

European Telecommunications Standards Institute  
**MTS#33**  
**24 to 25 October 2001**  
**Sophia-Antipolis**

**Source: STF188**

**Title: REG/MTS-00072 'Guidelines for the use of SDL as  
a descriptive tool' (incomplete draft)**

**Date: 12 October 2001**

**Document for: Discussion**

**Agenda item: 5.5**



# REG/MTS-00072 V1.1.2 (October 2001)

---

*ETSI Guide*

---

## **Methods for Testing and Specification (MTS); Guidelines for the use of SDL as a descriptive tool**



---

Reference

REG/MTS-00072

---

Keywords

SDL, MSC, ASN.1, UML, methodology

***ETSI***

650 Route des Lucioles  
F-06921 Sophia Antipolis Cedex - 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

---

***Important notice***

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <http://www.etsi.org/tb/status/>

If you find errors in the present document, send your comment to:  
editor@etsi.fr

---

***Copyright Notification***

No part may be reproduced except as authorized by written permission.  
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute yyyy.  
All rights reserved.

# Contents

Intellectual Property Rights .....	7
Foreword .....	7
1 Scope .....	9
2 References.....	9
3 Definitions and abbreviations .....	9
3.1 Definitions .....	9
3.2 Abbreviations.....	10
4 Introduction.....	10
5 Using specification languages in protocol standards .....	11
5.1 Introduction.....	11
5.2 Layered protocols .....	12
5.3 Developing a protocol specification.....	12
5.3.1 Specifying requirements.....	12
5.3.2 Developing a logical model.....	12
5.3.3 Developing a physical model .....	12
6 Naming Conventions .....	15
6.1 General .....	15
6.1.1 Case sensitivity .....	16
6.1.2 Length of names .....	16
6.1.3 Reserved words.....	16
6.2 SDL and MSC .....	17
6.2.1 Use of non-significant characters .....	17
6.2.2 Multiple use of names.....	17
6.2.3 Making names meaningful.....	18
6.2.3.1 Block, process and instance names .....	18
6.2.3.2 Procedure, operator and method names .....	19
6.2.3.3 Signal names .....	19
6.2.3.4 Signal List and interface names .....	19
6.2.3.5 SDL State names .....	20
6.2.3.6 Names of Variables and Constants .....	20
6.2.3.7 Timers.....	20
6.3 Data types .....	20
7 Presentation and layout of diagrams .....	21
7.1 The general flow of behaviour across a page.....	21
7.2 Diagrams covering more than one page.....	23
7.2.1 SDL behaviour diagrams .....	23
7.2.2 UML activity diagrams .....	28
7.2.3 Symbols common to all pages.....	28
7.3 Text extension symbols .....	29
7.4 Alignment and orientation of symbols .....	29
7.4.1 Alignment .....	29
7.4.2 Orientation.....	31
8 Structuring behaviour descriptions .....	31
9 Using procedures, operators methods and macros.....	31
10 Using decisions.....	31
11 System structure, communication and addressing .....	31
12 Specification and use of data .....	31
13 Using Message Sequence Charts (MSC).....	31
13.1 Introduction.....	31

13.2 Relationship between MSC and SDL .....	32
13.3 Presentation and layout .....	32
13.3.1 Annotations .....	32
13.4 Naming and scope .....	33
13.5 MSC document .....	33
13.6 Structuring .....	34
13.6.1 Architecture .....	34
13.6.1.1 Instance .....	34
13.6.1.2 Instance decomposition .....	35
13.6.1.3 Dynamic instances .....	35
13.6.1.4 Environment .....	35
13.6.2 Behaviour .....	36
13.6.2.1 High-level MSC (HMSC) .....	37
13.6.2.2 MSC reference in basic MSC .....	38
13.6.2.3 Inline expression .....	39
13.7 Data .....	40
13.8 Message .....	40
13.8.1 Incomplete messages .....	42
13.9 Condition .....	42
13.10 Action .....	43
13.11 Timer .....	43
13.12 Control Flow .....	45
13.13 Time .....	45
13.14 General ordering and coregion .....	46
<b>Annex A (informative): Reserved words .....</b>	<b>48</b>
A.1 .....	SDL 48
A.1.1 Keywords .....	48
A.1.2 Predefined words .....	49
A.2 .....	MSC 49
A.3 .....	ASN.1 50
A.4 .....	UML 50
<b>Annex B (informative): Summary of guidelines .....</b>	<b>51</b>
History .....	53

---

## Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<http://www.etsi.org/ipr>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

---

## Foreword

This ETSI Guide (EG) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).





---

# 1 Scope

The present document establishes a set of guidelines for the formal use of Specification and Description Language (SDL) for descriptive, rather than detailed design, purposes. It also provides some guidance on the use of Message Sequence Charts (MSC), Abstract Syntax Notation 1 (ASN.1) and the Unified Modeling Language (UML) when used in conjunction with SDL. The objective of the guidelines is to provide assistance to rapporteurs of protocol standards so that the SDL that appears in ETSI deliverables is formally expressed, easy to read and understand and at a level of detail consistent with other standards. The present document applies to all standards that make use of SDL to specify protocols, services or any other type of behaviour.

Users of the present document are assumed to have a working knowledge of SDL. It should not be considered to be an SDL tutorial and should be read in conjunction with EG 201 383[1] and EG 201 015 [2].

---

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

- [1] EG 201 383 (V1.1): "Methods for Testing and Specification (MTS); Use of SDL in ETSI deliverables; Guidelines for facilitating validation and the development of conformance tests".
- [2] EG 201 015 (V1.2): "Methods for Testing and Specification (MTS); Specification of protocols and Services; Validation methodology for standards using SDL; Handbook".
- [3] EG 201 872 (V1.2): "Methods for Testing and Specification (MTS); Methodological approach to the use of object-orientation in the standards making process
- [4] ITU-T Recommendation Z.100 (1999): "Specification and description language (SDL)".
- [5] ITU-T Recommendation Z.105 (1999): "SDL combined with ASN.1 (SDL/ASN.1)".
- [6] ITU-T Recommendation Z.109 (1999): "SDL combined with UML"
- [7] ITU-T Recommendation Z.120 (1999): "Messages sequence chart".
- [8] ITU-T Recommendation X.680 (1997): "Information technology - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

---

# 3 Definitions and abbreviations

## 3.1 Definitions

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

**data type:** set of data values with common characteristics (equivalent to the ITU-T Recommendation Z.100 [4] 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 option:** statement in a standard that may or may not be supported in an implementation

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

**validation:** 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:** 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
MSC	Message Sequence Chart
Pid	Process Identity
SDL	Specification and Description Language
UML	Unified Modeling Language

---

## 4 Introduction

The ITU-T Specification and Description Language (SDL) defined in ITU-T Recommendation Z.100 [4] is a powerful tool for specifying the essential requirements of standardized protocols or services. The level of formality with which the SDL in a standard is expressed can depend on a large number of factors such as the size and complexity of the system to be standardized and the skills and experience of the standards writers. The specification of a protocol or service as a complete formal model enables the validation of the standard before approval and publication. However, well-constructed, formal SDL has a valuable role to play in providing a simple illustration of the process-related aspects of a standardized system.

SDL is most often found in protocol standards with some associated ASN.1 and MSC. Additionally, as the language specifications converge, SDL is also likely to be used in conjunction with UML in standards. It is, therefore, sensible to consider the relationships between all of these languages and notations when offering guidelines on SDL. The present document is concerned primarily with the development of easy-to-read SDL but also provides some guidance on the use of ASN.1, MSC and UML where this overlaps with the use of SDL.

**NOTE:** Although in the strictest sense SDL, MSC and UML are considered to be languages while ASN.1 is a notation, for the sake of simplicity the term "notation" is used throughout the present document to denote both languages and notations.

In order to gain the maximum benefit from the use of descriptive SDL, it is necessary for a consistent approach to be taken in its specification by all rapporteurs. In the context of the present document, the term "descriptive SDL" can be taken to mean SDL which is:

- formally expressed:
  - uses only language constructs and symbols that are defined in ITU-T Recommendations Z.100 [4] and Z.105 [5];
- complete:
  - a full model with System, Block, Process and Procedure diagrams as necessary;
  - has a comprehensive data specification using SDL data or, preferably, ASN.1;
  - uses "correct" SDL;
  - is not necessarily a simulation or validation model;
- easy to read and understand:

- uses meaningful names and identifiers;
- the model structure complements the specification;
- has an open layout which requires a minimum of effort to follow;
- the "how" is hidden from the "what";
- complex programming structures are avoided;
- extensive comments annotate the model;
- at a level of detail consistent with other standards:
  - is not over-engineered;
  - is not an implementation model;
  - does not constrain implementations to methods and techniques which are beyond the scope of the standard.

By following the set of simple guidelines presented in the present document, it will be possible for the following benefits to be realized:

- Comprehension of the specification can be improved;
- Ambiguity can be avoided in the translation of the descriptive SDL into a validation model.

Achieving consistency in the presentation and level of detail specified across a wide range of standards is one of the keys to maintaining the perceived quality of ETSI's products.

The guidelines for the use of SDL for descriptive purposes are grouped in the present document according to the following broad classifications:

- naming conventions;
- presentation and layout of SDL processes;
- diagram structures;
- the use of procedures and operators;
- the use of decisions;
- communications and addressing;
- the specification and use of data;
- the use of Message Sequence Charts (MSC) in association with SDL.

Each of the guidelines is highlighted within the document in *bold and italic text*. They are all collected together in tabular form in Annex B.

---

## 5 Using specification languages in protocol standards

### 5.1 Introduction

This chapter gives some consideration to the process of standardizing communication protocols so that guidance can be given on where SDL, ASN.1, MSC and UML can be used effectively.

