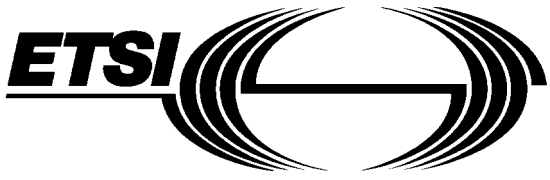
TD <>

DEG/MTS-00053 V0.2.0 (1998-07)

**Methods for Testing and Specification (MTS);
Use of SDL in European Telecommunication Standards;
Guidelines for facilitating validation and the development of conformance tests**



European Telecommunications Standards Institute

Reference
DEG/MTS-00053

Keywords
<keyword[, keyword]>

ETSI Secretariat

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

X.400
c= fr; a=atlas; p=etsi; s=secretariat

Internet
secretariat@etsi.fr
http://www.etsi.fr

Contents

Intellectual Property Rights 8
Foreword 8
Introduction 8
1 Scope 9
2 References 9
3 Definitions and abbreviations 10
3.1 Definitions 10
3.2 Abbreviations 11
4 Principles and general guidelines 11
5 SDL in European Telecommunication Standards 11
5.1 Introduction 11
5.2 Validation of specifications 11
5.2.1 Formal validation 12
5.3 Testing of telecommunication products 12
5.3.1 Conformance testing 12
6 Normative interfaces and requirements 13
6.1 Normative interfaces 13
6.2 Normative requirements 14
7 Specification and description language concepts 16
7.1 Introduction 16
7.2 Basic SDL 17
7.2.1 General rules 17
7.2.1.1 Lexical rules 17
7.2.1.2 Visibility rules, names and identifiers 17
7.2.1.3 Informal text 17
7.2.1.4 Drawing rules 17
7.2.1.5 Comment 18
7.2.1.6 Text extension 18
7.2.1.7 Text symbol 18
7.2.2 Basic data concepts 18
7.2.2.1 Data type definitions 18
7.2.2.2 Values and literals 18
7.2.2.3 Expressions 18
7.2.3 System structure 18
7.2.3.1 Organisation of SDL specifications 18
7.2.3.1.1 Package 18
7.2.3.1.2 Referenced definition 18
7.2.3.2 System 18
7.2.3.3 Block 18
7.2.3.4 Process 19
7.2.3.5 Service 19
7.2.3.6 Procedure 19
7.2.4 Communication 19
7.2.4.1 Channel 19
7.2.4.2 Signal route 19
7.2.4.3 Connection 19
7.2.4.4 Signal 19
7.2.4.5 Signal list definition 19
7.2.5 Behaviour 19
7.2.5.1 Variables 19
7.2.5.1.1 Variable definition 19

7.2.5.1.2 View definition 20
7.2.5.2 Start 20
7.2.5.3 State 20
7.2.5.4 Input (and invalid stimuli) 20
7.2.5.5 Save 20
7.2.5.6 Spontaneous transition 20
7.2.5.7 Label 20
7.2.5.8 Nextstate 20
7.2.5.9 Join 21
7.2.5.10 Stop 21
7.2.5.11 Return 21
7.2.6 Action 21
7.2.6.1 Task 21
7.2.6.2 Create 21
7.2.6.3 Procedure call 21
7.2.6.4 Output 21
7.2.6.5 Decision 21
7.2.7 Timer 21
7.2.8 Internal input and output 21
7.3 Structural Decomposition Concepts in SDL 22
7.3.1 Partitioning 22
7.3.1.1 Block partitioning 22
7.3.1.2 Channel partitioning 22
7.3.2 Refinement 22
7.4 Additional Concepts of Basic SDL 22
7.4.1 Macro 22
7.4.1.1 Macro, graphical behaviour 22
7.4.1.2 Macro, structural 22
7.4.1.3 Macro, textual 22
7.4.2 Generic system definition 23
7.4.2.1 External synonym 23
7.4.2.2 Simple expression 23
7.4.2.3 Optional definition (The select construct) 23
7.4.2.4 Optional transition string 23
7.4.3 Asterisk state 23
7.4.4 Multiple appearance of state 23
7.4.5 Asterisk input 23
7.4.6 Asterisk save 23
7.4.7 Implicit transition 24
7.4.8 Dash nextstate 24
7.4.9 Priority Input 25
7.4.10 Continuous signal 25
7.4.11 Enabling condition 25
7.4.12 Imported and Exported value 25
7.4.13 Remote procedures 25
7.5 Data in SDL 25
7.5.1 The data kernel language 25
7.5.1.1 Data type definitions 26
7.5.1.2 Literals and parameterised operators 26
7.5.1.3 Axioms and conditional equations 26
7.5.2 SDL expressions and data types 26
7.5.2.1 Special operators 26
7.5.2.2 Character string, Bit_String and Hex_String literals 27
7.5.2.3 Predefined data, equality and noequality 27
7.5.2.4 Boolean axioms, conditional terms and error! 28
7.5.2.5 Ordering 28
7.5.2.6 Syntypes and range conditions 28
7.5.2.7 Structures, SEQUENCE, SET and CHOICE 28
7.5.2.8 Inheritance 28
7.5.2.9 Generators 29
7.5.2.10 Synonyms 29

7.5.2.11 Name class literals and literal mapping 29
7.5.2.12 Operator definitions and operator applications 29
7.5.2.13 Indexed primary and indexed variable 29
7.5.2.14 Field primary and field variable 29
7.5.2.15 Structure primary (and array value) 30
7.5.2.16 Conditional expression 30
7.5.2.17 Variable access 30
7.5.2.18 Assignment statement 30
7.5.2.18.1 Default initialization 30
7.5.2.19 Imperative operators 30
7.5.2.19.1 Now expression 30
7.5.2.19.2 Import expression 30
7.5.2.19.3 Pid expression 30
7.5.2.19.4 View expression 30
7.5.2.19.5 Timer active expression 30
7.5.2.19.6 Anyvalue expression 31
7.5.2.20 Value returning procedure call 31
7.5.2.21 External data 31
7.6 Structural Typing Concepts in SDL 31
7.6.1 Types, instances, and gates 31
7.6.1.1 Type definitions 31
7.6.1.1.1 System type 31
7.6.1.1.2 Block type 31
7.6.1.1.3 Process type 32
7.6.1.1.4 Service type 32
7.6.1.2 Type expression 32
7.6.1.3 Definitions based on types 32
7.6.1.3.1 System definition based on system type 32
7.6.1.3.2 Block definition based on block type 32
7.6.1.3.3 Process definition based on process type 32
7.6.1.3.4 Service definition based on service type 32
7.6.1.4 Gate 32
7.6.2 Context parameter 32
7.6.3 Specialization 33
7.6.3.1 Adding properties 33
7.6.3.2 Virtual type 33
7.6.3.3 Virtual transition/save 33

8 Concepts in MSC 33
8.1 Introduction to MSC 33
8.2 General rules 34
8.2.1 Lexical rules 34
8.2.2 Visibility and naming rules 34
8.2.3 Comment 34
8.2.4 Drawing rules 34
8.2.5 Paging of MSCs 35
8.3 Message Sequence Chart documents 35
8.3.1 Message Sequence Chart 35
8.3.2 Instance 35
8.3.3 Message 35
8.3.3.1 Message overtaking 35
8.3.3.2 Incomplete messages 35
8.3.3.3 Correspondence between SDL and MSC behaviour 35
8.3.4 Environment and gates 36
8.3.5 General ordering 36
8.3.6 Condition 36
8.3.7 Timer 37
8.3.8 Action 37
8.3.9 Instance creation 37
8.3.10 Instance stop 37
8.4 Structural concepts 37
8.4.1 Coregion 37

8.4.2 Instance decomposition 37
8.4.3 Inline expression 37
8.4.4 MSC reference 37
8.4.5 High-level MSC (HMSC) 37

Annex A: Summary of use of SDL and MSC in ETSI Standards 38
A.1 Selection of SDL concepts 38
A.2 Selection of MSC concepts 44
A.3 List of supplementary guidelines 46

History 48

Intellectual Property Rights

This clause is always the first unnumbered clause.

If you have received any information concerning an essential IPR related to this document please indicate the details here.

Foreword

This clause is always the second unnumbered clause.

To be drafted by the ETSI secretariat.

Introduction

This clause is optional. If it exists, it is always the third unnumbered clause.

1 Scope

This ETSI Guide (EG) specifies a set of guidelines for the use of the Specification and Description Language (SDL) and Message Sequence Charts (MSC) in ETSI standards that specify services, protocols or other forms of behaviour. Furthermore, brief guidelines for the use of the Abstract Syntax Notation One (ASN.1) for the definition of data structures in combination with SDL are included in this EG.

The purpose of the rules and guidelines is to assist rapporteurs to produce standards which it is possible to validate using automatic tools and which contain requirements expressed in a way that facilitates the conformance testing of products that are claimed to implement the standard.

SDL is defined in ITU-T Recommendation Z.100 [1], SDL combined with ASN.1 is specified in ITU-T Z.105 [2] and the use of MSCs is defined in ITU-T Recommendation Z.120 [3]. ASN.1 is defined in ITU-T Recommendations X.680 [4], X.681 [5], X.682 [6] and X.683 [7].

The technical quality criteria specified in EG 201 014[12] are relevant to all standards but the need for "clarity", "consistency" and the "correct use of formalisms" are particularly important in the facilitation of validation and the derivation of conformance tests. Consequently, this guide uses these three principles as the basis for the development of its guidelines.

This guide is not a tutorial in the use of SDL and it does not offer guidelines for the validation of a standard or the production of a conformance test suite. ETSI publications that cover these subjects are:

- the Handbook to the validation methodology for standards using SDL, EG 201 015 [13];
- the Handbook for SDL, ASN.1 and MSC development, ETR 298 [11];
- the standard for Protocol and profile conformance testing specifications, ETS 300 406 [9];

These documents should all be read in conjunction with this EG.

SDL is a large and complex language containing many features and facilities which are ideally suited to use in protocol standards and some which are not so well suited. This guide identifies those constructs which a rapporteur can safely use in a standard to facilitate the validation of a specification and the development of conformance tests. It also identifies a number of constructs that should be avoided in standards.

2 References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ITU-T Recommendation Z.100 (1993): "Specification and description language (SDL)".

[2] ITU-T Recommendation Z.105 (1994): "SDL combined with ASN.1 (SDL/ASN.1)".

[3] ITU-T Recommendation Z.120 (1993): "Messages sequence charts".

[4] ITU-T Recommendations X.680 (1994): "Information technology - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[5] ITU-T Recommendations X.681 (1994): "Information technology - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1): Information object specification".

[6] ITU-T Recommendations X.682 (1994): "Information technology - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1): Constraints specification".

[7] ITU-T Recommendations X.683 (1994): "Information technology - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications".

[8] ISO 9646-1 (1992): "Information technology - Open Systems Interconnection-Conformance testing methodology and framework - General concepts".

[9] ETS 300 406 (1995): "Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology".

[10] ETR184 (1995): "Methods for Testing and Specification (MTS); Overview of validation techniques for European Telecommunication Standards (ETSs) containing SDL".

[11] ETR298 (1996): "Methods for Testing and Specification (MTS); Specification of protocols and services; Handbook for SDL, ASN.1 and MSC development".

[12] EG 201 014 (1997): "Methods for Testing and Specification (MTS); ETSI Standards-making; Technical quality criteria for telecommunications standards".

[13] EG 201 015 (1997): "Methods for Testing and Specification (MTS); Specification of protocols and Services; Validation methodology for standards using SDL; Handbook".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following definitions apply:

abstract data type: the definition of data in terms of abstract properties rather than in terms of a concrete implementation. An abstract data type defines a collection of data types (that is Z.100 sorts), a set of operators which are applied to the values of these data types and a set of rules defining the behaviour when the operators are applied to the values. (see Z.100 [1] 2.3.1, 5.1, Annex C)

conformance requirement: the description of a characteristic in a standard with which a product implementing that standard is expected to conform.

conformance testing: the process of establishing the extent to which an Implementation Under Test (IUT) satisfies both static and dynamic conformance requirements, consistent with the capabilities stated in the implementation conformance statement (ISO 9646-1 [8], subsections 3.4.10, and 3.5.6).

data type: a set of data values with common characteristics (equivalent to the Z.100 term sort).

NOTE: When preceded by the word "abstract" then data type is always considered as part of the term "abstract data type" and not as the term "data type".

implementation conformance statement: a document supplied by the manufacturer of a product that defines which standards are claimed to be implemented and which implementation options in the standards are supported.

implementation option: a statement in a standard that may or may not be supported in an implementation.

normative interface: a physical or software interface of a product on which requirements are imposed by a standard.

state space: the collection of all states of a system that can be reached from the initial state.

validation: the process, with associated methods, procedures and tools, by which an evaluation is made that a standard can be fully implemented, conforms to rules for standards, satisfies the purpose expressed in the record of

requirements on which the standard is based and that an implementation that conforms to the standard has the functionality expressed in the record of requirements on which the standard is based.

validation model: a detailed version of a specification, possibly including parts of its environment, that is used to perform formal validation.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ASN.1	Abstract Syntax Notation no. 1
HMSC	High-level message Sequence Chart
ICS	Implementation Conformance Statement
ISDN	Integrated Services Digital Network
MSC	Message Sequence Chart
PCO	Point of Control and Observation
SDL	Specification and Description Language

4 Principles and general guidelines

The use of SDL within a telecommunications standard is not the same as its use in the design of a product. Whereas a product design needs to specify every detail of the operation a system, a standard should specify only the minimum requirements for interoperability. This means that an SDL specification in a standard may require modification in order to produce a validation model that is complete enough to be executable.

