested packages;
- Mixing blocks and processes;
- Type based creation of processes;
- Typed Pid sorts;
- Interfaces;
- PR/GR harmonization;
- Composite states;
- Object modelling support.

These features are described briefly below, with the exception of object modelling support, which is related to UML and is treated in 2.4.

### 2.2.3 New Data Model

There have been a number of problems associated with the SDL data model based on the algebraic approach using ACT ONE. In addition, the SDL-92 inheritance model for data was not consistent with other types in SDL. The algebraic approach to define new kinds of data has not proved popular with users, and it is considered difficult to use and difficult to implement fully. Most users used only the language-defined kinds of data, and wanted data similar to object oriented programming languages. It was therefore agreed that the data model for SDL could be changed to support new features, to have typing similar to other types in SDL and to support most of the current use of data in SDL-92.

#### 2.2.1 Exceptions

Exceptions were a specific topic of study and a short requirements list was:

The specific rationale for change was:

- It had not been possible to properly incorporate error! in the existing ACT ONE model;

- The algebraic approach for defining new sorts of data was difficult to master and it is difficult to make efficient implementations from ACT ONE;
- The existing model produces some undesirable language rules (such as not permitting a struct field as an in/out parameter).

The new model brings SDL data more in line with the other object/type features of SDL (**block type**, **process type** etc.). It makes data in SDL easier to understand for someone who is familiar with data in a (so-called) object oriented programming language such as C++ or Java.

SDL-92 data models (**newtype** sorts of data in the old terminology; “**value types**” in the new terminology) are still valid in SDL-2000, except where they rely on full implementation of axiomatic descriptions.

**value types** in SDL-2000 are either the language-defined types, or composite **value types** (such as Array, Struct, String ...) constructed using other non-composite or composite types. Operators can still be defined with a **value** type (as is possible in SDL-96), with the body of the operator described in algorithmic SDL or as **external** (as allowed in SDL-96).

The SDL-2000 model introduces “**object types**” for data that can refer to “**value types**” and have polymorphic operators and methods.

Methods introduced with a type are applied to instances of the type using the dot notation familiar from other languages. For example:

```
counter.increment(10) /* counter is a variable
and increment is a method */
```

Although user definition of axioms in the new model is not supported, the built-in data types still have a description that uses axioms. These data types therefore still have the formal description as before. Other data types are constructed from these types and the features of SDL-2000.

## 2.2.4 Nested Packages

Nested packages are allowed. This provides a more flexible packaging mechanism, and makes it easier to mix SDL with some other languages.

## 2.2.5 Mixed Blocks and Processes

The restriction that blocks and processes could not be mixed in one diagram is removed. Many users have requested this change, and there seems to be no good reason for maintaining it, especially without block partitioning.

The harmonization has gone further with agents as a concept that covers the system, blocks and processes. Processes can also be nested within processes, and the service concept is replaced by composite states that are state aggregations (see 2.2.10).

When a process is defined within another process definition, then each instance of the outer process contains a number of instances of the inner process. All these instances are scheduled to run in an alternating way, and the inner process instances can access the data of the outer process instance directly.

When a process definition is directly contained within a block, instances of the process run concurrently, and concurrently with any other instances in the block, including the instance that owns any variables of the block. Processes within a block access block variables by implicit remote procedure calls to the instance for the variables.

## 2.2.6 Type Based Creation

An agent instance may be created from an agent type. In SDL-2000, it is not necessary to have an explicit definition of the agent instances in the case that there is only one set in the context of the creation and there are no initial instances. Also because no instance name is needed in this case, create can be used in another process type which would be impossible otherwise.

## 2.2.7 Typed Pid Sorts

These are similar to the Pid sort, except that the actual Pid value must belong to the correct process type. The benefit is additional security and confidence that a Pid actually identifies a process of a specific process type when used as a destination after to for an output, **import** or a remote procedure call.