## 5.2 Layered protocols

There are numerous approaches to the design of communications protocols, each of which is valid in the situation that it is used. Probably the most well known and well used is the ISO layered model or a derivative of it where a protocol system is segmented into distinct logical layers with distinct responsibilities.

The communication between peer layers in this logical model never takes place directly but is achieved through the services of the lower layer. However, this peer-level communication is often specified in a standard without consideration of the signaling between layers. The interface between two adjacent layers is usually called the Service Access Point (SAP) although other terms such as user access and network access are also used. Protocol standards will, in most cases, be considerably simpler if they are restricted either to horizontal communication (peer-to-peer) or vertical communication (inter-layer). Mixing the two can lead to a confusing specification which is difficult to understand.

## 5.3 Developing a protocol specification

For many years, protocol standards have been prepared using the three-stage process described in ITU-T Recommendation I.130. Although the detailed practices specified in this document might now be considered to be out of date and its use is not as widespread as it once was, the underlying method upon which it is based is still relevant as good engineering design practice. Simply put, this is:

1. Specify requirements from the user's perspective;
2. Develop a logical model to meet those requirements;
3. Develop a physical specification of the protocol.

### 5.3.1 Specifying requirements

Specifying a protocol without first evaluating what it is intended to achieve and what constraints are to be applied to it will almost certainly end in a poor specification. These requirements can easily be expressed using free text and some informal diagrams but the use of UML for this purpose (as described in EG 201 872) means that this information can be checked by automatic tools and used as input to the later stages of specification.

At this stage of the specification, there should be no need to consider the possible physical architecture of any system implementing the protocol. Requirements should be expressed, as far as possible, entirely from the user's perspective (although the "user" may be a terminal or network application acting on behalf of a human user)

### 5.3.2 Developing a logical model

Before considering the physical specification of a protocol, there are benefits to be gained by specifying a model based on logical blocks so that the flow of information necessary for meeting the specified requirements can be defined without concern for the detailed format that such information should take. The identification of possible normative interfaces between blocks is also simpler without the constraints imposed by a specific physical architecture.

The overlap between UML and both SDL2000 and MSC2000 is such that all of these languages are suitable for this level of specification. In fact, it is unlikely that models developed in either UML or SDL2000 with MSCs would be appreciably different.

Once the logical model is complete, it is necessary to specify a physical model upon which "real" implementations of the protocol standard can be based. This model should not, in most cases, be a detailed implementation model but should be constrained to specify the minimum protocol requirements to guarantee interworking between modules from different suppliers. A good first step towards this physical model is to define a set of legitimate scenarios for the distribution of the logical blocks within a set of physical entities. Textual tables have traditionally been used quite effectively for this purpose but UML deployment diagrams can provide a graphical means of presenting these requirements.

### 5.3.3 Developing a physical model

If systems implementing a standardized protocol are to inter-work without problems, it is necessary to specify the detailed content and format of signals between physical entities and the temporal relationships that must exist between

these signals. For this specification to be complete and accurate, it may be necessary to describe the behaviour of the physical entities which make up the protocol system.

ASN.1 is generally accepted as the notation to be used within protocol standards for the definition of signal data structures. Although it is not a particularly intuitive notation to use, it has the significant benefit that there are a number of standardized sets of rules (for example, Basic Encoding Rules – BER and Packed Encoding Rules – PER) for encoding ASN.1 structures into "concrete" data items with more or less efficiency. In those cases where even PER does not produce a compact enough encoding, Encoding Control Notation (ECN) specified in TS 101 969 enables users to define and use their own encoding rules in a standardized form. A further benefit of using ASN.1 is that ITU-T Recommendation Z.105 specifies exactly how ASN.1 is used in conjunction with SDL so that data items defined in an ASN.1 module can be used directly in the SDL associated with that module.

The following simple example uses ASN.1 to specify the structure of an address which comprises a length parameter and the address value itself

```
Address ::= SEQUENCE {
    length  BIT STRING(SIZE(8)),
    value   OCTET STRING }
```

Message Sequence Charts (MSCs) are an ideal notation for describing signal flows and a simple example is shown in Figure 1

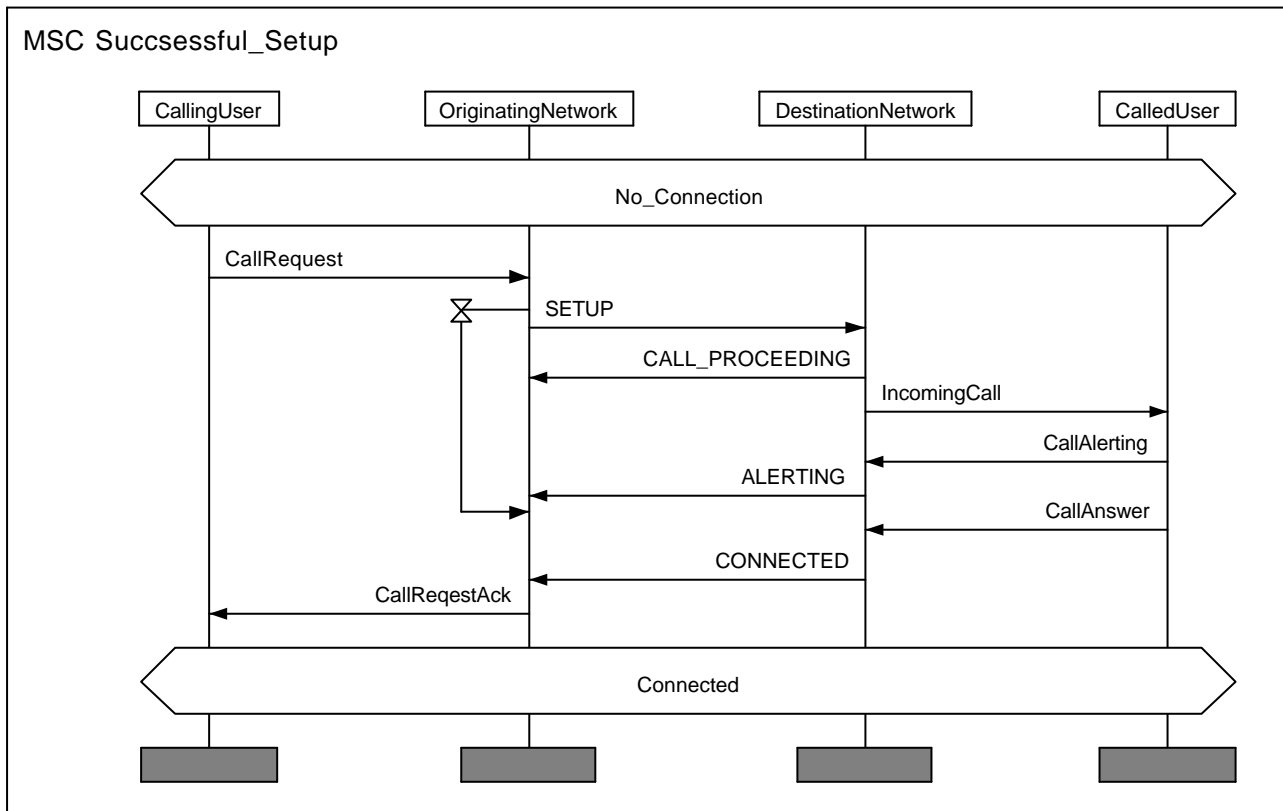
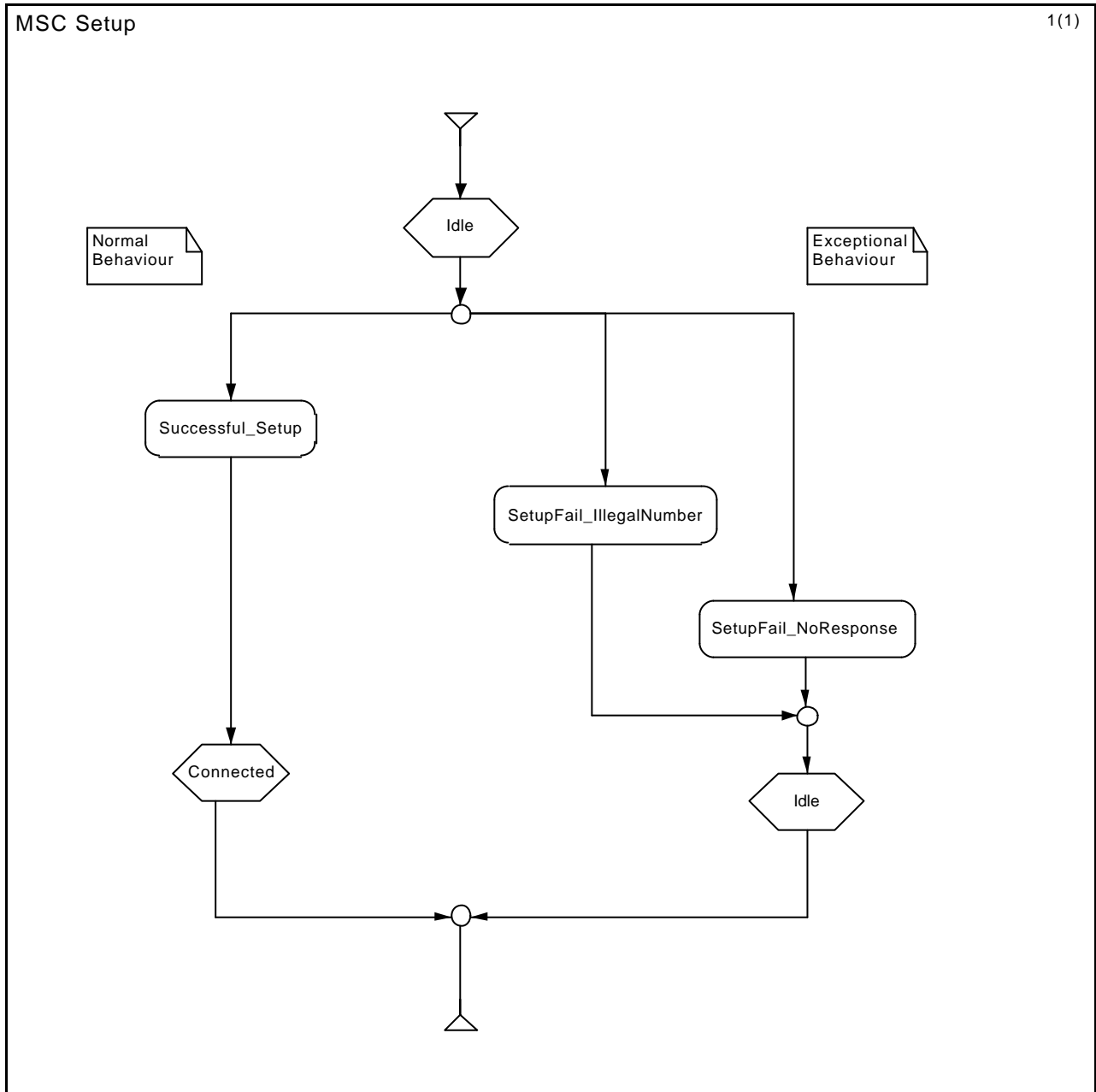