The purpose of a protocol standard is to specify the observable (testable) behaviour at the external, normative interfaces of the system. In general terms, this means the relationships between stimuli and their corresponding responses. Thus, ***the specific behaviour described using SDL in a standard is indicative only of the relationship between incoming and outgoing signals and should not be treated as binding at the language or structure level***. The only strict requirements on the structure of an implementation of the standard are the relationships that are specified between functional entities linked by normative interfaces.

5 SDL in European Telecommunication Standards

5.1 Introduction

SDL diagrams should be presented in the main body of an ETSI standard together with the textual description, tables and figures. It is possible for either the SDL or the text to be considered as the normative specification within a standard with the other providing supporting information. Whichever one is chosen, ***a standard should make it clear whether the SDL specification is normative or informative***.

It is impossible to avoid the duplication of information if both text and SDL are used to describe the normative requirements for a protocol or service, particularly if the text is normative with supporting SDL. This is quite acceptable as the combination of textual and graphical presentations can make the specification easier to understand.

However, ***if the normative description of a protocol is the SDL, the duplication of information should be limited by using the text only to bring together requirements that are distributed over several SDL diagrams***.

5.2 Validation of specifications

EG 201 015 [13] defines validation as follows:

> "The process, with associated methods, procedures and tools, by which an evaluation is made that a standard can be fully implemented, conforms to rules for standards, satisfies the purpose expressed in the record of requirements on which the standard is based and that an implementation that conforms to the standard has the functionality expressed in the record of requirements on which the standard is based.".

This is a very broad definition which covers validation approaches from structured visual inspection of the text up to field trials of telecommunications products. The overall validation process and a range of validation techniques are described in ETR184 [10]

One of the most frequently used techniques for early validation of a standard is a detailed review of its contents performed by experts. ETR 184 [10] and EG 201 015 [13] have identified formal validation as an important means of further improving the quality of protocol and service standards.

5.2.1 Formal validation

Formal validation of a standard includes the checking of both the syntactic and semantic correctness of the specification and establishing that the known requirements of the specified system are clearly and unambiguously expressed within the standard.

The main validation techniques implemented in automatic tools are:

- interactive simulation;
- "exhaustive" state space exploration;
- random state space exploration.

All these techniques depend on the specification being executable by computer. It is, therefore, essential that the SDL specifications included in standards should comply with the rules of syntax and semantics described for the language in ITU-T Recommendation Z.100 [1]. Before such a specification can be formally validated it will also be necessary to specify:

- which implementation options have been selected;
- those parts of the system that are out of the scope of the standard but which are necessary to complete the specification such that it is executable;

5.3 Testing of telecommunication products

5.3.1 Conformance testing

The purpose of standardizing telecommunication systems, services, protocols and interfaces is to enable the inter-working of similar or associated products made by different manufacturers. Testing the conformance of a product to a standard is considered to be essential in ensuring that the product is able to interoperate with other products that implement the same standard or a complementary one.

A conformance test consists of two parts (ISO 9646-1 [8]):

- the static conformance review:
  - checking whether the choices between the implementation options that the manufacturer claims to have implemented is a combination permitted by the standard.
- the dynamic conformance test:
  - execution of test cases to determine whether the product has implemented the standard correctly.

The purpose of a conformance test suite is to check whether a product conforms to the standard. Each test case within a test suite is related to one conformance requirement of the standard.

In order to facilitate the process of test suite development, an ETSI standard should be able to identify:

- the conformance requirements in the standard;
- the interfaces of the product type that can be accessed for test execution;
- how the requirements can be tested using the available interfaces in a product that claims to implement the standard.

Following the guidelines in this EG when writing an SDL specification in a standard should ensure that these criteria are met.

6 Normative interfaces and requirements

6.1 Normative interfaces

The main goal of telecommunication standards is to identify normative interfaces and to define requirements on them such that products that implement the standards (but which may be built by different manufacturers), can be interconnected to jointly provide a telecommunication service.

An example of a normative interface in ISDN is the user-network interface (the S reference point).

A normative interface should not be confused with a Point of Control and Observation (PCO) which is a concept in conformance testing (ISO 9646-1 [8]). Normative interfaces can coincide with PCOs, but a PCO is not necessarily a normative interface.

A standard specifying a communications service or protocol normally defines only a subset of the messages that may pass across a normative interface. In an SDL model of such a service or protocol, these message routes are referred to as communications paths and these would be considered to be normative where they are coincident with normative interfaces.

In an SDL specification, ***normative communications paths should be marked with the comment "Normative".***

The example in Figure 1 shows a simple system where the channel between the "Terminal" block and the "Network" block is identified as a normative communications path.

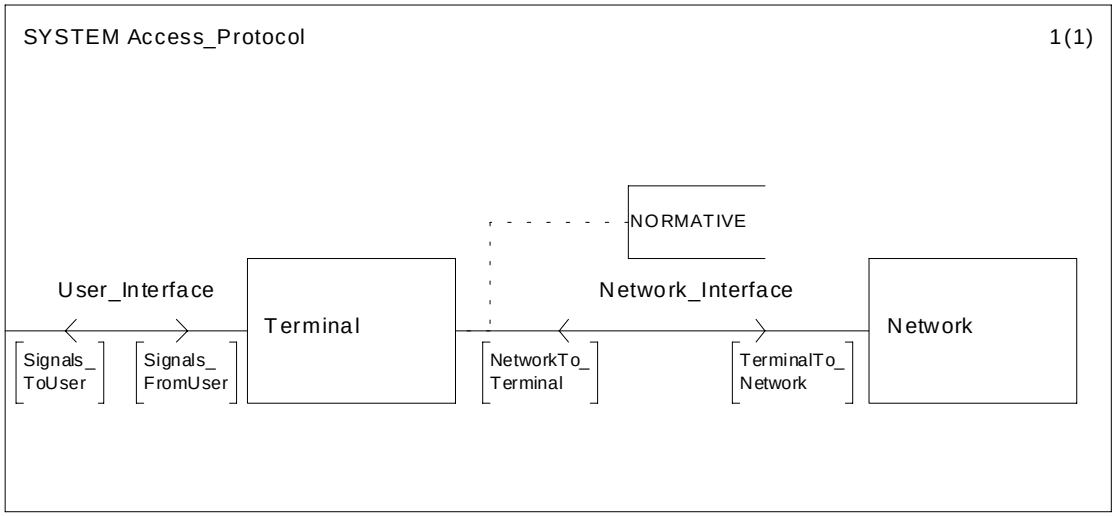


Figure 1: A system diagram with a channel marked "Normative"

NOTE: Some SDL tools do not allow comments to be attached to communications channels directly. The effect can only be achieved by attaching the comment to one of the blocks and then dragging the end of the comment line so that it appears to be attached to the channel. As an alternative, it is also possible to append the comment */* normative */* to the channel name

Figure 2 shows an example where all channels between blocks are marked as normative communications paths while the channels to the environment are considered non-normative, i.e. no requirements are imposed on them.

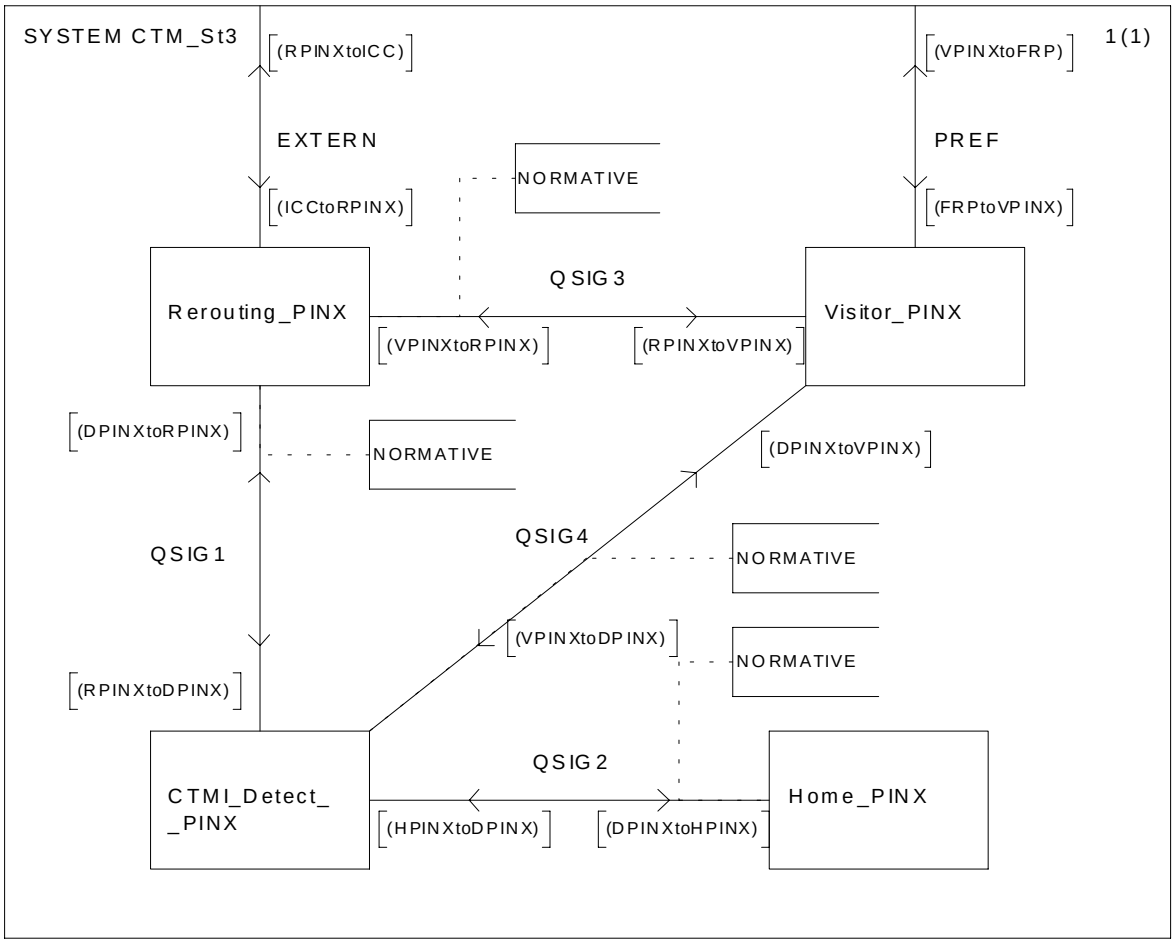


Figure 2: A system diagram with several channels that are marked as normative

The examples shown in Figure 1 and Figure 2 have used SDL channels to model the message paths at normative interfaces. SDL allows processes to be defined in isolation but with a block and a system definition implied. Such a process definition has an implied signal route to the boundary of the implied block and a delaying channel connecting the implied block to the boundary (environment) of the implied SDL system. If this channel is intended to be a normative communications path, it cannot easily be marked as normative. ***Processes should be defined within an explicit block and system.*** In those cases where this is not desirable an extended comment should be added to identify which of the implied communications paths are to be considered as normative.

The specification of process types (or any other SDL object types) permits systems to be defined using a number of discrete building blocks, each of which might be the subject of a separate standard. However, unlike processes, process types cannot be specified in isolation and should always be defined within a package.

6.2 Normative requirements

It is important to make a distinction between the terms “Normative Communications Path” and “Normative Requirements”. The former is a simple identification of the fact that a particular communications path carries only protocol messages that are the subject of the specific standard or another related standard. The latter refers to the structure and contents of messages passed across a normative communications path and to aspects of behaviour which should be visible in a product implementing the standard. This behaviour includes the temporal relationships that must exist between messages and the criteria to be established before a particular message is sent.

For each normative communications path there will be a list of messages that can be exchanged in each direction. ***Messages carried across normative communications paths should be specified in SDL signal definitions and marked as normative requirements.***

SDL Signal definitions specify the data types of message parameters. The normative status of a message should extend to its parameters. ***All data types specified in the parameter lists of normative signals should be defined using ASN.1 or SDL data formalisms and marked as normative requirements themselves.***

The definitions of signals and data types needed for internal messages or for communication across non-normative interfaces should be separated from normative definitions.

The following mechanism for marking is suggested. The SDL comment /*NORMATIVE REQUIREMENT*/ is inserted immediately before a normative definition and applies to all definitions that follow it in the same text (usually within a text box) until either the end of the text (end of the text box) is reached, or there is a marking /*INFORMATIVE*/.

An example of signal and data specifications marked as normative requirement is given in Figure 3.

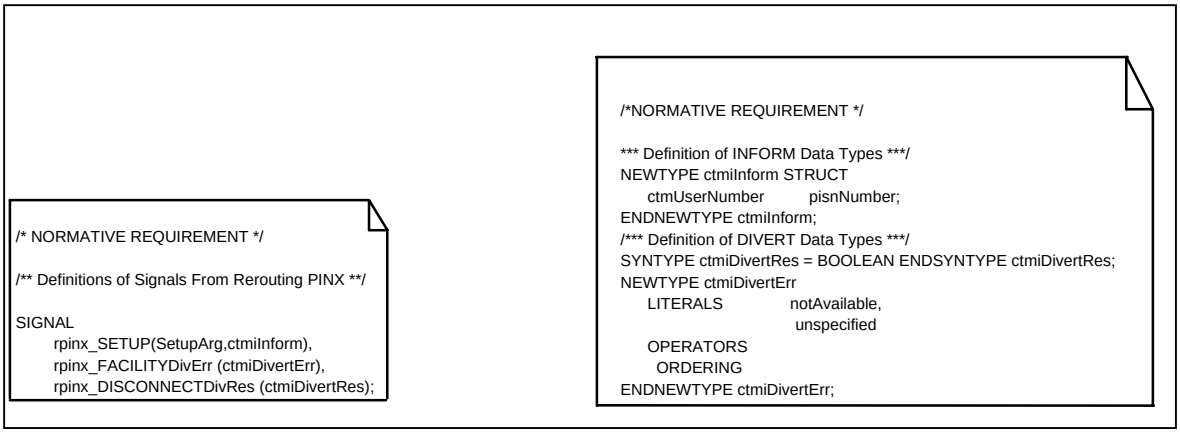


Figure 3: Normative requirements imposed by signal and data definitions in SDL diagrams

In this particular example, the data definitions are given in SDL data formalism. The alternative is to define data using ASN.1 as described in ITU-T Recommendations Z.105 [2] and X.680 [4]

The definition of messages and their parameters is not sufficient to completely define a normative communications path. The sequences of messages expected at the normative interface also need to be specified. These sequences can be defined by the behaviour specified in SDL process graphs that send and receive messages across the normative communications path.