## 2.2.8 Interfaces

An interface is a type that contains the definition of a number of signals, remote variables and remote procedures and may be associated with channels, gates, connections or signal sets. Because an interface is a type it can have context parameters. An interface is a scope unit for the contained definitions, which nevertheless are visible outside the interface (like operators defined with a sort of data). An interface can inherit from one or more other interfaces.

## 2.2.9 PR/GR Harmonization

In SDL-92 there are some inconsistencies between the text in SDL/PR and the text in SDL/GR, where the same grammar could be used in both cases (for example punctuation and whether items are optional). These were made consistent wherever possible.

### 2.2.10 Composite States

This introduces a hierarchical state mechanism. One benefit is the ability to hide sub-states within a super-state, so that the details are hidden and it is easy to have an overview of process behaviour. Another benefit is that when a design is elaborated, a state may be split into sub-states without losing the original state.

A composite state can consist of just one set of sub-states.

### 2.3 Incorporation of Z.105 and the Data Model

Use of ASN.1 is strongly related to the data model of SDL. As it was agreed to change the underlying data model of SDL-92, the Z.105 Recommendation had to be updated, because it relied on the algebraic model.

The old Recommendation Z.105 introduced some inconsistencies between SDL with and without ASN.1. These inconsistencies are removed in Z.100 for SDL-2000. One consequence is that SDL-2000 is case sensitive, which also has some benefits when using SDL with other languages (for example, SDL is often translated by tools for implementation into C which is case sensitive).

Consideration was given to incorporating Z.105 completely into Z.100, but the view was taken that Z.105 should remain a separate document describing how the ASN.1 syntax can be used with SDL. However, it was also agreed that some functionality of Z.105 should be incorporated into Z.100 as SDL concepts so that choice, optional and default fields are now part of SDL.

Z.105 (11/99) is a mapping of ASN.1 to Z.100 (11/99) features, so that a data type specified in ASN.1 has an equivalent specification according to Z.100 (11/99). This is not the case between SDL-92 and Z.105 (03/95): for example, SDL-92 allows “[” and “]” in names whereas Z.105 (03/95) does not, and CHOICE required some change to the SDL model.

Another change has been to separate the use of ASN.1 modules with SDL, and the definition of ASN.1 embedded in SDL. The use of ASN.1 modules as SDL packages is still within the scope of Z.105 (11/99)[17], but ASN.1 embedded within SDL is now in the Z.107 (11/99) Recommendation [18].

### 2.4 SDL with UML

Partly because of some common heritage (see 1.6), it was always the case that UML and the ITU set of languages had similarities and complemented each other. Even before UML existed, it was recognised [16] that object mod-

elling was needed in conjunction with SDL. The suggested technique (in [14]) was OMT, the basis of object modelling in UML. There are small differences between MSC and UML sequence diagrams.

UML defines language meaning allowing some semantic variations, and leaves open the notation, so that various different notations may be used. The ITU standard for SDL defines both the notation and the meaning. UML allows a set of notations to be considered together. There seems to be no reason why SDL cannot be considered to define state machines within UML.

On the other hand, the Z series Recommendations had no language for class diagrams and there was no reason not to adopt the widely accepted UML notation for this. By making this notation part of SDL, it binds the object models closely to SDL and also makes the notation readily available to SDL users not otherwise using UML.

The standard Z.109 goes one step further and details the use of SDL combined with UML. It defines a UML profile for SDL, and in that way defines how UML can be mapped to SDL. Of course, this mapping only applies for that part of UML that is relevant to SDL [9]. The sequence diagram part of UML would map to MSC.

One change to SDL has been to allow the reference symbols that link to SDL diagrams (such as block type, process type, and procedure) to be shown as generic class symbols: a three partition box for the object identity, its attributes and behaviour. These can be used wherever a linking symbol is valid. A further change is that multiple occurrences of a these linking symbols are allowed in the same context, provided they are consistent. This enables an object model at one level to be presented as a number of pages of an SDL diagram showing different associations on different pages.