Figure 1: Example of a simple MSC

In anything but the simplest protocol, it is not possible to show all of the possible sequences of signals. It is, therefore, quite acceptable to use MSCs to illustrate only a representative sample of sequences. These examples should specify a reasonable range of successful and unsuccessful situations to enable readers to make an informed judgement of what the flows would be in other unspecified scenarios.

High level MSC (HMSC) diagrams can be used to provide an overview of the relationships between detailed sequences of signals in more complex scenarios. The simple example in Figure 2 shows how an HMSC can be used to segregate normal behaviour from exceptional behaviour.



**Figure 2: Example HSMC**

In order to complete the picture of possible signal sequences, the behaviour of each physical entity needs to be specified and SDL is an ideal graphical language for this purpose. By using SDL's language features to specify system architecture, communication paths, signals and behaviour and using ASN.1 to define signal parameter structures, it is possible to build a complete model. This can then be used to improve the quality of the overall specification by simulating and testing a range of possible scenarios.

The present document offers a number of guidelines on the use of SDL with ASN.1, MSC and UML to produce protocol standards which are easy to read and understand and which unambiguously express the requirements for an implementation.

## 6 Naming Conventions

### 6.1 General

In common with most modern programming languages, SDL, MSC, ASN.1 & UML permit the use of alphanumeric names to identify individual entities within a specification. Examples of entities that can be identified in this way are:

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>- SDL</li> <li style="padding-left: 20px;">- blocks</li> <li style="padding-left: 20px;">- procedures</li> <li style="padding-left: 20px;">- signals</li> <li style="padding-left: 20px;">- variables and constants</li> <li>- MSC</li> <li style="padding-left: 20px;">- instances</li> <li style="padding-left: 20px;">- messages</li> <li style="padding-left: 20px;">- timers</li> <li style="padding-left: 20px;">- conditions</li> </ul> | <ul style="list-style-type: none"> <li>- ASN.1</li> <li style="padding-left: 20px;">- type references</li> <li style="padding-left: 20px;">- identifiers</li> <li style="padding-left: 20px;">- value references</li> <li style="padding-left: 20px;">- module references</li> <li>- UML</li> <li style="padding-left: 20px;">- classes &amp; objects</li> <li style="padding-left: 20px;">- states</li> <li style="padding-left: 20px;">- events</li> <li style="padding-left: 20px;">- attributes</li> </ul> |
|---|--|

It is likely that protocol standards will incorporate SDL, MSC, ASN.1 or UML specifications of structure and behaviour. Frequently, two or more of these are used in combination within the same standard and in these cases it is certain that some entities defined in one notation will also be used in another. Examples of these are:

- ASN.1 data types which are used by SDL;
- SDL processes which are mapped to MSC instances.

Although the lexical rules in each notation are similar, they are by no means identical. Table 1 identifies the most significant differences in the construction of identifiers within these four languages and notations.

**Table 1: Significant differences in the lexical rules of SDL, MSC, ASN.1 and UML**

Notation	Significant differences
SDL	<ul style="list-style-type: none"> <li>- name may be hyphenated over more than one line using the underscore ("_") character</li> <li>- names may contain non-printing characters (which are ignored) only if preceded by "_" (which is also ignored)</li> <li>- names may contain "_" but not "-"</li> </ul>
MSC	Same as SDL
ASN.1	<ul style="list-style-type: none"> <li>- names are restricted to a single line</li> <li>- names may only contain printing characters</li> <li>- names may contain "-" but not "_"</li> </ul>
UML	<ul style="list-style-type: none"> <li>- names are restricted to a single line</li> <li>- names may only contain printing characters</li> <li>- the use of "_" and "-" in names is not specified and are most likely to be tool dependant</li> </ul> <p>NOTE: In practice, the lexical rules of UML are likely to vary according to the tool used and the target software language)</p>

The choice of names is likely to be affected by the individual application but <sup>(62)</sup>*a naming convention that can be applied consistently to each notation used should be chosen*. Taking this approach will help to avoid ambiguities when names need to be modified to comply with conflicting lexical rules in each language and notation used. Even in those instances where it is planned to use only one notation, consideration should also be given to the rules of the others when specifying a naming convention as one or more of these may be used to augment the specification at a later stage

One of the most common such conflicts occurs between ASN.1 and SDL where the use of dash ("-") characters is permitted in ASN.1 but not in SDL while underscores ("\_") may be used in SDL but not in ASN.1. ITU-T Recommendation Z.105 specifies that a dash character within an ASN.1 name is mapped to an underscore when it is converted to SDL. This is a reasonable approach but it still leaves a visible difference between an ASN.1 type name and its corresponding SDL type. For example:

*Setup-contents* in ASN.1 is equivalent to *Setup\_contents* in SDL.

***While it is acceptable to use the underscore character to delineate words within most SDL entity names, it is advisable to avoid the use of the dash character in ASN.1 types and values in order to avoid conflicts and mis-interpretation in the associated SDL***

### 6.1.1 Case sensitivity

SDL, MSC, ASN.1 and UML are all sensitive to the case of characters within names. As an example, the name "ABC" is not the same as "AbC" or "Abc". The ASN.1 syntax goes further by specifying that names beginning with an upper-case letter should be interpreted as type references and that those beginning with lower-case letters should be interpreted as value references or identifiers such as information elements in a SEQUENCE or CHOICE. Although the case of the first character of a name does not have the same syntactical significance in either SDL or MSC, it is a useful way of distinguishing between types and values, particularly when used in conjunction with ASN.1. However, <sup>(62)</sup>***the general use of names which differ only in character case to distinguish between entities should be avoided.***

Although errors are likely to be detected by automatic syntax checking tools, <sup>(62)</sup>***care should be taken to ensure the consistent use of character case within names throughout an ASN.1, SDL, MSC or UML specification.***

The capitalization of the first character of each word within a name is an acceptable method of delineation between the component parts of the name.

Example: The procedure name, "DeliverMessageContents" can easily be interpreted to imply that the purpose of the procedure is to "Deliver the Message Contents".

Although it works well in many cases, this method can result in names that are quite difficult to read if they contain acronyms or larger numbers of short words. Examples of these are:

InvokeCCBSSupplementaryService;

AddOneToTheFirstItemOfOldData.

### 6.1.2 Length of names

The syntaxes of SDL, MSC, ASN.1 and UML place no restrictions on the number of characters that may be included in names but, in practice, the limits associated with the target language (e.g., Java or C++) should be respected. It is also worth noting that very long names can often be difficult to read. It is not possible impose a strict rule on the length of names but, as a general guideline, <sup>(62)</sup>***names of less than 6 characters may be too cryptic and names of more than 30 characters may be too difficult to read and assimilate.***

### 6.1.3 Reserved words

Although SDL, MSC, ASN.1 & UML all permit great flexibility in the use of names, there are certain reserved words which are keywords of the languages themselves and which, consequently, cannot be used as names. Lists of these reserved words can be found in Annex A.

NOTE: SDL keywords may be either all upper-case or all lower-case. Keywords using mixed case are not considered to be reserved words. For example, both "procedure" and "PROCEDURE" are SDL reserved words but "Procedure" is not.

The use of reserved words from one notation can be legitimately used as names within a specification based upon another but to avoid any conflict across specifications using multiple notations, <sup>(62)</sup>***the reserved words of all notations used within a standard should be avoided as defined names in each of the individual parts.***



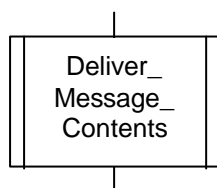
## 6.2 SDL and MSC

### 6.2.1 Use of non-significant characters

It is permissible to split a name across more than one line by introducing an underscore followed by a sequence of spaces and/or the *carriage-return* and *line-feed* control characters. So, the procedure name "DeliverMessageContents" in the example above could also be expressed as:

```
Deliver_
Message_
Contents
```

This is a very convenient notation when trying to fit a long name into a graphical symbol, thus:



It is worth noting that the underscore character is only insignificant when used as a hyphenation symbol and that the name:

```
DeliverMessage
```

is not the same as:

```
Deliver_Message
```

although it is identical to

```
Deliver_
Message
```

When a name using underscores to separate words is wrapped over more than one line, it is necessary to include two underscore characters where the hyphenation occurs, thus:

```
Deliver_
_Message
```