The sequencing of messages can also be expressed very well using MSCs without any associated SDL but there are some problems and limitations in this approach, thus:

- the number of MSCs required for such descriptions is usually large;
- while individual MSCs are readable and easy to understand, the collection of charts that is needed to describe a set of possible sequences is considerably less easy to read;
- for all practical purposes it is impossible to specify a compete set of MSCs that fully describes the overall behaviour of a protocol.

MSCs should be used to define the sequence of protocol messages required for normal operation and the sequences expected in the most significant exceptional cases. MSCs are useful in the specification of signalling protocols and can be used for the following purposes:

- as a starting point for developing SDL behaviour descriptions;
- for the validation of SDL specifications;
- for guiding the selection of test purposes;
- for the derivation or generation of test cases;
- as an aid to understanding.

The complete behaviour in terms of all observable message sequences should be defined using SDL behaviour descriptions. SDL descriptions are likely to duplicate the behaviour specified in MSCs but this kind of redundancy is

encouraged. However, ***the MSC description of a sequence of messages should be consistent with the SDL description of the same sequence***.

In an SDL specification, behaviour is described by the processes contained within SDL blocks. These blocks should be regarded as normative only in terms of the observable effects at their normative interfaces. Their division into processes and procedures should not be considered to be normative.

7 Specification and description language concepts

7.1 Introduction

ITU-T Recommendation Z.100 [1]specifies a wide range of language concepts for SDL. Many of these concepts can be used effectively in telecommunications protocol standards. There are others, however, which should be avoided or, at least, used with great care.

The concepts in this clause are presented in the same order as Z.100. but the numbering does not follow precisely because:

- some clauses (such as introductory sections) have been omitted;
- in some cases several sub-clauses in Z.100 are treated in one clause in this document;
- where necessary extra clauses have been added for concepts in Z.105;
- the use of some of Z.100 concepts are described together with another concept where they are more appropriate (for example the use of *internal input* is described under *input*);
- some additional clauses have been added (for example, for *macro*).

It is expected that SDL is used with ASN.1, and that ITU-T Recommendation Z.105 [2] (which is based on Z.100) is used. The relevant subclauses in this document also include guidelines on the use of Z.105.

The following subclauses identify the SDL concepts used in Z.100 and provide a classification and justification for each, as follows:

- Use freely
    - the feature is without any negative impact on validation and testing and may have positive benefits. It can be used wherever applicable but should always conform to the rules of SDL syntax and semantics;
- Use with care
    - use of the feature may make validation and testing more difficult;
- Not recommended
    - use of the feature usually makes validation and testing difficult;
- Do not use
    - use of the feature is likely to produce a model that cannot be validated or tested.

Particular attention is paid to those concepts that are classified as "use with care" and "not recommended" to ensure that the criteria for using or avoiding the concepts are clearly expressed.

7.2 Basic SDL

7.2.1 General rules

7.2.1.1 Lexical rules

ASN.1 does not permit the use of the characters '{', '}', '[', ']', '|' (vertical bar) and space in names although they are permitted by the lexical rules of SDL. These characters should not be included in any names used within the SDL in a standard.

SDL does not support the use of a hyphen in names and, thus, names with hyphens should be avoided in ASN.1 modules.

NOTE: Z.105 specifies that hyphens in ASN.1 names should be converted to underscores when imported into SDL specifications.

7.2.1.2 Visibility rules, names and identifiers

There are no specific recommendations on the use of SDL visibility rules, names and identifiers

7.2.1.3 Informal text

The syntax of SDL permits informal text to appear in both task symbols and decisions but in neither case does it specify what the resultant behaviour should be. To avoid this ambiguity, informal text should not be used. As shown in Figure 4, the following alternative approaches can be used:

- informal text in task symbols can be replaced with procedure calls (even if dummy procedures are specified initially);
- informal text in decisions can be replaced with a **syntype**. Meaningful terms for the decision results can be specified using **synonyms**.

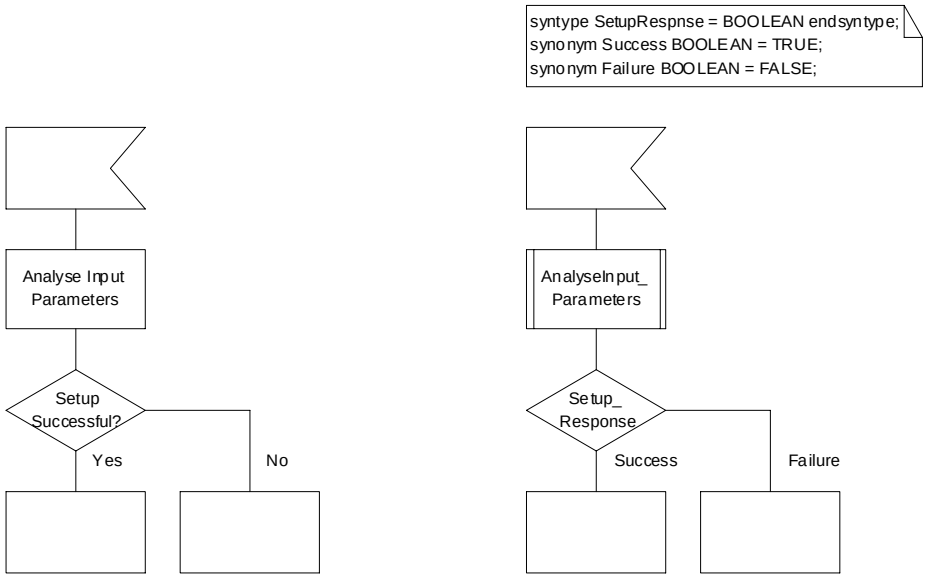


Figure 4: Alternatives to the use of informal text

NOTE: Although not often used in standards, the correct syntax of informal text is that it is enclosed in single quotes (e.g., 'text'). If the use of informal text is unavoidable in a standard then this form should be used.

7.2.1.4 Drawing rules

There are no specific recommendations on the use of SDL drawing rules

7.2.1.5 Comment

SDL comments should be used freely to annotate specifications.

7.2.1.6 Text extension

Text extensions may be used freely within specifications.

7.2.1.7 Text symbol

The text symbol is an essential component of an SDL specification and should be used wherever necessary.

7.2.2 Basic data concepts

An SDL description that does not use data is only useful for very simple descriptions. Data needs to be used to give formal meaning to tasks, decisions and parameters. More information on the use of *data types* and *expressions* can be found in subclause 7.5

7.2.2.1 Data type definitions

SDL data type definitions may be used freely within specifications although it is generally more acceptable to use the ASN.1 language to specify data types (sorts) in telecommunications standards. Neither SDL nor ASN.1 place implicit restrictions on the values of unbounded data types (such as Integer or Octet String) that can be assigned to a variable of that data type. As an example, integer variables are assumed to have a maximum value of infinity. ***A bounded range of possible values should be specified for all data types (SDL sorts), particularly implicit types such as INTEGER and REAL.***

7.2.2.2 Values and literals

SDL values and literals may be used freely within specifications.

7.2.2.3 Expressions

Expressions are a fundamental concept in SDL and should be used freely.

7.2.3 System structure

7.2.3.1 Organisation of SDL specifications

7.2.3.1.1 Package

The SDL package is a valuable tool in structuring a specification into easily identifiable units that can be reused in several places. It is a similar construct to the ASN.1 "module" and may be used freely.

7.2.3.1.2 Referenced definition

The use of graphical diagrams makes it essential that referenced definitions are used.

7.2.3.2 System

The SDL system construct is a valuable tool in the specification of telecommunications protocols and services and should be used wherever necessary.

7.2.3.3 Block

The SDL block construct is a valuable tool in the specification of telecommunications protocols and services and should be used wherever necessary.

7.2.3.4 Process

The SDL process construct is an essential tool in the specification of telecommunications protocols and services and should be used in all such standards.

7.2.3.5 Service

The semantics of the SDL service construct can be misleading and for this reason are not recommended for use in standards.

7.2.3.6 Procedure

SDL procedures are valuable for segregating specific aspects of behaviour within the specification of a telecommunications protocol or service and should be used freely.

7.2.4 Communication

7.2.4.1 Channel

Channels are essential for the interconnection of SDL blocks and should be used freely for this purpose in specifications.

It is possible to specify a channel with delay but this facility should be used with care. If more than one delaying channel is used to connect two blocks, the order of arrival of signals in the input queues is not determinable.

7.2.4.2 Signal route

Signal routes are essential for the interconnection of processes and for the connection of processes to the boundaries of blocks. As such, they should be used freely within specifications.

7.2.4.3 Connection

Connection labels are used in block diagrams to specify the connection of signal routes to channels at block boundaries. These should be used freely in specifications using blocks rather than block types. Specifications using block types should use gates for this purpose (see subclause 7.6.1.4).

7.2.4.4 Signal

SDL signals are essential in the identification and structural definition of protocol messages are permitted to pass across each communication path in a standardized system and should be used freely

7.2.4.5 Signal list definition

SDL signal list definitions provide a very convenient shorthand for collecting and naming groups of related signals. They can make a specification easy to read and should be used extensively.

7.2.5 Behaviour

7.2.5.1 Variables

Variables are an essential concept in the specification of SDL processes and should be used freely.

7.2.5.1.1 Variable definition

Variable definitions are used to declare the data type of an SDL *variable* (**dcl**) and, if necessary, to assign an initial value. They should, therefore, be used freely within specifications

If a variable is not initialized before it is accessed, then, according to Z.100, its contents are undefined. The subsequent behaviour of the system is likely to be unpredictable, particularly if the variable is based on a composite

data type such as an Array, a String or a Struct. For this reason, ***all composite data types should be initialized, preferably by a default initialization defined in the data type.*** For example:

```
newtype Chartable Array(Character, Octet)
    default (. 'FF'H .) comment initialize whole array to the hex value FF;
endnewtype Chartable;
dcl Chartab Chartable := (. 0 .) comment  alternative initialization to zero;
```

SDL permits variables to be identified in a **dcl** statement as **revealed** or **exported**. The dangers of using these facilities are discussed in subclauses 7.2.5.1.2 and 7.4.12 respectively.

7.2.5.1.2 View definition

SDL permits a *variable* in one process to be declared as a *view* of a *variable* (declared as *revealed*) in another process. This then implies that the viewing process can access the *variable* for reading at the same time that the revealing process is writing to it and this can lead to severe data synchronization problems. Consequently, view and reveal should not be used.

7.2.5.2 Start

The SDL start symbol is an important marker in a process diagram and should always be included. It is syntactically incorrect to omit it.

7.2.5.3 State

State symbols are essential in the specification of SDL processes and should be used freely.

7.2.5.4 Input (and invalid stimuli)

Input symbols are essential in the specification of SDL processes and should be used freely

A conformance test of a system may include tests for responses to invalid stimuli. However, the concept of an invalid stimulus does not exist in SDL. All declared signals are valid when received in inputs. The semantics of SDL assumes that the environment of the system sends only valid signals to the system.

If there is a need to specify the response of the system to an invalid stimulus, this should be defined explicitly using an appropriately named signal. Such a signal may represent more than one stimulus.

7.2.5.5 Save

Save symbols may be used freely within SDL specifications.

7.2.5.6 Spontaneous transition

A spontaneous transition can be executed any time without specifying precisely when and why it happens. It usually means that the stimulus that actually triggers such activity is not modelled. Spontaneous transitions lack the precision of specification that is necessary in most standards and should not be used

7.2.5.7 Label

Labels uniquely identify the point in a process graph where corresponding joins continue processing (see subclause 7.2.5.9) and should be used freely for this purpose.

7.2.5.8 Nextstate

The Nextstate symbol is essential in controlling the flow of a process and should be used freely.

SDL permits the use of a hyphen (-) character in a Nextstate symbol to indicate that the process should remain in the same state and this should be used with care (see subclause 7.4.8).

7.2.5.9 Join

Join symbols can be used freely although indiscriminate use can make process graphs difficult to read.

7.2.5.10 Stop

The stop symbol can be freely used for terminating dynamically created processes. They should not be used in processes that are not created dynamically unless the standard specifically requires operation to cease under certain circumstances.

7.2.5.11 Return

Return symbols identify the end of processing in procedure and operator diagrams and should be used freely for this purpose.

7.2.6 Action

7.2.6.1 Task

Task symbols are essential in specifying the functions represented by a process and should be used freely.

7.2.6.2 Create

Create symbols should be used freely when the dynamic creation of process instances is required.

7.2.6.3 Procedure call

Procedure call symbols represent in graphical form access to procedures (see subclause 7.2.3.6) and should be used freely for this purpose.