SDL is also extended to show more relationships between objects. The communication paths and creation relationships have always been part of SDL. SDL-2000 has named association symbols (lines with various ends such as arrows and diamonds) with text at each end describing a relationship between two objects. The usual object models of UML are now part of the SDL syntax. The SDL standard [1] puts constraints on the syntax and the use of names, but does not further define the meaning of the relationships. The meaning otherwise defined by the SDL standard should be changed by the inclusion or deletion of associations: they do not have any impact on the SDL defined semantics.

### 3 Open Issues for Further Study

When the first version of SDL was produced, some of the originators thought that the language was finished and no changes would be needed. In retrospect, it is easy to understand why this was not the case, and why there has been user demand for more features and new paradigms in the language. On the other hand, the basic graphical notation and the extended finite state machine (EFSM) paradigm at the core of SDL have stood the test of time. Even in 1980 it was difficult to foresee the relative low cost and high performance of commodity computers available in 2000, that can run complex SDL tools. It is therefore difficult to predict precisely how SDL will develop through the first century of the third millennium.

The best guess for the future is that SDL will continue to exist with EFSM at its core. The language is still evolving and use is growing.

The ITU-T study programme in the short term is addressing for 2001 the open issues of revised formal definition of SDL-2000, revised Common Interchange Format (CIF) for SDL-2000 and a methodology update. There is an ongoing study of time and performance issues that could lead to changes to SDL, or some way of linking requirements such as time deadlines to SDL models. There needs to be a binding of MSC data to SDL, and there is a need to be able to define encoding of SDL data on interfaces.

One known area of difficulty is the composition of services on a mix and match basis during system implementation. For example, with three telecommunication services (A, B and C), it should be possible to define the base system, the service A, the service B and the service C. It should then be possible to generate an implementation for any combination. That is: base, base+A, base+B, base+C, base+A+B, base+A+C, base+B+C, base+A+B+C. It is a methodology issue to define how this can be done (if it can) in SDL-2000. If it cannot be done, it is an unresolved language issue.

The previous examples are known user needs, but even in the relatively short period of 1996 to 1999 the focus, general environment and expectation changed so that the issue of object modelling changed from being considered as an auxiliary notation, to requiring integration into SDL. It is therefore difficult to predict what SDL-2004 may or may not include. It is only certain that it should be user driven: it is proposed that language development is carried out by the SDL Forum Society ([www.sdl-forum.org](http://www.sdl-forum.org)) and the results submitted to ITU-T for approval and publication. It is easy for individuals or organiza-

tions to participate in the SDL Forum Society, which has its objectives focussed on SDL/MSD and can be flexible. The ITU-T by comparison has authority and the infrastructure for publishing Recommendations and maintaining them over a long period. There is the opportunity to utilise the strengths of both organisations.

Other possible issues to be tackled for the future are the development of IP protocols in SDL and the adoption of MSD with SDL as a UML profile. IP in SDL is likely, because of the telecommunications over IP projects that are in progress, and MSD-2000 with SDL-2000 technically meets the UML profile requirements, whether officially recognised or not.

#### The SDL Forum Society

The SDL Forum Society is a non-profit making organization formed in order to promote the Specification and Description Language (SDL) and Message Sequence Chart (MSC).

The Society has existed since June 1990 and, as well as promoting the knowledge and usage of MSC/SDL and providing information on the development and use of SDL/MSD, one of its major functions is to promote and organise the SDL Forum which takes place once every two years.

The SDL Forum provides an opportunity for experts, users, toolmakers and even some critics of MSC and SDL to meet, engage in useful discussion and socialise. It is called a 'Forum' because it is a 'meeting place' for the exchange of ideas and a chance for people to meet others who are involved in the use of MSC and SDL. The plans for the Tenth SDL Forum are already under way, and it will be held in Copenhagen in June 2001.

The SDL Forum Society also supports the SDL and MSD Workshop – SAM – and this addresses three topics: language issues; the relation to other Languages or Techniques; Design, Methodology and Applications. Those involved in the workshop include researchers, users of the languages, and members of standardization bodies. The event is less formal than the SDL Forum. It enables intensive discussion of the languages, and evolution of ideas on the future development and application of SDL and MSD.