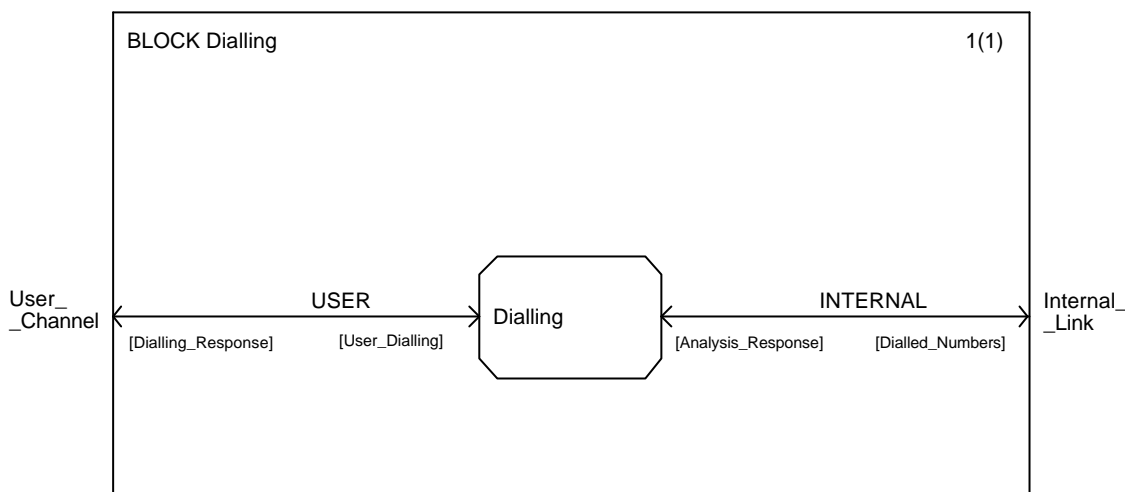
<sup>(62)</sup>***Readability is improved if the same convention for separating words within names is used throughout a specification.*** The one case where a combination of methods is recommended is in the use of acronyms within names that use capitalisation as the method of separation. An underscore on each side of the acronym clearly delineates it from the remainder of the name, thus:

```
Invoke_CCBS_SupplementaryService
```

<sup>(62)</sup>***In most cases an underscore character between each word removes any possibility of misinterpretation and this is the approach that is recommended.***

### 6.2.2 Multiple use of names

SDL permits entities belonging to different classes to be given the same name. As an example, it is syntactically correct for a process within a block named "Dialling" also to be given the name "Dialling" (see Figure 3). In addition, because of the scoping rules of the language, it would be possible for a process within another block in the same system to be named "Dialling".



**Figure 3: Example of a block and a process with the same name**

In many protocol standards, particularly those specifying supplementary services, the system comprises a small number of blocks, each of which contains only one process. In such situations, the use of the same name for the block and for its single process is valid but, as SDL allows it, a better approach may be to omit the block altogether as shown in chapter 11.

*In more complex models where each block is made up of a number of processes, the use of the same name for a block and one of its constituent processes is likely to cause confusion and should be avoided.*

Similar problems can also exist in the re-use of single names for multiple entities. For example, it is possible to have the same name for a signal list and for one of its constituent signals. As a general guideline, <sup>(62)</sup>*the use of a single name for multiple purposes should be avoided wherever possible.*

### 6.2.3 Making names meaningful

The freedom and flexibility allowed in the construction of names can be used to great benefit in improving the readability of a specification. If there is an entity whose function is to represent an alarm clock then it can be called "Alarm\_Clock" and there are no constraints to force the use of a more cryptic name such as "Alm\_Clk". However, this freedom can be abused and it would be quite legitimate for the alarm clock to be given the name "The\_Thing\_Beside\_The\_Bed\_That\_Makes\_A\_Loud\_Noise\_In\_The\_Morning" which is equally as unacceptable as the cryptic style.

Although it can appear useful during the development of a protocol standard, <sup>(62)</sup>*the addition of project-specific prefixes or suffixes can make meaningful names appear cryptic and should be used with great care.*

Apart from the general recommendations above, certain specific guidelines apply to each group of identifiable entities.

#### 6.2.3.1 Block, process and instance names

<sup>(62)</sup>*By giving blocks, processes and MSC instances names that represent the overall role that they play within the system, it is possible to distinguish process names from procedure names. If carefully chosen, they can help to link the SDL and MSC back to the corresponding subclauses in the text description.* Examples are:

```
originating_PINX;
Scenario_Management;
Functional_Entity_FE2;
alarm_clock.
```

As can be seen, these names are all *nouns* which indicate the general function of the process.

### 6.2.3.2 Procedure, operator and method names

Procedures, operators and methods (SDL operations) are key elements in breaking a complex process down into meaningful layers (see subclause 8.1). For this to be effective, <sup>(62)</sup>*the name chosen for an SDL operation should indicate the specific action taken by the operation*. Examples are:

```
Extract_Calling_Number_From_SETUP;
get_user_profile_from_database;
Send_Response;
ring_alarm_bell.
```

The names chosen here are all *verb phrases* indicating the specific activity to be carried out by the operation.

### 6.2.3.3 Signal names

There are often constraints on the length of signal names as they usually have to appear in quite small spaces within SDL symbols. It is, therefore, more difficult to arrive at meaningful names for them. However, poor naming of signals can make SDL and MSC very difficult to read, even when most other aspects are well presented. For example, the name "Rep\_Sgl\_Err" could easily be interpreted to mean:

```
Report Signal Error;
Report Single Error;
Repeat Signal Error;
Repeat Single Error.
```

The obvious approach is to express the name in full as, for example, "Report\_Signal\_Error" but this, again, is quite long. The problem can be overcome by using unambiguous abbreviations or abbreviations that are in common use. In the example above, "Err" is generally accepted as meaning "Error". Also, changing "Sgl" to "Sig" would make it much clearer that it was an abbreviation for "Signal" not "Single". <sup>(62)</sup>*If possible, it is advisable to leave at least one significant word in the name unabbreviated as this can help to provide the context for interpreting the remaining abbreviations*. So the example above would be acceptable if expressed as "Report\_Sig\_Err".

### 6.2.3.4 Signal List and interface names

SDL provides two mechanisms for collecting signals together into named logical groups. These are *signal lists* and *interfaces* as described in ???. For the purpose of defining names, these two can be treated identically.

In order to improve clarity, it is often advisable to group signals into interfaces or signal lists according to their capabilities and, consequently <sup>(62)</sup>*the name chosen for an interface or signal list should indicate the general function of the grouped signals*, for example:

```
UNI_Messages;
Mobility_Management;
user_input.
```

As an alternative and particularly in simple specifications <sup>(62)</sup>*where all signals between one block or process and another can be logically grouped together, signal list names can be chosen to indicate the origin and the destination of the associated signals*. Examples of this approach are as follows:

```
home_PINX_to_visitor_PINX;
HLRA_to_HLRB;
localExch_to_user;
between_AccessManagement_and_CallControl.
```

### 6.2.3.5 SDL State names

In most protocol standards, the SDL specification includes a large number of states and it is often tempting to assign cryptic and sequential names such as "state\_5" or "N3". Taking the time to formulate meaningful names for each state can add significantly to the readability of an SDL specification.

*(62) A state name should clearly and concisely reflect the status of the process while in that state.* Examples of such names are:

Idle;  
 Wait\_For\_SETUP\_Response;  
 Timing\_Signal\_Delay.

*(62) If it is important to number states then this should be done in conjunction with meaningful names* such as:

Releasing\_01;  
 Timing\_Response\_4.

### 6.2.3.6 Names of Variables and Constants

It is more difficult to specify some simple guidelines for the construction of names for variables and constants as they have widespread and diverse uses. It is still important to ensure that the name is meaningful in the context of the SDL specification. *(62) The name chosen for a variable should indicate in general terms what it should be used for.* For example:

SETUP\_message\_contents;  
 User\_Input;  
 Alarm\_Time.

*(62) Names used to identify constants can be more specific by indicating the actual value assigned to the constant.* For example:

User\_Not\_Known;  
 Twenty\_Five;  
 Characters\_A\_To\_Z.

### 6.2.3.7 Timers

Although the use of meaningful timer names, such as Response\_Sanity\_Timer, would improve the overall readability of a specification, it has become accepted practice to use the shorthand T1, T2, T3 etc. for timers within standards for protocols. To avoid confusion, the "Tn" notation should be used when naming timers unless an opportunity arises to use extended names in a completely new project where the use of the shorthand is not already established.

## 6.3 Data types

The definition of the ASN.1 notation, ITU-T Recommendation X.680 [?], specifies that type references must begin with an upper-case character and that value references and identifiers must begin with a lower-case character. When using ASN.1 in protocol standards, it has become the convention that a value reference uses the same name as its associated type reference (except where there are more than one value references derived from the same type reference) but that one is distinguished from the other by the case of its first character, thus:

```

-- Example of the use of identifiers with type references
Dog      ::= SEQUENCE {
            breed      Breed,
            name       Name }

Breed    ::= ENUMERATED {
            poodle,
            spaniel,
            alsation,
            boxer }

Name     ::= PrintableString

-- Example of the use of a value reference with a type reference
dogID   Name ::= "Rover"

```

For readability the name *breed* is preferable to *bREED*, even though the latter is, strictly speaking, permissible.

Although all data types associated with normative signals will usually be defined in ASN.1, other types can be specified using SDL's own data language features. For the sake of consistency with ASN.1, *the names of SDL data types should be capitalized while the names of literals and synonyms should begin with a lower-case character.*

---

## 7 Presentation and layout of diagrams

The syntaxes of both SDL and UML allows great freedom in the presentation and layout of both text and graphical symbols. Good presentation can considerably improve the readability of a specification whereas bad presentation can render it unintelligible. It is also worth noting that a single error resulting from the misunderstanding of a poorly presented diagram can be much more costly than all the pages of paper saved when packing symbols and diagrams tightly.

It is in SDL behaviour descriptions and in UML activity diagrams that presentation and layout have the most impact and the following aspects should be considered within a standard:

- the general flow of behaviour across a page;
- the spreading of diagrams over more than one page;
- the use of text extension symbols (in SDL);
- the alignment and orientation of symbols;
- the use of swimlanes (in UML).