Access to value-returning procedures cannot be gained using the procedure call symbol; the **call** keyword must be used instead.

7.2.6.4 Output

Output symbols are used to represent the flow of protocol messages from one process to another and should be used freely.

7.2.6.5 Decision

Decision symbols represent the dynamic choices that can be made within a process and should be used freely.

7.2.7 Timer

Timers are valuable in the specification of protocols and should be used freely.

7.2.8 Internal input and output

Internal input and output are semantically equivalent to normal input and output. If their use is felt to improve the readability of specification, they can be used freely.

7.3 Structural Decomposition Concepts in SDL

7.3.1 Partitioning

7.3.1.1 Block partitioning

Blocks can be partitioned into further blocks (using a block substructure) or a number of processes and this capability should be used freely.

In standards, blocks should not contain both processes and a block substructure as they represent alternative descriptions of the same behaviour. Also, in order to ease understanding for the human reader, the number of constituent processes specified in a single block should be kept to a minimum

7.3.1.2 Channel partitioning

Channel partitioning creates alternative descriptions of a system at different levels of abstraction. It is difficult to derive conformance requirements from a standard that specifies aspects of a system at more than one level of abstraction and, therefore, channel partitioning should not be used.

7.3.2 Refinement

Refinement is used to specify signals at different levels of abstraction. It is difficult to derive conformance requirements from a standard that specifies aspects of a system at more than one level of abstraction and, therefore, refinement should not be used.

7.4 Additional Concepts of Basic SDL

7.4.1 Macro

Macros are expanded before the general grammar rules of SDL language are applied. The first consequence is that it is difficult to analyse the errors arising from expanded macros. Second, the SDL contained in the unexpanded macros may be difficult to understand. Finally, macros are not subject to the SDL scoping rules leading to increased likelihood of name clashes. The overall effect is that macros can be prone to errors.

Macros should only be used for some limited, well thought out purposes where their use can be proven to bring benefits and the macros have be shown to produce the required functionality

Some specific guidance is given for each of the sub-categories.

7.4.1.1 Macro, graphical behaviour

A graphical behaviour macro call symbol is expanded with the matching graphical macro definition containing part of an SDL process graph. Macro definitions that have one inlet and one outlet are preferred. In most situations procedures represent a superior alternative to graphical behaviour macros.

7.4.1.2 Macro, structural

A structural macro call symbol is expanded with the matching graphical macro definition containing part of an SDL system or block diagram. ***Structural macros should not be used because they do not help in understanding the specification of structure***. Superior SDL constructs, such as the partitioning of structures into several layers and the use of instances of block and process types should be used instead.

7.4.1.3 Macro, textual

The textual macro call (keyword **macro** followed by macro name) is expanded with the text contained in the matching macro definition. These should be kept simple and used with care.

7.4.2 Generic system definition

A system specification may have optional parts and system parameters with unspecified values in order to meet various needs. Such a system specification is referred to as "generic". Its generic property is specified by means of external synonyms. Most standards tend to describe generic systems with many optional features.

7.4.2.1 External synonym

External synonyms are particularly appropriate in the specification of options and should be used freely

7.4.2.2 Simple expression

Simple expressions are SDL expressions that rely only on data types (their literals, values and operators) that are predefined as part of the SDL language standard. They can be evaluated statically without interpretation of the whole SDL model and should be used freely.

7.4.2.3 Optional definition (The select construct)

The select graphical symbol is a graphical line surrounding part of an SDL system or block diagram and containing a simple Boolean expression which is usually based on an external synonym. The select symbol can be quite effective where a simple exclusive choice is to be made between two similar sections of a specification. However, ***in standards where it is necessary to specify multiple or complex options, the use of system and block types may be easier to understand than the use of numerous select symbols***.

7.4.2.4 Optional transition string

The option symbol in a transition allows the processing to branch based on the value of the simple expression contained in the option symbol. However, unlike decisions where the value of the expression changes dynamically, the value of the expression in option symbol can only be determined statically. In standards, ***optional transitions should be used freely to model the differences in behaviour between implementation options***. The expressions in the option symbols should thus be based on external synonyms that reflect choices that can be specified in ICS documents.

7.4.3 Asterisk state

Asterisk state is a way of specifying that in all states the mentioned inputs are processed in the same way. This can contribute to the overall reduction of the size of the specification but may lead to misinterpretation, particularly if used in a complex process specification with a large number of states.

It should be used with care, particularly if in combination with asterisk input and save symbols.

7.4.4 Multiple appearance of state

Several appearances of a state with the same name are often needed to split the state specification over several pages of SDL diagrams. For the purpose of simplicity, this approach should be used freely but it is advisable to use comments to indicate the existence of other partial state specifications.

7.4.5 Asterisk input

Asterisk input is a convenient shorthand notation for specifying the actions that follow the reception of any signal not explicitly mentioned in a particular state. This may help to reduce the overall size of the specification. However, it tends to lead to errors in specifications and is likely to cause difficulties in validation and testing. The main problem is that the actual signals that are covered by this notation can be misunderstood, both by the rapporteur and by the readers. It is safer to use an explicit list of signals instead.

7.4.6 Asterisk save

Asterisk save is a convenient shorthand notation for specifying that all signals not explicitly mentioned in a particular state are kept in the input queue for later processing in some other state. The main problem is that the actual signals

that are covered by this notation can be misunderstood, both by the rapporteur and by the readers. It is safer to use an explicit list of signals instead.

7.4.7 Implicit transition

Any signal that is mentioned in neither an input nor a save symbol is discarded. If the signal requires such treatment, it should either not be sent to the input queue or it should be removed by an explicit empty transition.

Implicit signal consumption can be detected in the validation of an SDL model. While this may not be an error, it often indicates that a signal has arrived in a state where this was not anticipated. As this can imply that there might be problems in the design, it is very useful to examine such situations.

Implicit transitions should be avoided in the specification of behaviour in a standard.

7.4.8 Dash nextstate

Dash nextstate is a way of indicating that a transition should terminate in the same state that it started from. It is used when a transition or a part of it is common to more than one state.

It can lead to errors when part of the transition is specified in one state graph with a join statement to a label in another state graph where processing is completed. A common error in this case is to assume that the dash nextstate will terminate the transition in the state at the start of the thread containing the dash nextstate symbol. Figure 5 shows two state transitions which join at label "A". A transition from State_1 will return to State_1 when the dash nextstate symbol is encountered whereas a transition from State_2 will return to State_2, not State_1.

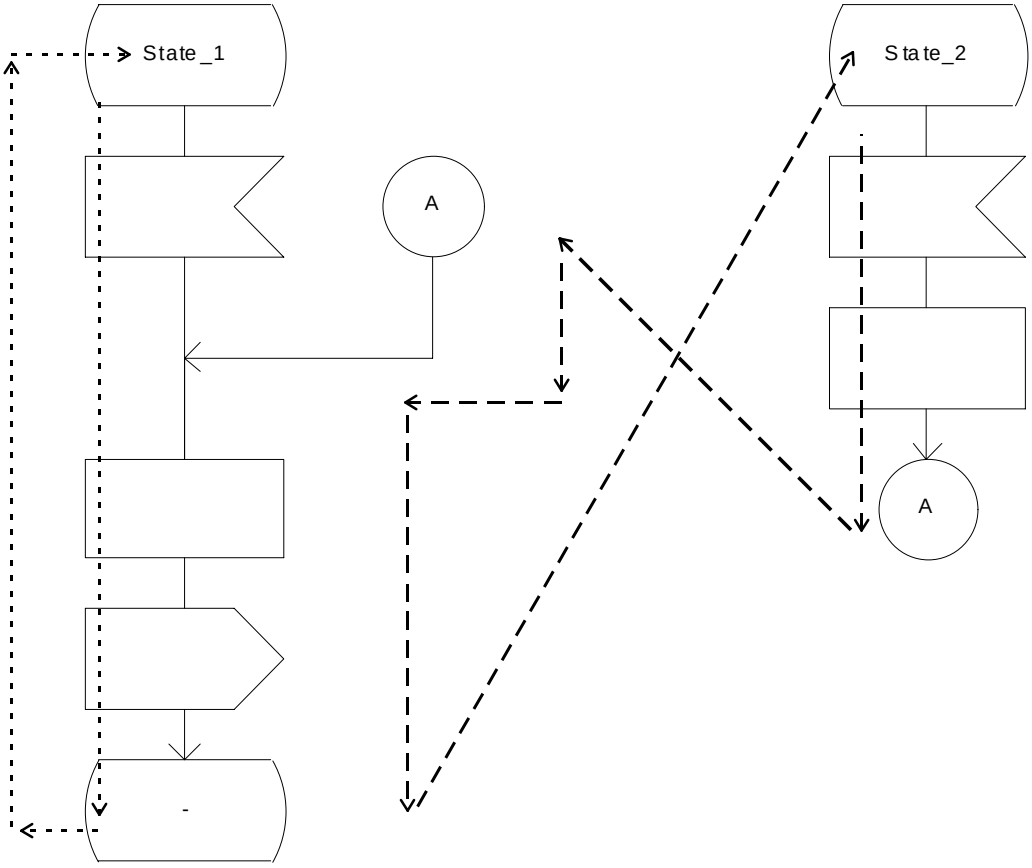


Figure 5: Example of the use of dashed nextstate

There appear to be no problems if the dashed nextstate is used where no part of the transition is common to several states, but that often changes in the course of the development. For such situations the recommendation still is to explicitly specify the state name in the nextstate symbol.

For the reasons stated above dash nextstate should preferably be used in situations where the whole transition is common to many states.

7.4.9 Priority Input

Priority inputs are useful for allowing certain events to be processed ahead of their position in the normal chronological sequence and should be used freely for this purpose.

7.4.10 Continuous signal

Continuous signals are a convenient way of triggering a transition on the basis of a boolean condition rather than a signal and can be used freely for this purpose.

7.4.11 Enabling condition

Enabling conditions can lead to non-deterministic observable behaviour which can result in either of the following situations when testing:

- the condition having parameters (for example, sent in a previous message), thus making the test more complex, in order to make the test decidable;
- apparent non-determinism (from the tester point of view) remaining in the validation model so that the test will often be "inconclusive".

Given this problem and the complex rules that govern the use of enabling conditions and since the same can be expressed using normal input and save symbols in different states, enabling conditions should not be used in standards.

7.4.12 Imported and Exported value

Import and export are shorthand notations for accessing variables from other processes. The implication of these constructs is that signals are sent between the processes. As SDL does not permit the explicit specification of the form and contents of these signals and as they may appear on normative interfaces, their use is unacceptable within standards and should be avoided. Explicit signal exchanges should be used instead of Import and export.

7.4.13 Remote procedures

Remote procedures are called from one process but defined and executed in another. As they can transfer values in both directions, they generate signals which are implicitly rather than explicitly specified and should be avoided except in instances where it is certain that the calling process and the remote procedure will always be co-located. In such cases, remote procedures can be used effectively to provide access to data that is common to a number of processes.

Remote procedure calls should always specify the destination of the call in the same way as output signals.

7.5 Data in SDL

Data is essential in giving formal meaning to any SDL system. It is conveyed in the parameters of signals and is held in variables within processes. Each of these parameters and variables has an associated data type (called sort in SDL) and can either be undefined or contain a value in the range that has been either explicitly or implicitly defined for that data type.

7.5.1 The data kernel language

Abstract data types are defined by a set of algebraic equations listed in the **axioms** of a **newtype**. This approach is the basis for establishing the properties of data in SDL and the Predefined data for SDL and this is called the data kernel in Z.100. SDL with ASN.1 is also defined using this approach. However, only a few of the corresponding SDL constructs should be used freely, as in general the algebraic approach is difficult to use and error prone.

7.5.1.1 Data type definitions

Data type definitions (using the keyword **newtype**) are essential for identifying new data types and for specifying their structure and scope. More information on the use of data types can be found in subclause 7.2.2.1

7.5.1.2 Literals and parameterised operators

Names for values of a data type are listed as **literals**, which can be used freely.

The name, parameter types and result type for each operator are listed under **operators**. The **operators** introduced by a **newtype** can be used freely. For clarity, it is preferable to have a textual operator definition or a reference to an operator diagram that is described informally, rather than informal text under **axioms**. An operator that is not described formally should be defined as **external**. Because the body of an **external** operator is not defined in SDL, such operators should be used with care and a good description of the operator should be given.

7.5.1.3 Axioms and conditional equations

Experience has shown that it is difficult to generate a set of correct and complete **axioms** and that in most cases it is not possible to derive an executable model from a specification which incorporates **axioms**. For these reasons, **axioms** should not be used to define the properties of operators.

Conditional equations are a class of equation specified under **axioms** and therefore should not be used.

7.5.2 SDL expressions and data types

Expressions are a fundamental concept in SDL and should be used freely.

Many constructs exist in SDL for writing expressions that depend simply on the literal values of data types and are independent of variables or value returning procedure calls or the SDL imperative operators (**now**, **import**, **parent**, **offspring**, **self**, **sender**, **active** or **any)**. These expressions are treated under "Passive use of Data" in Z.100 which defines constructs to allow:

- data types to be inherited;
- structures to be defined
- the use of:
  - infix operators

    for example, a + b instead of plus(a, b);
  - conditional expressions

    of the form, **if** boolean **then** consequence expression **else** alternative expression **fi**;
  - character string literals.

These constructs can and should normally be used freely with variables, value returning procedure calls and imperative operators.

Expressions that do not depend on the value of a variable or procedure call or imperative operator are called ground expressions in Z.100 and can be used freely. Ground expressions are the only expressions allowed in some contexts such as **synonym** definitions.