When joining the Society one of the benefits is discounted access to either the Forum or the Workshops, and discounted copies of the proceedings. There are also free updates on ITU-T activities related to MSC/SDL including access to ITU-T documents on studies in progress. The Society has recently been given permission from the ITU-T to distribute a version of the SDL-2000 and MSD-2000 Recommendations via its web site.

Other activities have included a Grant for student work on related research, and specific support for the continuing evolution of SDL and MSD, changing to meet current and future needs and technologies.

If you would like any further information on the Society, including information on how to join and details of the inexpensive membership fees, visit the web site at <http://www.sdl-forum.org>.

Some SDL models are available for sale as components and this market may increase. One factor is that the major SDL tool supplier has about one third of the world real time development market (in all fields, not just telecommunications). SDL-2000 should give excellent support for producing components. With the continued growth of the telecommunications market and increase in SDL use in other areas, an SDL component market seems likely.

## References

- 1 ITU-T. *Specification and Description Language (SDL)*. Geneva, ITU-T, 2000. (Z.100 (11/99).)
- 2 ITU-T. *CCITT Specification and Description Language (SDL)*. Geneva, ITU-T, 1994. (Z.100 (03/93).)
- 3 ITU-T. *Functional Specification and Description Language (SDL)*. Geneva, ITU-T, 1989. (Z.100(1989).)
- 4 ISO. *Information Processing Systems – Open Systems Interconnection – LOTOS – a Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. Geneva, ISO. (ISO/IEC 8807.)
- 5 *Introduction to SDL-88*. (2000, September 06) [online] – URL: <http://www.sdl-forum.org/sdl88tutorial/>
- 6 Bræk, R. SDL Basics. *Computer Networks and ISDN Systems*, 28, 1585–1602, 1996.
- 7 Færgemand, O, Olsen, A. Introduction to SDL-92. *Computer Networks and ISDN Systems*, 26, 1143–1167, 1994.
- 8 Sarma, A. An introduction to SDL-92. *Computer Networks and ISDN Systems*, 28, 1603–1615, 1996.
- 9 Møller-Pedersen, B. SDL-92 as an object oriented notation. *Telektronikk*, 89 (2/3), 71–83, 1993.
- 10 ITU-T. *Data networks and open system communications – OSI networking and system aspects – Abstract Syntax Notation One (ASN.1)*. Geneva, ITU-T. (X.680-681.)
- 11 Verhaard, L. An introduction to Z.105. *Computer Networks and ISDN Systems*, 28, 1617–1628, 1996.
- 12 ITU-T. *SDL Combined with ASN.1 (SDL/ASN.1)*. Geneva, ITU-T, 1995. (Z.105 (03/95).)
- 13 ITU-T. *Common Interchange Format for SDL*. Geneva, ITU-T, 1997. (Z.106 (10/96).)
- 14 ITU-T. *Use of MSC and SDL (with ASN.1)*. Geneva, ITU-T, 1997. (Supplement 1 (04/96) to Rec. Z.100 (03/93) SDL+ Methodology.)
- 15 ITU-T. *Corrections to Recommendation Z.100 (03/93)*. Geneva, ITU-T, 1997. (Z.100 Addendum 1 (10/96).)
- 16 Reed, R. Methodology for Real Time Systems. *Computer Networks and ISDN Systems*, 28, 1685–1701, 1996.
- 17 ITU-T. *SDL combined with ASN.1 modules*. Geneva, ITU-T, 2000. (Z.105 (11/99).)
- 18 ITU-T. *SDL with embedded ASN.1*. Geneva, ITU-T, 2000. (Z.107 (11/99).)
- 19 ITU-T. *SDL combined with UML*. Geneva, ITU-T, 2000. (Z.109 (11/99).)
- 20 ITU-T. *Criteria for the use of formal description techniques by ITU-T*. Geneva, ITU-T, 2000. (Z.110 (11/99).)
- 21 ITU-T. *Object definition Language*. Geneva, ITU-T, 1999. (Z.130 (02/99).)