### 7.1 The general flow of behaviour across a page

SDL and UML both allow the lines connecting symbols to flow in any direction across a page. As an example, the process shown in Figure 4 is legal SDL but is quite difficult to read.

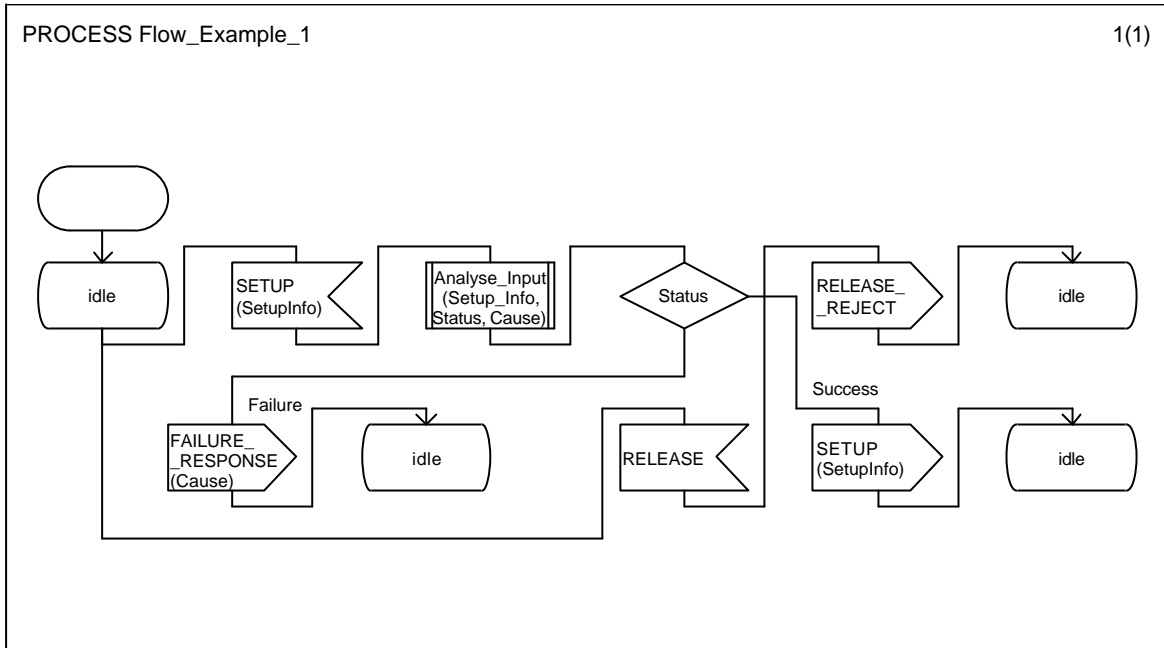


Figure 4: Example of poor layout of legal SDL

The readability of this process is greatly improved simply by laying it out in a "top-to-bottom" form, as in Figure 5.

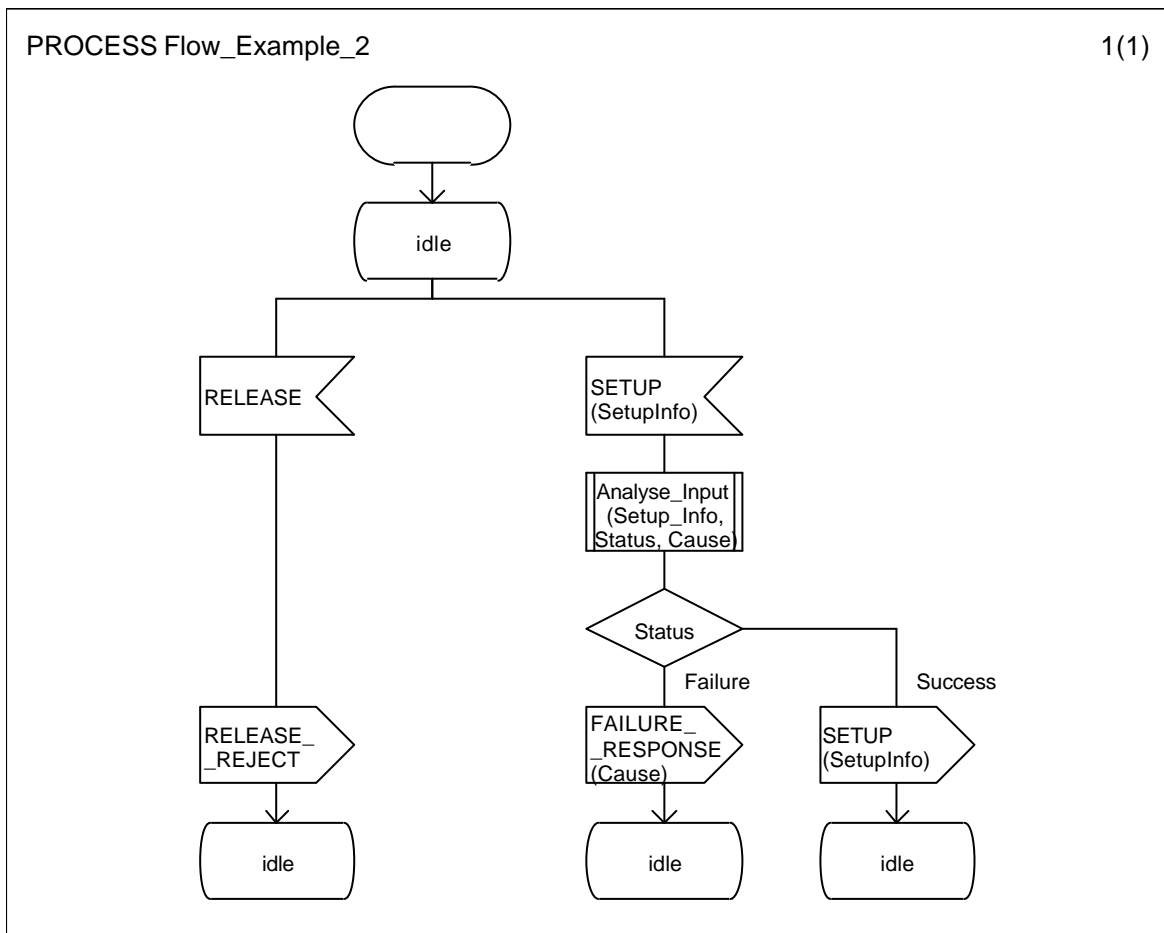


Figure 5: Example of improved layout

The orientation of SDL process symbols and, to a lesser extent, UML activity symbols is such that they naturally flow vertically and it is, thus, easier to read diagrams that follow this convention. Thus, <sup>(62)</sup>the general flow of SDL process

*diagrams and UML statechart and activity diagrams should be from the top of the page towards the bottom.* However, in some UML instances the flow may be better expressed using a left-to-right flow across the page.

Even in class diagrams and others where there is no "flow" expressed, readability can be improved if there is a general top-to-bottom layout on the page based on hierarchy or some other pertinent characteristic.

## 7.2 Diagrams covering more than one page

### 7.2.1 SDL behaviour diagrams

In most cases within standards it is not possible to constrain SDL process descriptions to one page. Only two options exist for breaking a diagram across a page boundary without affecting the readability. These are:

- using the NEXTSTATE symbol;
- using a connector symbol.

If it can be accommodated within the general structure of a description, <sup>(62)</sup>*the flow on a page of an SDL process should end in a NEXTSTATE symbol rather than a connector* as shown in Figure 6 and Figure 7. In general, this makes specifications easier to read. In addition, <sup>(57)</sup>*states that are entered from NEXTSTATE symbols on other pages should always be placed at the top of the page.*