7.5.2.1 Special operators

The infix operators (=>, **or**, **xor**, **and**, **in**, /=, =, >, <, <=, >=, +, /, *, //, **mod**, **rem**, -) and the monadic prefix operators (-, **not**) should be used freely for the data types defined by the package Predefined and for other data types derived from them. If new operators are defined using operator diagrams, special operator names and symbols should not be used.

7.5.2.2 Character string, Bit_String and Hex_String literals

Character string literals are used to denote values of the predefined data types Character (for single characters) and Charstring (for one or more characters) and should be used freely for this purpose.

Bit_String or Hex_String literals are used to denote values of (or derived from) the Predefined data types Bit_String and Octet_String and should be used freely for this purpose

7.5.2.3 Predefined data, equality and noequality

The **package** 'Predefined' of Z.105 defines a number of useful data types such as Integer, Boolean and IA5String. Some of these data types have a fixed size (for example, Boolean can only take the values 'True' and 'False') whereas others are not bounded (integers, for example, can take any value up to infinity).

There are a number of other aspects that need to be considered when using predefined data, as follows:

- The data type **any** can have an infinite number of values but without denotations for the values. It should only be used for values that are passed transparently from one external interface to another. For validation and testing it is necessary to replace **any** by an actual data type. For these reasons the use of **any** is not recommended.
- The items **Bit**, **Boolean**, **Character**, **Null** and **Octet** are data types that can be used freely. Each of these data types have a finite number of values as follows:
  - **Bit**(2);
  - **Boolean**(2);
  - **Character**(128);
  - **Null**(1);
  - **Octet**(255).
- The data types **Integer** and **Natural** do not have defined bounds. Therefore, **syntype** and **newtype** should be used to limit the range of values for variables and parameters of these types.
- The items **Duration**, **Real** and **Time** are data types based on rational numbers (that is any number that can be represented by one integer divided by another integer) and therefore there are neither defined bounds nor a limit to the precision of these numbers. The unlimited precision means that there is always some valid value that is between any other two values. These types should therefore be used with care and syntype and newtype should be used to limit the range of values for variables and parameters of these types.
- The **Pid** data type can be freely used for process identity values. It is not possible to use it for any other purpose.
- The items **Array**, **Bag**, **Powerset**, **String** and **String0** are generators that can be used to define new data types.
  - **Bag**, **String** and **String0** generate data types that define strings of unlimited length so these must be used with care to ensure that the actual length is bounded.
  - **Arrays** can be used freely, provided the index has a finite number of values.
  - **Powerset** should be used with data types that have a limited number of values such as a user defined enumerated data type. A **Powerset** of a data type that has a large or infinite number of values also needs large or infinite numbers of values. They should be used with care and should not be sent as a signal parameter across a normative interface.
- The items **Bit_String**, **Charstring**, **GeneralizedTime**, **GraphicString**, **IA5String**, **NumericString**, **Octet_String**, **PrintableString**, **UniversalString**, **UTCTime** and **VisibleString** should be used with care because they are based on string generators and therefore have no maximum length. It is preferable that a **size** (see Z.105) is specified, for example:

```
syntype Bits16 = Bit_String constants size(0:16) endnewtype Bits16;
```

- The **String** data type is also used to provide the operators that ASN.1 **SEQUENCE OF** data types inherit. Therefore ASN.1 **SEQUENCE OF** data types can be used freely, provided there is a **size** specified.

- The purpose of the **Bag** data type is to provide the operators that ASN.1 **SET OF** data types inherit, and should not be used directly. However, ASN.1 **SET OF** data types can be used freely, provided there is a **size** specified to limit the actual size of the **Bag** values.

- The purpose of the **Enumeration** data type is to provide the operators that ASN.1 **ENUMERATED** data types inherit, and therefore should not be used directly. However, ASN.1 **ENUMERATED** data types can be used freely.

- The items **EXTERNAL_Type**, **Object_element**, **Object_Identifier**, and **ObjectDescripter** are ASN.1 data types that can be used according to the ASN.1 rules.

By default, all data types have the "=" and "<=" operators defined, so these can be used freely. The keyword **noequality** can be used for defining Predefined data but should not be used in normal specifications.

7.5.2.4 Boolean axioms, conditional terms and error!

Boolean axioms, conditional terms and the keyword **error!** are used to specify operator properties using axioms and therefore should not be used (see subclause 7.5.1.3).

NOTE: Conditional expressions (rather than conditional terms) can be used freely.

7.5.2.5 Ordering

The keyword **ordering** can be used freely when introducing a set of literals in SDL for a user defined type. It has the effect of defining the operators ",", ">", "<=" and "<=", such that the literals are ranked in the order they are listed in the specification.

7.5.2.6 Syntypes and range conditions

A **syntype** can be used freely in specifications to introduce an alternative name for a data type and a **constants** range condition associated with this name. This enables the value used in variables and parameters to be bounded, and enables checks to be made that the values are in range. Therefore, **constants** with a bounded range condition should be used freely.

The **constants** range condition limits values that a parameter or variable of a data type can have and can be used freely, provided the number of values is bounded.

7.5.2.7 Structures, SEQUENCE, SET and CHOICE

An SDL **struct**, an ASN.1 SEQUENCE or an ASN.1 SET can be used freely to define a structured data type that has named fields, where each field may have a different data type. For fields of a SEQUENCE or a SET, the attributes optional and default can be used freely.

An ASN.1 CHOICE data type is a special case of a structured data type where all the fields are mutually exclusive, and value of the structure is the same as the value of the present field. CHOICE data types should be used with care, because the actual value which is present may not be clear.

7.5.2.8 Inheritance

A **newtype** with the keyword **inherits** can be used freely to specify that one data type is distinct from another one but inherits the same set of values, literals and the specified set of operators. An ASN.1 sort assignment is an alternative syntax for the same thing and can be used freely to inherit values, literals and all operators.

NOTE: Z.105(03/95) contains an error in stating that:

```
Integerlist ::= INTEGER(0..5 | 10);
```

is the same as:

```
syntype S = Integer(0..5 | 10) endsyntype S;
```

whereas it is, in fact, the same as:

```
newtype Integerlist
    inherits Integer
    operators all
        Integer(0..5 | 10)
endnewtype Integerlist;
```

7.5.2.9 Generators

Although generators are used to define templates for some Predefined data types such as Array, general use is not recommended because there is only limited checking that can be done on a generator before it is applied. Data types with context parameters provide an alternative mechanism for defining such templates that is consistent with the way other types used in SDL.

7.5.2.10 Synonyms

Synonyms should be used freely to give meaningful names to values. They can be defined as **external** for values that are used to distinguish implementation options.

7.5.2.11 Name class literals and literal mapping

Name class literals allows a large number of literals to be defined for a data type. It is useful mechanism for defining the literals of Predefined data such as Integer and Charstring. However, it is not recommended for general use because:

- it can be difficult to validate that specifications using the construct are correct;
- it can easily lead to data types with infinite numbers of values.

Wherever possible the literals should be listed explicitly.

Because literal mapping is associated with both name class literals and the use of **axioms,** it should not be used (see subclause 7.5.1.3).

7.5.2.12 Operator definitions and operator applications

Operator definitions allow operators to be defined in a manner resembling value returning procedures. They should be used freely to specify the function of operators introduced in a **newtype**.

Operator applications can be used freely to invoke either user defined operators, or operators inherited from Predefined data.

7.5.2.13 Indexed primary and indexed variable

An indexed primary is normally used to access an array or a string item and can be used freely for this purpose. An indexed primary can also be used with any other data type for which the Extract! Operator is defined but it is not recommended that this operator is introduced explicitly for user defined data types.

When assigning a value to an indexed variable (array element), it is required that the complete (array) variable is assigned a value before an assignment is made to the indexed part of it. Otherwise, indexed variables can be used freely.

7.5.2.14 Field primary and field variable

A field primary is normally used to access the field of a structured data type and can be used freely for this purpose.

When assigning a value to a field variable (structure element), it is required that the complete (structure) variable is assigned a value before an assignment is made to the indexed part of it. Otherwise, indexed variables can be used freely.

7.5.2.15 Structure primary (and array value)

A structure primary of the form, '(. <expression list> .)', can be used freely to construct a structure value from a list of values corresponding to the fields of the structure. The same construct can also be used freely in the form, '(. <expression> .)' to construct a value for a complete array where every element has the same value.

A structure primary can also be used with any other data type for which the Make! Operator is defined, but it is not recommended that this operator is introduced explicitly for user defined data types.

7.5.2.16 Conditional expression

A conditional expression (**if** <boolean> **then** <ground expression> **else** <ground expression> **fi**) can be used freely wherever an expression is needed.

7.5.2.17 Variable access

Variables can be accessed freely but care should be taken that a variable has a value associated with it either by assignment or from initialization. A variable that does not have a value is undefined and an access in this case means that the further behaviour of the system is undefined.

7.5.2.18 Assignment statement

Assignment statements can be used freely.

7.5.2.18.1 Default initialization

Default initialisations ensure that an initial value is pre-set into the relevant variable. They can help to avoid indeterminate behaviour and should be used freely.

7.5.2.19 Imperative operators

7.5.2.19.1 Now expression

The **now** operator to obtain the global system time should be used with care, because it is unlikely that real systems can maintain a globally synchronised clock. Its primary use within standards should be in the '**now** + nn' form for setting timers:

Example:

```
set (now + 50, Timer1)
```

7.5.2.19.2 Import expression

The **import** expression for obtaining the value of a variable in another process should not be used. For more details see subclause 7.4.12

7.5.2.19.3 Pid expression

The Pid expressions **parent**, **offspring**, **sender** and **self** provide the identities of the current and related processes. They can be used freely for the control of processes but, because of their transient nature, should not be used for identification purposes within the parameters of signals sent across normative interfaces.

7.5.2.19.4 View expression

The **view** expression for obtaining the value of a variable in another process should not be used. For more details see subclause 7.2.5.1.2.

7.5.2.19.5 Timer active expression

The **active** expression can be used freely to check if timers are active.

7.5.2.19.6 Anyvalue expression

The **any** expression should not be used because its behaviour is not determinable and is, therefore, impossible to validate or test.

However, within a validation model, the **any** expression may be useful for specifying non-deterministic behaviour, such as user activity, where the interaction with the system is via non-normative service interfaces. Wherever **any** is used, a comment should be added to explain the basis for determining the value.

7.5.2.20 Value returning procedure call

Value returning procedures can be used very effectively to improve the readability of an SDL specification and should be used freely.

7.5.2.21 External data

Because the meaning of external data to refer to an **alternative** data formalism is not well defined, it is likely to lead to misunderstanding and ambiguities and should not be used.

7.6 Structural Typing Concepts in SDL

SDL systems, blocks and processes have the basic characteristics of objects - they contain internal data and communicate in principle using messages. The typing concept gives SDL more object-oriented characteristics - inheritance, specialization and parameterization of definitions.

7.6.1 Types, instances, and gates

7.6.1.1 Type definitions

SDL structures can be specified of without the use of types. However, type definitions can be useful for the following reasons:

- they enable a specification to have several instances of the same type;
- they essential for the object-oriented use of SDL;
- they can inherit the properties of an existing type and specialize it by adding or changing properties.

Type definitions are particularly useful in packages where definitions can be collected for use in the specification of several different systems.

Type definitions should be used freely in situations where above benefits can be realized.

7.6.1.1.1 System type

System types are unlikely to cause any specification or interpretation problems and, thus, can be used freely. However, many standards are described as only one system and in these cases, system definitions, rather than system types, are likely to result in a clearer overall specification.

7.6.1.1.2 Block type

Unlike block definitions, block type definitions can be reused in several places and are an attractive approach in standardizing systems that have numerous blocks with similar functions. In such situations, block type definitions can be used freely.

7.6.1.1.3 Process type

Unlike process definitions, process type definitions can be specialized and reused in several places. They are an attractive approach in standardizing systems that have numerous processes with similar functions and can be used freely in such situations.

7.6.1.1.4 Service type

Service type definitions should not be used for the same reasons described for service definitions in subclause 7.2.3.5.

7.6.1.2 Type expression

A type expression is used when defining one type (system, block, process) in terms of another and can be used freely for this purpose.

7.6.1.3 Definitions based on types

In this document the definitions based on type are referred to as 'type instances'. They are necessary if type definitions are used.

7.6.1.3.1 System definition based on system type

System definitions based on system types (system type instances) are necessary if a system definition is given as a type and can be used freely for this purpose.

7.6.1.3.2 Block definition based on block type

Block definitions based on block types (block type instances) are necessary if a block definition is given as a type and can be used freely for this purpose.

It is possible to specify that there are more than one block type instance with the same name by specifying the required number in the instance specification. However, there is no SDL construct for the dynamic creation of such block instances. Also, addressing messages to a particular instance requires an extensive and complex additional description of system initialization. It is, therefore, recommended that the number of instances is limited to the default value of one.

7.6.1.3.3 Process definition based on process type

Process definition based on process type or process type instances are useful where several process type instances are required or where specialization is being used. They should be used freely for this purpose. Process instances can be dynamically created and terminated. The initial and maximal number of process instances should be specified.

7.6.1.3.4 Service definition based on service type

Service definition based on service type should not be used for the same reasons described for service definitions in subclause 7.2.3.5

7.6.1.4 Gate

Gates specify the interface of a block or process type giving it a name and the list of signals that can be used in each direction. They are essential for specifying how instances of block and process types are connected to communications paths and should be used freely for this purpose.

7.6.2 Context parameter

Context parameters provide a means of defining system types, block types, process types and procedures with parameters that are given actual identities for the context in which they are used. The actual context parameter provided when the type is used will be an identifier for an entity that conforms to the formal context parameter. Context parameters should be used with care as overuse may make the description difficult to understand.

7.6.3 Specialization

Specialization is to the ability to define a new system, block or process type such that it inherits all the properties of an existing type plus new properties and some modifications to existing properties. SDL supports this but also has mechanisms for limiting the amount of change that is possible.

Specialization should be used with care for the following reasons. First, the overall properties of a type can only be deduced by applying the changes specified in the specialized type to the inherited type. Since there could be several levels of specialization, understanding the result can be difficult. Second, inheritance introduces dependencies between type definitions which could be difficult to keep under control.

7.6.3.1 Adding properties

When using specialization, properties such as channels, processes, states and transitions can be added to the resulting system, block or process. For the reasons given in subclause 7.6.3, properties should only be added with care.

7.6.3.2 Virtual type

A virtual type is always defined in the context of another type. For example a virtual process type can be defined as part of a block type definition. If another block type definition inherits the existing block definition the virtual process type definition can be redefined under the same name. Only the keyword **finalized** prevents the type definitions to be redefined further.

Virtual types used in a controlled way can be useful but, because they can become very difficult to understand, they should be used with care.

7.6.3.3 Virtual transition/save

In a process that is redefined, input symbols and save symbols that may be changed contain a keyword “Virtual”. As part of the redefinition, the following can occur:

- a new transition can be defined to replace the old one;
- inputs can be transformed into save symbol;
- save symbols can be transformed into inputs.

Changed transitions begin with an input symbol containing either the keyword “Redefined” or “Finalized”. The former can be further specialized while the latter cannot.

For the reasons given in subclause 7.6.3, virtual transitions and saves should be used with care.

8 Concepts in MSC

8.1 Introduction to MSC

Message Sequence Charts (MSCs) are used to describe sequences of events that can be performed by a standardized system. They are useful for giving an overview and have an important role to play in the understanding and development of both the SDL in a standard and the corresponding validation and testing material. The MSCs are often used as the basis of test sequences.

The MSCs in a standard should be consistent with the SDL in that the SDL should be capable of handling the stimuli and generating the responses defined by the MSCs. However, the MSCs cannot be considered normative in the same sense as the SDL. A system is not required to follow the sequence described in an MSC unless it is the only sequence that is valid according to the SDL. Thus, ***MSCs should be used to supplement the SDL to give descriptions of some valid sequences of behaviour***.

The availability of examples of possible message flows is very helpful during the development of test specifications for protocol or service standards. Thus, ***MSCs should be used to give at least one example of message exchanges for each required system function and should give examples of message exchanges in exceptional conditions.***

The following subclauses identify the MSC concepts used in Z.120[3] and provide a classification and justification for each, as follows:

- Use freely
    - the feature provides positive support for the SDL description and is without any negative impact on validation and testing;
- Use with care
    - use of the feature has some limited impact on validation and testing, or may be difficult to understand and therefore has limited value in support of the SDL.;
- Not recommended
    - the feature is complex or difficult to understand , and makes validation and testing difficult;
- Do not use
    - use of the feature is likely to produce behaviour traces that cannot be validated or tested.

Particular attention is paid to those concepts that are classified as "use with care" and "not recommended" to ensure that the criteria for using or avoiding the concepts are clearly expressed.

MSC's used in standards should always conform to the rules of syntax and semantics elaborated in ITU-T Recommendation Z.120 [3].

8.2 General rules

MSC may be expressed in either a graphical form or in text. Wherever MSCs are used to illustrate message flows in a standard, they should be expressed in the graphical form as this is much clearer and easier to understand than the textual form.

8.2.1 Lexical rules

The MSC lexical rules closely follow those of SDL except that MSC has a different set of keywords. In order that MSC can be used freely with SDL and ASN.1, ***the characters '{', '}', '[', ']', '|' (vertical bar) and space should not be used in MSC names, and the use of SDL keywords for names should be avoided***.

8.2.2 Visibility and naming rules

The set of MSCs in a standard can be considered a single MSC document, which is the only scope unit in MSC. Names are globally visible and cannot be qualified as in SDL. Thus, no two items belonging to the same class (condition, instance, message, MSC, timer) should have the same name. Note that this does not apply to gate names, which are implicitly qualified by being associated with a specific MSC. A message name identifies the purpose and content of a message, and the name can be used for several communications of the message.

As far as possible, ***names in an MSC should be the same as the names of corresponding entities in the SDL***. For example, a message name should be the same as a corresponding signal name, and an MSC instance should have the same name as the corresponding SDL process or block.

8.2.3 Comment

Comments should be used freely to annotate the MSC descriptions, and to link to them to the corresponding SDL.

8.2.4 Drawing rules

The rules given in Z.120 [3] should be followed. To aid understanding, ***attention should be given to the order of MSC instances left to right and avoiding messages that cross instances***.

8.2.5 Paging of MSCs

Partitioning an MSC over more than one "page" (normally a figure in a standard) is undesirable. Composition mechanisms should be used to avoid diagrams that are larger then one page. However, partitioning over pages is sometimes unavoidable. ***Where paging is used, the MSC pages should be numbered according to Z.120*** [3].

8.3 Message Sequence Chart documents

The MSC diagrams in a standard can be considered as an MSC document without the need to include the syntax for an <MSC document>. This syntax may be omitted because in this context the set of MSC diagrams is obvious and the related SDL is also obvious. If omitted, the MSC document (formally required by Z.120 [3]), can be derived implicitly and will include all the basic MSC diagrams and high level MSC (HMSC) diagrams in the standard.

Basic MSC should normally be used in combination with an HMSC (see 8.4.5) so that there is an overview of the whole set of behaviour traces covered by the MSC.

8.3.1 Message Sequence Chart

A basic Message Sequence Chart describes a sequence of message interchanges between instances. It will normally start from a particular condition of the system and finish with in a specific condition. ***A basic MSC should be given a name that is meaningful for the sequence***.

8.3.2 Instance

Instances are fundamental to MSC and should be used freely. They represent the communicating entities and can correspond to the SDL entities or to items in the environment.

For clarity, the instance name should be placed inside the instance head symbol with the instance kind placed above. ***The form of heading used should be consistent in all MSC diagrams***.

8.3.3 Message

Message symbols and instance entities are essential MSC concepts and should be used freely.

MSC does not require the parameters of a message communication to be described. However, a parameter description should be provided although this has no formal meaning in the MSC notation.

8.3.3.1 Message overtaking

Within the syntax and semantics of MSC, it is possible for one message to overtake another message. Message overtaking can represent message re-ordering in an underlying communication layer or the effect of saving signals in an SDL process input queue. ***Message overtaking should be avoided, except for sequences where it is essential to show the behaviour when overtaking takes place***.

8.3.3.2 Incomplete messages

A incomplete message communication is represented by a lost message symbol or a found message symbol. A lost message is one that is output but is never input whereas a found message is one that appears without an obvious source instance. ***Lost and found message should be used with care*** because they normally correspond either to the behaviour of the environment or the behaviour of the underlying system. They should not be used to describe traces of normal behaviour of systems.

8.3.3.3 Correspondence between SDL and MSC behaviour

Some traces described in MSC can cover situations that cannot logically occur in the SDL in a standard because assumptions are made about the characteristics of the environment or underlying layers for the protocol or service being described. For example, the SDL structure description of a transport layer may be a channel whereas an actual underlying layer may lose or re-order messages. If the protocol requires that such message loss or re-ordering is handled, the SDL functional behaviour will include the handling of lost and re-ordered messages though this could

not occur according to the properties of the SDL channel. A testing or validation model would probably need to replace the channel with a functional model. A related MSC should therefore be provided that reflects the loss or re-ordering of messages. Message overtaking (see 8.3.3.1) and incomplete messages (see 8.3.3.2) are useful concepts in these situations.

MSC descriptions that cover behaviour which does not correspond directly to the SDL should be clearly annotated.

8.3.4 Environment and gates

A set of basic MSC diagrams that does not contain any referenced MSC diagrams is considered as communicating with the environment of the system. The environment is normally represented by the frame of the MSC though it may be also be represented by one or more instances.

An MSC that contains a referenced MSC diagram (see 8.4.4) can have communications with the referenced diagram. For example, a message can be output in the referencing diagram and be input in the referenced diagram. For messages to (or from) a referenced MSC diagram, the message symbols are associated with gates. There are corresponding gates on the frame of the reference diagram. Where there is only one communication of each message between the two diagrams the gates do not need to be further identified. Unnamed gates are considered to be implicitly named by the message and the message direction. Where there are multiple communications of a message the gates need to be named to distinguish one from another.

Similarly, an MSC that contains an inline expression (see 8.4.3) can have communications with the inline expression. Gates are used to identify the messages to (or from) the inline expression.

Gates should be used with care because they make the MSC more complex and, therefore, more difficult to understand. Gates should not be given names unless this is necessary because names can clutter MSC diagrams and the message symbol name is usually sufficient.

If there is no message symbol leading to (or from) a gate in the referencing diagram then there is an implicit message symbol leading from (or to - respectively) the frame of the referencing diagram. A similar rule applies for gates on inline expressions. This can lead to some ambiguity of whether the gate should be connected to an ordinary message symbol or to a lost or found message symbol***. There should always be an explicit message symbol for each gate of a referenced diagram or inline expression in the enclosing diagram*** so that it is clear what communication actually takes place.

Gates on referenced MSC diagrams and inline expressions can also be used to order otherwise unordered events within a diagram or inline expression. These order gates are associated with the corresponding events by a general order symbol (see 8.3.5) and must be named. The order of events within a diagram or expression is controlled by the order (top to bottom) that names appear when the diagram is referenced or the inline expression is used. Diagrams with order gates are not easy to understand and the general order symbol can be confused with a message instance symbol. For these reasons, the use of general order gates is not recommended.

8.3.5 General ordering

To specify event ordering in cases where the events are not ordered by message communication, general ordering is used. General ordering clutters a diagram and can be difficult to understand when there are several general order symbols. It is often possible to replace the general ordering with simpler diagrams if alternatives are used. For the purposes of understanding the behaviour, diagrams without general ordering are often sufficient. The use of general ordering is not recommended, and wherever possible it should be replaced with simpler sequences that adequately cover the required cases.

It is recommended that the general order symbol is used rather than the single line symbol, because the latter can only be used within column instances.

8.3.6 Condition

Conditions are very useful for indicating initial and final conditions and should be used freely. However, conditions should not be used for composing several related MSC. HMSCs should be used for this purpose.

8.3.7 Timer

It is very useful to show timer activity (start, restart or expiration) in relation to other messages in MSC diagrams. Timers should thus be used freely.

8.3.8 Action

MSCs should predominately describe message exchanges but in some situations it is useful to indicate also the action that is performed after some message is received.

8.3.9 Instance creation

Instance creation should be used with care. In standardization, it should be avoided in the same manner as dynamic process creation in SDL specifications. Instance creation should only be shown if it is needed to understand the MSC diagram.

8.3.10 Instance stop

Instance stop should be used with care in situations where instance creation is also used.

8.4 Structural concepts

8.4.1 Coregion

Coregions are useful for showing situations where two or more events need to occur in any order before proceeding. They should, however, be used with care to ensure, in particular, that the number of events in the coregion is small.

8.4.2 Instance decomposition

Instance decomposition introduces a level of complexity into MSC diagrams that generally is not desirable. Its use, therefore, is not recommended.

8.4.3 Inline expression

Inline expressions are used to define, in a concise form, several different sequences that can occur at the same place in the enclosing diagram. A diagram using an inline expression is equivalent to several diagrams where the inline expression is replaced by each of the defined sequences in turn. Inline expressions give the benefit of conciseness at the expense of making the language used more complex and should, therefore, be used with care.

8.4.4 MSC reference

MSC references are a means of connecting several MSC diagrams together. Their use is essential if HMSC diagrams are included. However, without an HMSC diagram, it may be difficult to relate the diagrams to one another. In such case, references should be used with care.

8.4.5 High-level MSC (HMSC)

High-level MSCs can be used to relate a number of simpler MSC diagrams generally improving their readability making them easier to understand. HMSCs provide useful overviews of systems and should be used freely.

Annex A: Summary of use of SDL and MSC in ETSI Standards

A.1 Selection of SDL concepts

Table 1 identifies the SDL concepts specified in Z.100 and provides a classification and references for each.