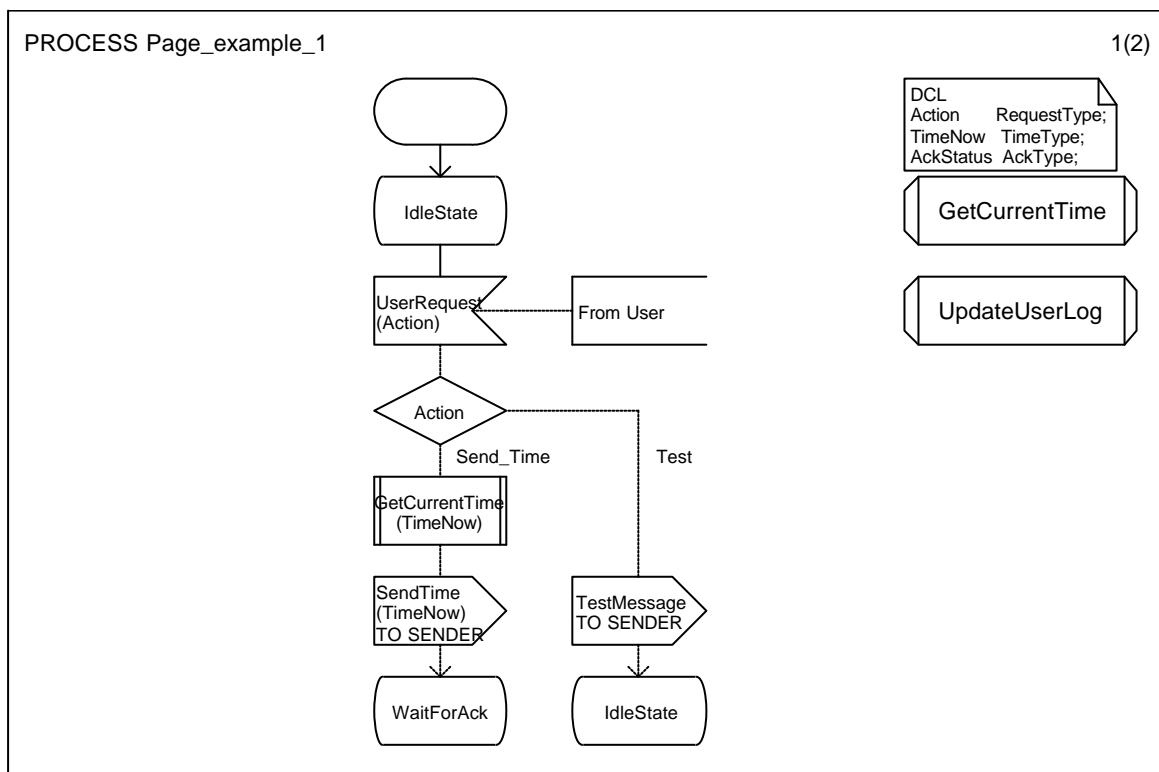
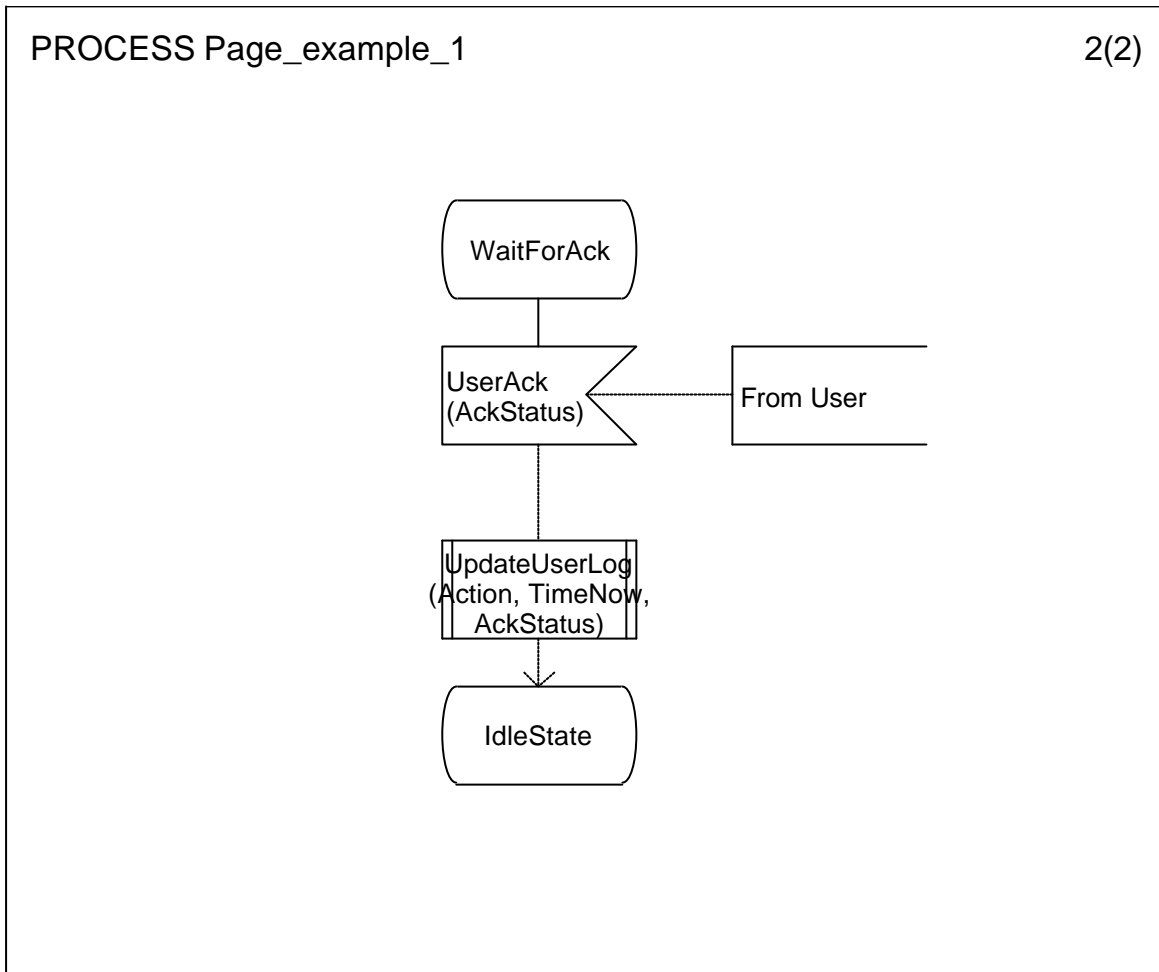


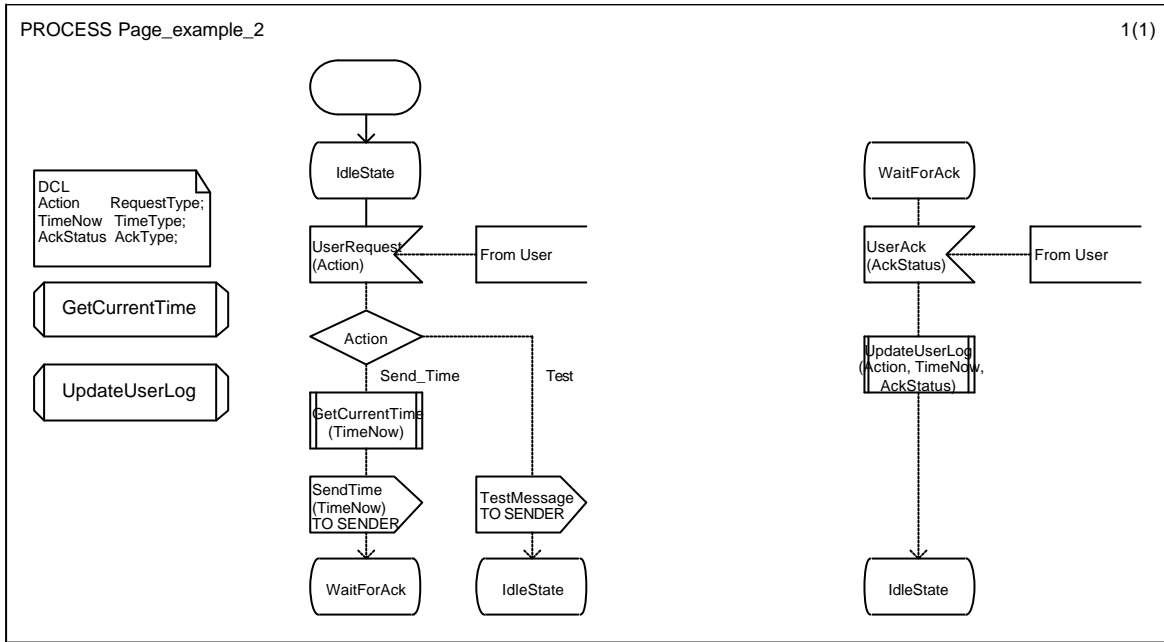
Figure 6: Paging using NEXTSTATE symbol (page 1)



**Figure 7: Paging using NEXTSTATE symbol (page 2)**

Although it would be possible to draw the example shown in Figure 6 and Figure 7 in a single thread with the "WaitForAck" state embedded part-way through, it is easier to locate individual states in a more complex specification if each thread is limited to a single transition (the processing between one state and the next one). <sup>(62)</sup>*Where transitions are short and simple they can be arranged side-by-side on a single page* as shown in Figure 8. However, <sup>(57)</sup>*when two or more transitions are shown on one page, there should be sufficient space between them to make their separation clear to the reader.*





**Figure 8: Transitions aligned on a single page**

When a single transition extends beyond the length of one page, a connector symbol can be used to provide a link to the next page. An example is shown in Figure 9 and Figure 10.