Table 1: Classification of SDL concepts

Concept	Category	Recommendation on Use	Internal Reference	External References
Abstract data type	data	Use with care	7.2.2.1	Z.100 2.3.1 page 23 Z.100 5.1.1 page 115 Z.100 5.1.2 page 115
Adding	type	Use with care	7.6.3.1	Z.100 6.3.1 page 182
Any sort	data	Not recommended	7.5.2.3	Z.105 III page 47
Anyvalue expression	expression	Do not use	7.5.2.19.6	Z.100 5.4.4.6 page 163
Any decision	expression	Do not use	7.5.2.19.6	Z.100 5.4.4.6 page 163
Array	data	Use freely	7.2.5.1.1 7.5.2.13 7.5.2.15	Z.100 Annex C.8
ASN.1 module	data	Use freely	7.2.1.1 7.2.2.1 7.5.1.1 7.5.2.3 7.5.2.7 7.5.2.8	Z.105 II.1.1 page 33
Assignment	behaviour	Use freely	7.5.2.18	Z.100 5.4.3 page 157
Asterisk input	behaviour	Use with care	7.4.5	Z.100 4.6 page 103
Asterisk save	behaviour	Use with care	7.4.6	Z.100 4.7 page 103
Asterisk state	behaviour	Use with care	7.4.3	Z.100 4.4 page 102
Axioms	behaviour	Do not use	7.5.1.3	Z.100 5.2.3 page 121
Bit string	data	Use with care	7.5.2.2 7.5.2.3	Z.105 III page 47
Block	structure	Use freely	7.2.3.3	Z.100 2.4.3 page 30
Block definition based on block type	structure	Use freely	7.6.1.3.2	Z.100 6.1.3.2 page 173
Block partitioning	structure	Use with care	7.3.1.1	Z.100 3.2.2 page 83
Block type	structure	Use freely	7.6.1.1.2	Z.100 6.1.1.2 page 167
Boolean	data	Use freely	7.5.2.3 7.5.2.16	Z.100 Annex C.1 Z.105 III page 48
Boolean axiom	behaviour	Do not use	7.5.1.3	Z.100 5.3.1.5 page 129
Channel	structure	Use freely	7.2.4.1	Z.100 2.5.1 page 42
Channel partitioning	structure	Do not use	7.3.1.2	Z.100 3.2.3 page 86
Character	data	Use freely	7.5.2.2 7.5.2.3	Z.100 Annex C.2
Character string	data	Use with care	7.5.2.2 7.5.2.3	Z.105 III page 48 Z.100 Annex C.4
Character string literals	data	Use freely	7.5.2.2	Z.100 5.3.1.2 page 127
Choice	data	Use with care	7.5.2.7	Z.105 4.2.4 page 13 Z.105 4.4.1 page 19 Z.105 II.3.5 page 41 Z.105 III page 48
Comments	presentation	Use freely	7.2.1.5	Z.100 2.2.6 page 21 Z.105 2.6 page 4
Conditional equation	data	Do not use	7.5.1.3	Z.100 5.2.4 page 124
Conditional expression	behaviour	Use freely	7.5.2.16	Z.100 5.4.2.3 page 155

Concept	Category	Recommendation on Use	Internal Reference	External References
Conditional term	data	Do not use	7.5.2.4	Z.100 5.3.1.6 page 130
Constraint	behaviour	Use with care	7.6.1.4 7.6.2	Z.100 6.1.4 page 175 Z.100 6.2.1 page 179 Z.100 6.2.2 page 179 Z.100 6.2.7 page 181 Z.100 6.2.9 page 182 Z.100 6.3.2 page 184
Context parameter	type	Use with care	7.6.2	Z.100 6.2 page 177
Continuous signal	behaviour	Use freely	7.4.10	Z.100 4.11 page 105
Create	behaviour	Use freely	7.2.6.2	Z.100 2.7.2 page 63
Dash nextstate	behaviour	Use with care	7.4.8	Z.100 4.9 page 104
Data definition	data	Use freely	7.2.2.1 7.5.1.1	Z.105 4.1 page 6
Data type (sort in Z.100)	data	Use freely	7.2.2.1 7.5.1.1	Z.100 5.2.1 page 116
Data type assignment	data	Use freely		Z.105 4.1.1 page 6
Data type context parameter	data	Use with care	7.6.2	Z.100 6.2.9 page 182
Data type Expressions	data	Use freely	7.2.2.1 7.5.1.1	Z.105 4.2 page 7
Decision	behaviour	Use freely	7.2.6.5	Z.100 2.7.5 page 68
Default initialization	data	Use freely	7.5.2.18.1	Z.100 5.4.3.3 page 159
Delay	behaviour	Use with care	7.2.4.1	Z.100 2.5.1 page 42
Duration	data	Use with care	7.5.2.3	Z.100 Annex C.11
Enabling condition	behaviour	Do not use	7.4.11	Z.100 4.12 page 106
Enumerated	data	Use freely	7.5.2.3	Z.105 4.2.5 page 14 Z.105 III page 49
Environment	structure	Use freely	7.2.5.4 7.5.2.19.6	Z.100 1.3.2 page 5
Equality	data	Use freely	7.5.2.3	Z.100 5.3.1.4 page 129
Error	behaviour	Do not use	7.5.2.4	Z.100 1.3.3 page 5 Z.100 5.3.1.7 page 131
Expression	behaviour	Use freely	7.2.2.3 7.5.2	Z.100 2.3.4 page 23 Z.100 5.3.3.1 page 148
External data	data	Do not use	7.5.2.21	Z.100 5.4.6 page 164
External synonym	data	Use freely	7.4.2.1	Z.100 4.3.1 page 97
Field	data	Use freely	7.5.2.14	Z.100 5.3.3.5 page 151 Z.100 5.4.3.2 page 158
Finalized	type	Use with care	7.6.3.2 7.6.3.3	Z.100 6.3.2 page 183
Frame	presentation	Use freely	7.2.1.4	Z.100 2.2.5 page 20
Gate	type	Use freely	7.6.1.4	Z.100 6.1.4 page 175
Generator	type	Not recommended	7.5.2.9	Z.100 5.3.1.12 page 139
Identifier, keyword, name, qualifier and lexical rules	presentation	Use freely	7.2.1.1	Z.100 2.2.1 page 13 Z.100 7.2 page 198 Z.105 2.1 page 3 Z.105 2.3 page 3 Z.105 2.7 page 4 Z.105 2.8 page 5
Implicit transition	behaviour	Not recommended	7.4.7	Z.100 4.8 page 103
Import and Export	behaviour	Do not use	7.4.12	Z.100 2.6.1.1 page 50 Z.100 4.13 page 109 Z.100 5.4.4.2 page 161
Indexed primary or variable	behaviour	Use freely	7.5.2.13	Z.100 5.3.3.4 page 151 Z.100 5.4.3.1 page 158

Concept	Category	Recommendation on Use	Internal Reference	External References
Infix operator	behaviour	Use freely	7.5.2 7.5.2.1	Z.100 5.3.1.1 page 126 Z.105 2.5 page 4
Informal text	behaviour	Do not use	7.2.1.3	Z.100 2.2.3 page 20
Inheritance	type	Use freely	7.5.2.8	Z.100 5.3.1.11 page 137
Input	behaviour	Use freely	7.2.5.4	Z.100 2.6.4 page 53
Instance	structure	Use freely	7.6.1	Z.100 1.3.1 page 3
Integer	data	Use with care	7.5.2.3	Z.100 Annex C.5 Z.105 III page 49
Integer Naming	data	Use freely	7.2.1.1	Z.105 4.2.6 page 15
Internal input and output	presentation	Use freely	7.2.8	Z.100 2.9 page 71
Join	behaviour	Use freely	7.2.5.9	Z.100 2.6.8.2.2 page 60
Keywords	presentation	Use freely	7.2.1.1	Z.100 2.2.1 page 13 Z.105 2.2 page 3
Lexical rules	presentation	Use freely	7.2.1.1	Z.100 2.2.1 page 13 Z.105 2.1 page 3 Z.105 2.3 page 3 Z.100 7.2 page 198 Z.105 2.7 page 4
Literal	data	Use freely	7.2.2.2 7.5.1.2	Z.100 2.3.3 page 23 Z.100 5.2.2 page 119
Literal mapping	data	Do not use	7.5.2.11	Z.100 5.3.1.15 page 144
Macro, graphical behaviour	presentation	Not recommended	7.4.1.1	Z.100 4.2 page 91
Macro, structural	presentation	Do not use	7.4.1.2	Z.100 4.2 page 91
Macro, textual	presentation	Use with care	7.4.1.3	Z.100 4.2 page 91
Name	presentation	Use freely	7.2.1.1 7.2.1.2	Z.100 2.2.1 page 13 Z.105 2.1 page 3 Z.105 2.3 page 3 Z.100 7.2 page 198 Z.105 2.8 page 5
Name class literals	data	Not recommended	7.5.2.11	Z.100 5.3.1.14 page 143
Natural	data	Use with care	7.5.2.3	Z.100 Annex C.6
Nested diagram	presentation	Use with care		Z.100 2.4.13 page 26
Newtype	data	Use freely	7.5.2.3	Z.100 5.2.1 page 117
Nextstate	behaviour	Use freely	7.2.5.8 7.4.8	Z.100 2.6.8.2.1 page 59
Noequality	data	Do not use	7.5.2.3	Z.100 5.3.1.4 page 129
Now	behaviour	Use with care	7.5.2.19.1	Z.100 5.4.4.1 page 160
Null	data	Use freely	7.5.2.3	Z.105 III page 49
Object identifier	data	Use freely		Z.105 4.2.2.3 Z.105 III page 49
Octet string	data	Use with care	7.5.2.3	Z.105 III page 50
Operator	behaviour	Use freely	7.5.1.2 7.5.2.1 7.5.2.12 7.5.2.19	Z.100 5.2.2 page 119 Z.100 5.3.2 page 146 Z.100 5.4.2.4 page 156 Z.105 II.1.3 page 34 Z.105 II.3.1 page 3
Operator diagram	behaviour	Use freely	7.2.5.11 7.5.1.3 7.5.2.1	Z.100 5.3.2 page 146
Option	behaviour	Use freely	7.4.2.1 7.4.2.3 7.4.2.4	Z.100 4.3.4 page 100
Optional fields	data	Use freely		Z.105 4.2.1 page 8
Ordering	data	Use freely	7.5.2.5	Z.100 5.3.1.8 page 132
Output	behaviour	Use freely	7.2.6.4 7.2.8	Z.100 2.7.4 page 65
Package	structure	Use freely	7.2.3.1.1	Z.100 2.4.1.2 page 24 Z.105 3 page 5
Page	presentation	Use freely	7.2.1.4	Z.100 2.2.5 page 20

Concept	Category	Recommendation on Use	Internal Reference	External References
Pid	data	Use with care	7.5.2.19.3	Z.100 5.4.4.3 page 161 Z.100 Annex C.10
Powerset	data	Use with care	7.5.2.3	Z.100 Annex C.9
Predefined	data	Use freely	7.5.2.3	Z.100 5.3.1.3 page 128
Priority Input	behaviour	Use freely	7.4.9	Z.100 4.10 page 104
Procedure	behaviour	Use freely	7.2.3.6	Z.100 2.4.6 page 39 Z.100 2.7.3 page 64
Procedure context parameter	type	Use with care	7.6.2	Z.100 6.2.2 page 179
Process	behaviour	Use freely	7.2.3.4	Z.100 2.4.4 page 32 Z.100 6.1.1.3 page 168
Process context parameter	type	Use with care	7.6.2	Z.100 6.2.1 page 179
Process definition based on process type	type	Use freely	7.6.1.3.3	Z.100 6.1.3.3 page 173
Qualifier	presentation	Use freely	7.2.1.2	Z.100 2.2.2 page 17 Z.105 2.1 page 3 Z.105 2.3 page 3 Z.100 7.2 page 198
Range condition	data	Use freely	7.5.2.6	Z.100 5.3.1.9.1 page 134 Z.105 4.3 page 16
Real	data	Not recommended	7.5.2.3	Z.100 Annex C.7 Z.105 III page 50
Redfined	type	Use with care	7.6.3.2	Z.100 6.3.2 page 183
Referenced diagram	presentation	Use freely	7.2.3.1.2	Z.100 2.4.13 page 26
Remote procedure	behaviour	Use with care	7.4.13	Z.100 4.14 page 112
Remote procedure context parameter	type	Use with care	7.6.2	Z.100 6.2.3 page 180
Remote variable context parameter	type	Not recommended	7.6.2	Z.100 6.2.6 page 181
Return	behaviour	Use freely	7.2.5.11	Z.100 2.6.8.2.4 page 61
Save	behaviour	Use freely	7.2.5.5	Z.100 2.6.5 page 55
Select	structure	Use with care	7.4.2.3	Z.100 4.3.3 page 98
Sequence	data	Use freely	7.5.2.7 7.5.2.3	Z.105 4.2.2 page 9 Z.105 II.3.2 page 37 Z.105 III page 51
Sequenceof	data	Use freely	7.5.2.3	Z.105 4.2.3 page 12 Z.105 II.3.3 page 38 Z.105 III page 51
Service	structure	Not recommended	7.2.3.5	Z.100 2.4.5 page 37 Z.100 6.1.1.4 page 170
Service definition based on service type	type	Do not use	7.6.1.3.4	Z.100 6.1.3.4 page 174
Set	data	Use freely	7.5.2.7 7.5.2.3	Z.105 III page 51
SET OF	data	Use freely	7.5.2.3	Z.105 II.3.4 page 40 Z.105 III page 51
Signal	behaviour	Use freely	7.2.4.4	Z.100 2.5.4 page 48
Signal context parameter	type	Use with care	7.6.2	Z.100 6.2.4 page 180
Signal list	structure	Use freely	7.2.4.5	Z.100 2.5.5 page 49
Signal refinement	structure	Do not use	7.3.2	Z.100 3.3 page 89
Signal route	structure	Use freely	7.2.4.2	Z.100 2.5.2 page 44

Concept	Category	Recommendation on Use	Internal Reference	External References
Sort (Data type in Z.100) assignment	data	Use freely	7.5.2.18	Z.105 4.1.1 page 6
Sort context parameter	data	Use with care	7.6.2	Z.100 6.2.9 page 182
Sort Expressions	data	Use freely	7.5.2	Z.105 4.2 page 7
Sort	data	Use freely	7.2.2.1 7.5.1.1	Z.100 5.2.1 page 116
Specialization	type	Use with care	7.6.3	Z.100 6.3 page 182
Spontaneous transition	behaviour	Do not use	7.2.5.6	Z.100 2.6.6 page 56
Start	behaviour	Use freely	7.2.5.2	Z.100 2.6.2 page 51
State (and multiple appearance)	behaviour	Use freely	7.2.5.3 7.4.4	Z.100 2.6.3 page 52
Stop	behaviour	Use freely	7.2.5.10	Z.100 2.6.8.2.3 page 60
String	data	Use freely	7.5.2.2	Z.100 Annex C.3 Z.105 2.4 page 4 Z.105 4.4.3 page 22
Struct	data	Use freely	7.5.2.7 7.2.5.1.1	Z.100 5.3.1.10 page 136 Z.100 5.3.3.6 page 152
Subrange	data	Use freely		Z.105 4.2.7 page 16
Subtypes	data	Use freely		Z.105 III page 52
Synonym	data	Use freely	7.5.2.10	Z.100 5.3.1.13 page 142 Z.100 5.3.3.3 page 150
Synonym context parameter	type	Use with care	7.6.2	Z.100 6.2.8 page 181
Syntype	data	Use freely	7.5.2.6	Z.100 5.3.1.9 page 132
System	structure	Use freely	7.2.3.2	Z.100 2.4 page 23 Z.100 2.4.2 page 28 Z.100 6.1.1.1 page 166
System definition based on system type	type	Use freely	7.6.1.3.1	Z.100 6.1.3.1 page 172
Tag	data	Use freely		Z.105 I.1 page 30 Z.105 III page 52
Task	behaviour	Use freely	7.2.6.1	Z.100 2.7.1 page 62
Text extension	presentation	Use freely	7.2.1.6	Z.100 2.2.7 page 22
Text symbol	presentation	Use freely	7.2.1.7	Z.100 2.2.8 page 22
Time	data	Use with care	7.5.2.3	Z.100 Annex C.12
Timer	behaviour	Use freely	7.2.7	Z.100 2.8 page 70
Timer active	behaviour	Use freely	7.5.2.19.5	Z.100 5.4.4.5 page 162
Timer context parameter	type	Use with care	7.6.2	Z.100 6.2.7 page 181
Transition	behaviour	Use freely		Z.100 2.6.8 page 58
Type	type	Use freely	7.6.1.1 7.6.1.2	Z.100 1.3.1 page 3 Z.100 6.1.1 page 166 Z.100 6.1.2 page 171
Useful type	data	Use freely		Z.105 III page 52
Value	data	Use freely	7.2.2.2	Z.100 2.3.3 page 23
Value returning procedure	behaviour	Use freely	7.5.2.20	Z.100 5.4.5 page 163
Value assignments	data	Use freely		Z.105 4.1.2 page 7
Value expression	data	Use freely		Z.105 4.4 page 19
Variable	behaviour	Use freely	7.2.5.1	Z.100 2.3.2 page 23 Z.100 2.6.1 page 50 Z.100 5.4.1 page 153 Z.100 5.4.2 page 154 Z.100 5.4.2.2 page 155 Z.105 4.1 page 6

Concept	Category	Recommendation on Use	Internal Reference	External References
Variable context parameter	type	Use with care	7.6.2	Z.100 6.2.5 page 181
View and Reveal	behaviour	Do not use	7.2.5.1.2 7.5.2.19.4	Z.100 5.4.4.4 page 162 Z.100 2.6.1.2 page 51
Virtual	type	Use with care	7.6.3.2	Z.100 6.3.2 page 183
Virtual transition/save	type	Use with care	7.6.3.3	Z.100 6.3.3 page 184
Visibility	presentation	Use freely	7.2.1.2	Z.100 2.2.2 page 16

A.2 Selection of MSC concepts

Table 2 identifies the MSC concepts specified in Z.120 and provides a classification and references for each.

Table 2: Classification of MSC concepts

Concept	Category	Recommendation on Use	Internal Reference	External References
Action	informal description	Use freely	8.3.8	Z.120 4.8 page 30
Alternative expression	composition	Use with care	8.4.3	Z.120 5.3 page 34
Axis	entity structure	Use freely	8.2.4 8.3.2	Z.120 4.2 page 17
Coevent	sequence	Use with care	8.4.1	Z.120 5.1 page 31
Comment	informal description	Use freely	8.2.3	Z.120 2.1 page 9 Z.120 2.3 page 10
Condition	sequence	Use freely	8.3.6	Z.120 4.6 page 25
Coregion	sequence	Use with care	8.4.1	Z.120 5.1 page 31 Z.120 6.11 page 53
Create	sequence	Use with care	8.3.9	Z.120 4.9 page 30 Z.120 4.9 page 30
Decomposition	composition	Do not use	8.4.2	Z.120 5.2 page 33 Z.120 6.13 page 54
Document	composition	Use freely	8.3	Z.120 3 page 11
Duration	sequence	Use freely	8.3.7	Z.120 4.7 page 29
End (HMSC)	sequence	Use freely	8.4.5	Z.120 5.5 page 39
Environment	entity structure	Use freely	8.3.4	Z.120 4.4 page 20
Event	sequence	Use freely	8.3.3 8.3.7	Z.120 4.1 page 13, Z.120 4.3 page 19
Exception expression	composition	Use with care	8.4.3	Z.120 5.3 page 34
Expression	composition	Use with care	8.4.3	Z.120 5.5 page 39
Found message	sequence	Use with care	8.3.3.2	Z.120 4.3 page 20
Frame	presentation	Use freely	8.3.4	Z.120 4.1 page 14
Gate	composition	Not recommended.	8.3.4	Z.120 4.1 page 13 Z.120 4.4 page 22
General order	sequence	Do not use	8.3.5	Z.120 4.5 page 24 Z.120 6.11 page 53 Z.120 6.12 page 54
Global condition	sequence	Use freely	8.3.6	Z.120 4.6 page 27
Head	entity structure	Use freely	8.2.4 8.3.2	Z.120 4.1 page 12
High-level MSC (HMSC)	composition	Use freely	8.4.5	Z.120 5.5 page 39
HMSC condition	composition	Use freely	8.4.5	Z.120 5.5 page 40
Identifier	presentation	Use freely	8.2.1 8.2.2	Z.120 3 page 12
In gate	composition	Not recommended	8.3.4	Z.120 4.4 page 22
Incomplete message	sequence	Use with care	8.3.3.2	Z.120 4.3 page 18
Infinity	composition	Use with care	8.4.3	Z.120 5.3 page 34
Inline expression	composition	Use with care	8.4.3	Z.120 5.3 page 33
Inline gate	composition	Not recommended	8.3.4	Z.120 4.4 page 22 Z.120 5.3 page 34
Inline order gate	composition	Not recommended	8.3.4	Z.120 4.4 page 23
Instance	entity structure	Use freely	8.3.2	Z.120 4.1 page 12 Z.120 4.2 page 16
Instance axis	entity structure	Use freely	8.2.4 8.3.2	Z.120 4.2 page 17
Instance decomposition	composition	Do not use	8.4.2	Z.120 5.2 page 33 Z.120 6.13 page 54

Concept	Category	Recommendation on Use	Internal Reference	External References
Instance event	sequence	Use freely	8.3.3 8.3.6 8.3.7 8.3.8 8.3.9 8.4.1	Z.120 4.1 page 13
Instance head	entity structure	Use freely	8.2.4 8.3.2	Z.120 4.1 page 13
Instance kind	entity structure	Use freely	8.3.2	Z.120 4.1 page 13
Kind	entity structure	Use freely	8.3.2	Z.120 4.1 page 13
Local condition	sequence	Use freely	8.3.6	Z.120 6.7 page 50
Loop expression	composition	Use with care	8.4.5	Z.120 5.3 page 34 Z.120 6.19 page 61
Lost message	sequence	Use with care	8.3.3.2	Z.120 4.3 page 20
Message	sequence	Use freely	8.3.3	Z.120 4.3 page 17
Message instance	sequence	Use freely	8.3.3	Z.120 4.3 page 18
MSC document	composition	Use freely	8.3	Z.120 3 page 11
Node	composition	Use freely	8.4.5	Z.120 5.5 page 40
Note	informal description	Use freely	8.2.1	Z.120 2.1 page 8
Operand	composition	Use with care	8.3.8	Z.120 5.3 page 35
Optional expression	composition	Use with care	8.4.3	Z.120 5.3 page 34
Order gate	composition	Not recommended	8.3.4	Z.120 4.1 page 13 Z.120 4.4 page 22
Orderable event	sequence	Not recommended	8.3.5	Z.120 4.1 page 13
Ordered event	sequence	Not recommended	8.3.5	Z.120 4.5 page 25
Out gate	composition	Not recommended	8.3.4	Z.120 4.4 page 22
Parallel expression	composition	See expression.	8.4.3	Z.120 5.3 page 34
Parameter	informal description	Use freely	8.3.3	Z.120 4.3 page 18
Qualifier	informal description	Use freely	8.3	Z.120 3 page 12
Reference	composition	Not recommended (except for HMSCs)	8.4.4	Z.120 4.3 page 18 Z.120 5.4 page 36 Z.120 5.4 page 37 Z.120 6.17 page 59
Reference identification	composition	Not recommended	8.2.2 8.4.4	Z.120 5.4 page 37
Reset	sequence	Use freely	8.3.7	Z.120 4.7 page 28
SDL document identifier	informal description	Use freely	8.3 8.3.3.3	Z.120 3 page 12
SDL reference	informal description	Use freely	8.3 8.3.3.3	Z.120 3 page 12
Set	sequence	Use freely	8.3.7	Z.120 4.7 page 28
Shared condition	sequence	Use freely	8.3.6	Z.120 4.6 page 26 Z.120 5.3 page 34 Z.120 6.8 page 51
Shared expression	composition	Use with care	8.4.3	Z.120 5.3 page 33
Shared MSC reference	composition	Not recommended	8.4.4	Z.120 5.4 page 37
Start (HMSC)	sequence	Use freely	8.4.5	Z.120 5.5 page 39
Stop	sequence	Use with care	8.3.10	Z.120 4.10 page 31
Substitution	composition	Do not use	8.4.4	Z.120 5.4 page 37 Z.120 5.4 page 37
Substructure	composition	Do not use	8.4.2	Z.120 5.2 page 33
Timeout	sequence	Use freely	8.3.7	Z.120 4.7 page 28
Timer	sequence	Use freely	8.3.7	Z.120 4.7 page 27

A.3 List of supplementary guidelines

Associated with each SDL and MSC concept identified in this document there is a recommendation for its use within telecommunications standards. In addition, there are a number of supplementary guidelines offered throughout the document and these are summarised below:

PRINCIPLES AND GENERAL GUIDELINES

1 The specific behaviour described using SDL in a standard is indicative only of the relationship between incoming and outgoing signals and should not be treated as binding at the language or structure level

SDL IN EUROPEAN TELECOMMUNICATION STANDARDS

2 A standard should make it clear whether the SDL specification is normative or informative

3 If the normative description of a protocol is the SDL, the duplication of information should be limited by using the text only to bring together requirements that are distributed over several SDL diagrams

NORMATIVE INTERFACES AND REQUIREMENTS

4 Normative communications paths should be marked with the comment “Normative

5 Processes should be defined within an explicit block and system

6 Messages carried across normative communications paths should be specified in SDL signal definitions and marked as normative requirements

7 All data types specified in the parameter lists of normative signals should be defined using ASN.1 or SDL data formalisms and marked as normative requirements themselves

8 MSCs should be used to define the sequence of protocol messages required for normal operation and the sequences expected in the most significant exceptional cases

9 The complete behaviour in terms of all observable message sequences should be defined using SDL behaviour descriptions

10 The MSC description of a sequence of messages should be consistent with the SDL description of the same sequence

SPECIFICATION AND DESCRIPTION LANGUAGE CONCEPTS

11 A bounded range of possible values should be specified for all data types (SDL sorts), particularly implicit types such as INTEGER and REAL

12 All composite data types should be initialized, preferably by a default initialization defined in the data type

13 Macros should only be used for some limited, well thought out purposes where their use can be proven to bring benefits and the macros have be shown to produce the required functionality

14 Structural macros should not be used because they do not help in understanding the specification of structure

15 In standards where it is necessary to specify multiple or complex options, the use of system and block types may be easier to understand than the use of numerous select symbols

16 Optional transitions should be used freely to model the differences in behaviour between implementation options

17 Implicit transitions should be avoided in the specification of behaviour in a standard

CONCEPTS IN MSC

18 MSCs should be used to supplement the SDL to give descriptions of some valid sequences of behaviour

19 MSCs should be used to give at least one example of message exchanges for each required system function and should give examples of message exchanges in exceptional conditions

20 The characters '{', '}', '[', ']', '|' (vertical bar) and space should not be used in MSC names, and the use of SDL keywords for names should be avoided

21 Names in an MSC should be the same as the names of corresponding entities in the SDL

22 Attention should be given to the order of MSC instances left to right and avoiding messages that cross instances

23 Where paging is used, the MSC pages should be numbered according to Z.120

24 Basic MSC should normally be used in combination with an HMSC

25 A basic MSC should be given a name that is meaningful for the sequence

26 The form of heading used should be consistent in all MSC diagrams

27 Message overtaking should be avoided, except for sequences where it is essential to show the behaviour when overtaking takes place

28 Lost and found message should be used with care

29 MSC descriptions that cover behaviour which does not correspond directly to the SDL should be clearly annotated

30 There should always be an explicit message symbol for each gate of a referenced diagram or inline expression in the enclosing diagram

History

Document history	
Sept 1997	1st draft derived from ETS 300 414 without change except for new Style sheet and sub-clause renumbering
July 1998	1st complete stable draft of EG with all new contents.
July 1998	Version 0.2.0 taking into account comments from CNET and made available for MTS approval