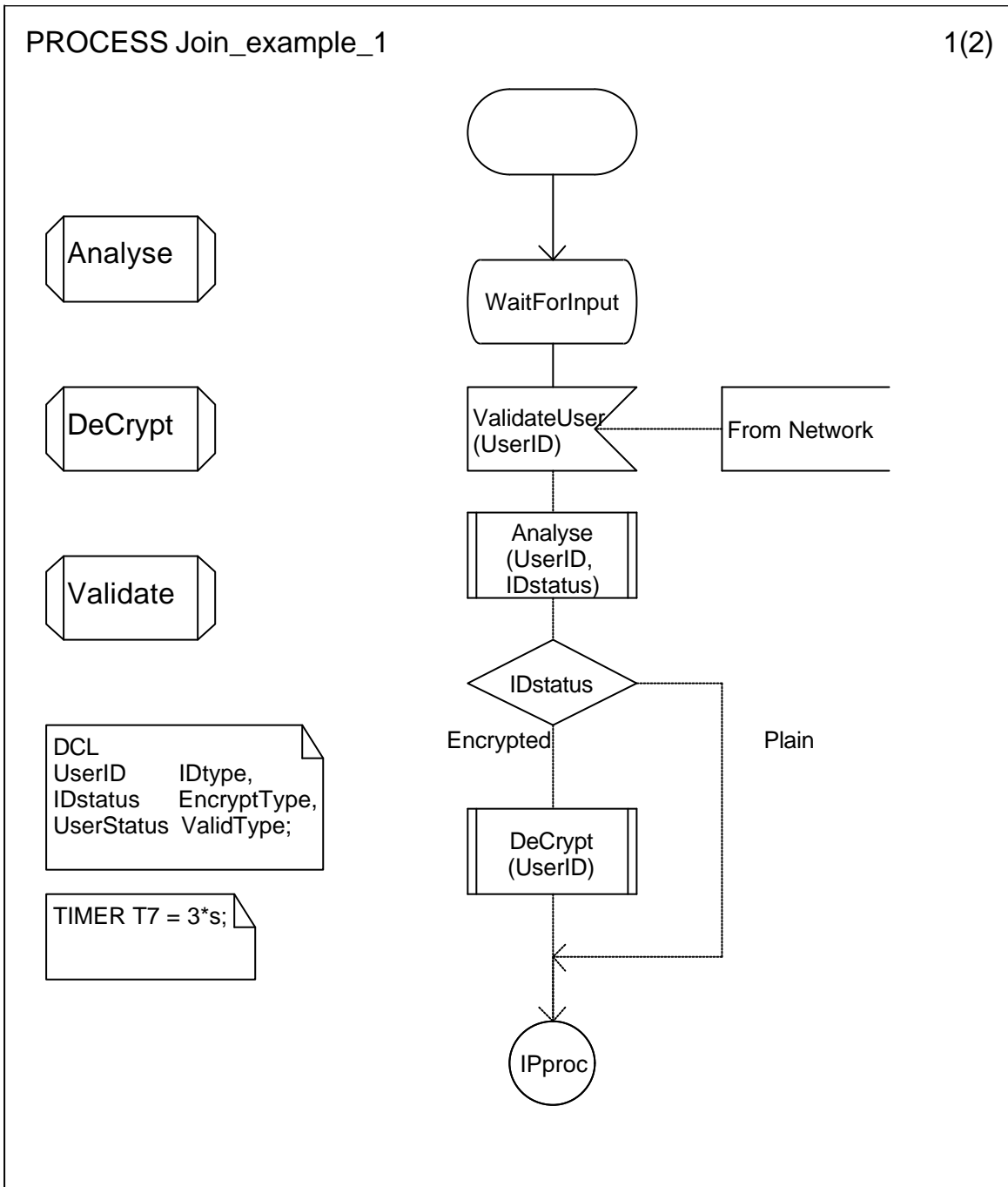
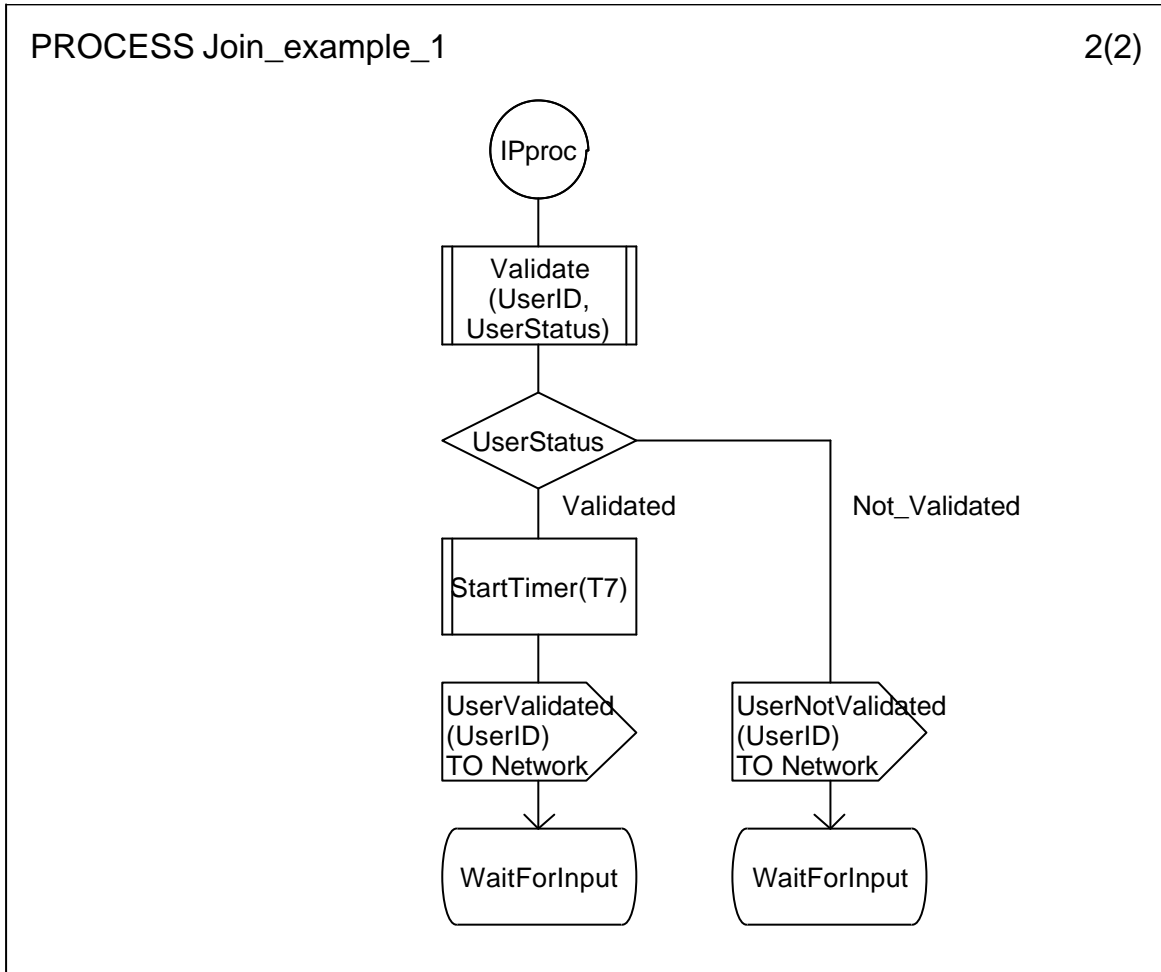
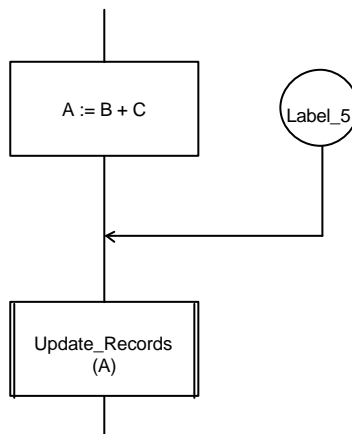


Figure 9: Paging using a connector symbol (page 1)



**Figure 10: Paging using a connector symbol (page 2)**

As can be seen in Figure 9 and Figure 10, the syntax of SDL allows a connector symbol to have a process flow line to it or from it but not both. Figure 11 shows how it is possible for a connector to be attached to a symbol anywhere on a page. These can be difficult to locate and so, to avoid confusion, <sup>(62)</sup>*connector symbols should generally only be used to provide a connection from the bottom of one page to the top of another*. However, long transitions can often be avoided by careful use of procedures (see subclause 9).



**Figure 11: Example of poor use of a connector symbol**

## 7.2.2 UML activity diagrams

UML does not support the concept of physical pages in its specifications but it may still be necessary to spread a distinct element of behaviour over more than one activity diagram. In this instance, there is only one mechanism that can be used for linking the diagrams and that is by using a state symbol. An activity diagram which terminates in a state other than the "End" state, will be assumed to continue at a subsequent instance of the same state in another activity diagram. In the example shown in Figure 12, the activity in the right-hand part of the diagram continues on from the "Connected" state on the left-hand side. Particularly in those cases where the specification of behaviour is distributed over many diagrams, *(62)activity diagrams or statechart diagrams should use text boxes indicate what functions are specified in other diagrams or in which diagram the behaviour continues.*

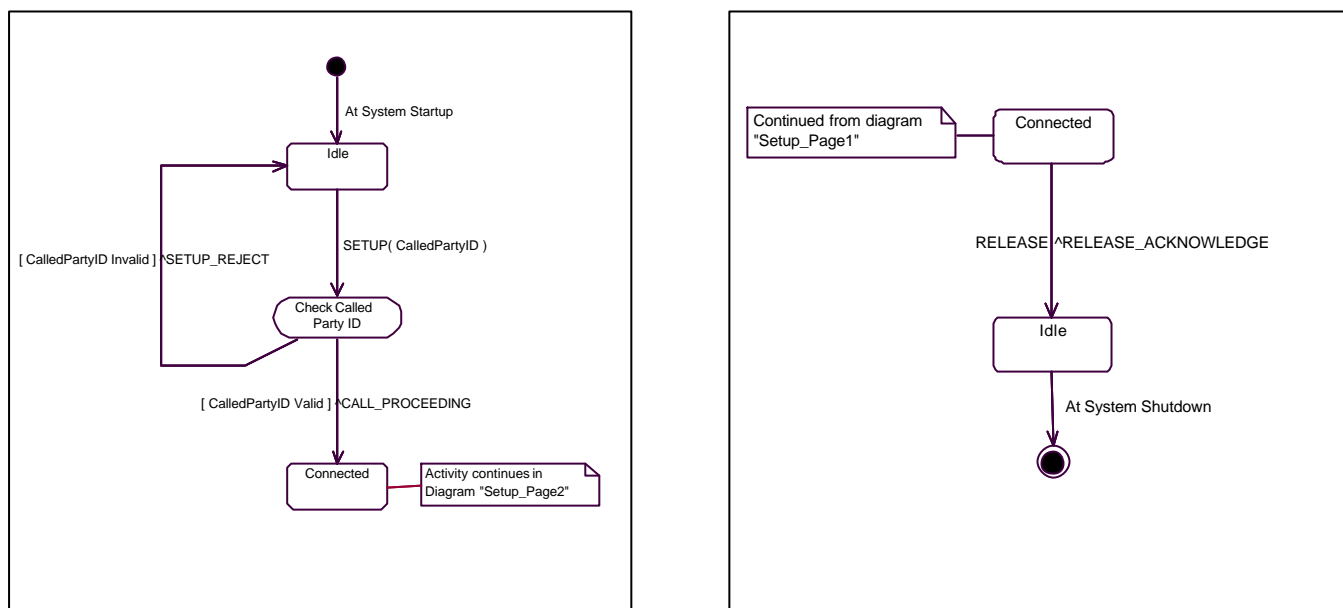


Figure 12: Example of UML activity diagram pages linked at a state

## 7.2.3 Symbols common to all pages

An SDL process description (which may exist in a system, block or process diagram) should not be considered to be simply a "flow-chart" specifying a sequence of actions and decisions to be taken by a particular entity. In order to be complete, a process description may include the following:

- a specification of formal parameters;
- variable, signal and data definitions;
- class and interface definitions;
- procedure references;
- class reference;
- the process graph, itself.

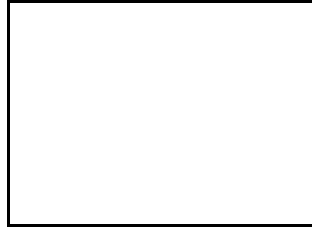
Symbols such as procedure references and text boxes containing DCL, TIMER and other declarative statements are valid for all pages of the process in which they appear. The language syntax allows them to be drawn on any page but, for easier reading, *(62)all reference symbols and text boxes containing common declarations should be collected together at a single point within the process chart.* For simple processes, and where space allows, these symbols can be placed together on the first page with the first transition, as can be seen in Figure 8 and Figure 9. In other cases, a separate page (or pages, if necessary) can be used to collect these symbols together.

To further improve the readability of the SDL, *(62)separate text box symbols should be used for each different type of declaration* (for example, variable declarations, timers, signal specifications, data type specifications and formal

parameters). It can also be useful to sub-divide these groupings into separate text boxes according to application-specific criteria (for example, grouping all of the BOOLEAN SYNONYM definitions together).

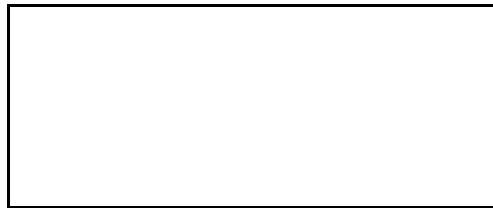
## 7.3 Text extension symbols

The SDL symbols are not always large enough to contain all of the text necessary to specify the task represented by the symbol and if the character size is set to a value that makes it readable, the text spills over into the area surrounding the symbol as can be seen in Figure 13.



**Figure 13: Text overflowing a symbol**

This can be difficult to read and, in the strict sense of the language, is syntactically incorrect. Therefore, <sup>(62)</sup>*when the text associated with a task symbol overflows its symbol boundaries, a text extension should be used to carry the additional information* as shown in Figure 14. The syntax of SDL specifies that the text in the extension symbol is added after the text in the task symbol. To avoid misinterpretation, care should be taken to ensure that the text extension symbol appears to the right of or below the task symbol unless all of the text is placed in the extension symbol. However, as a general rule the text extension symbol should not contain all of the text. For example, in the case of signals, the signal name should be placed inside the input or output symbol.



**Figure 14: Use of Text Extension symbol**

Even in cases where the text does not overflow the symbol, this is a useful presentation method which can be used to separate the signal name from the parameter list in inputs and outputs. For reasons of clarity, it is not advisable to split the parameter list between the primary symbol and the extension.

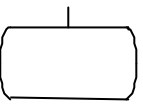
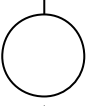
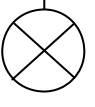
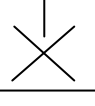
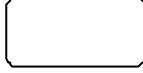

As an alternative to the use of a text extension symbol, SDL permits the re-sizing of both a task symbol and the text contained in it.

## 7.4 Alignment and orientation of symbols

### 7.4.1 Alignment

Neither SDL nor UML place any semantic significance on the placement and alignment of symbols but a process or activity page that is carefully arranged and not over filled with symbols and connecting lines will always be easier to read and interpret than one that is not.

There is no particular benefit to be gained by aligning symbols of a particular type except that <sup>(62)</sup>*symbols that terminate the processing on a particular page should be aligned horizontally* to make it easier for the reader to identify all of the points where processing ceases or continues on a new page or thread. These symbols include:

- SDL NEXTSTATE symbol 
- SDL Connector symbol 
- SDL RETURN symbol 
- SDL STOP symbol 
- UML STATE symbol 
- UML END STATE symbol 

In the example shown in Figure 15, the processing on the page can end in a number of different states. The alignment of all of the associated NEXTSTATE symbols at the bottom of the page makes it clear what all of these possibilities are.

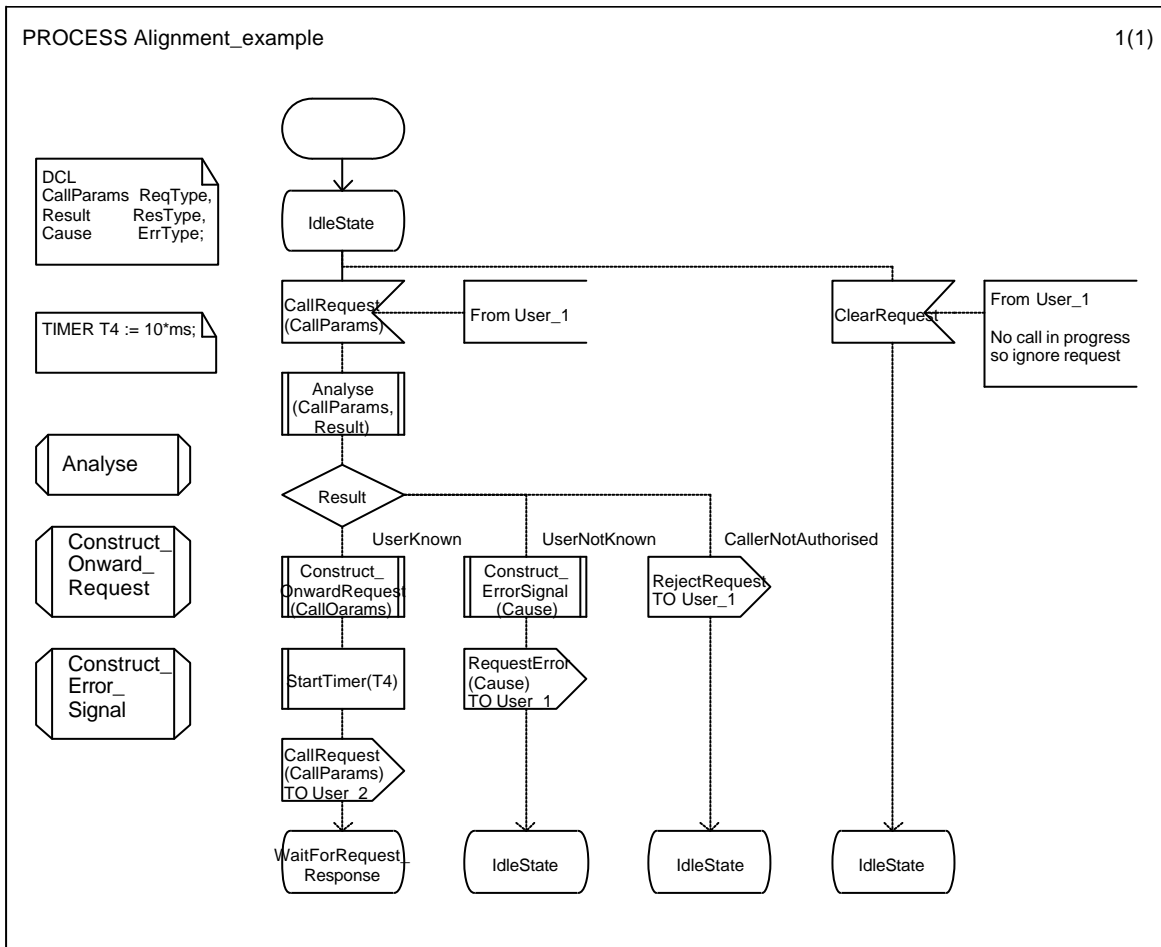


Figure 15: Example showing the alignment of NEXTSTATE symbols

## 7.4.2 Orientation

Most SDL symbols are symmetrical and, thus, cannot be shown in different orientations. INPUT and OUTPUT symbols are different in that they can be shown either right facing or left facing, thus



SDL accepts both orientations as correct but does not assign any specific meaning to either. However, <sup>(62)</sup>*in simple systems where each process communicates with only one or two other processes, the orientation of INPUT and OUTPUT symbols can be used to improve the readability of the SDL. However, to avoid possible specification errors and mis-interpretation, explicit methods of identifying the source and destination of signals should be used.* Symbol orientation should not be considered to be a substitute for the use of a "From" comment on an INPUT or the TO and VIA statements in an OUTPUT as described in subclause 11. <sup>(57)</sup>*If used, the significance of the orientation of SDL symbols should be clearly explained in the text introducing each process diagram.*

---

# 8 Structuring behaviour descriptions

---

## 9 Using procedures, operators methods and macros

---

## 10 Using decisions

---

## 11 System structure, communication and addressing

---

## 12 Specification and use of data

---

## 13 Using Message Sequence Charts (MSC)

### 13.1 Introduction

The Message Sequence Charts (MSC) language is defined in ITU-T Recommendation Z.120.

A basic MSC describes a scenario and consists of interacting instances. An instance is an object that has the properties of a certain entity. On an instance, the ordering of events is specified. Events can be message outputs, message inputs, local actions and timer events.

An HMSC (High-level Message Sequence Chart) is a roadmap of scenarios, where the details are hidden and described in basic MSCs or HMSCs that are Referenced in the HMSC.

## 13.2 Relationship between MSC and SDL

As far as possible, entities in MSC should correspond to SDL entities. Normally, it is only useful to specify a subset of a system's behaviour in MSC. It is also common not to reproduce the complete SDL architecture in MSC, but to represent only the important communicating parts with MSC instances.

## 13.3 Presentation and layout

There should be a reasonable amount of information in an MSC diagram, making the specification easy to comprehend but each diagram should be limited to the information that fits into one printed page. The structuring mechanisms in MSC can be used to avoid large diagrams <sup>(62)</sup> *If splitting a scenario into several distinct MSC diagrams is not feasible, vertical paging of diagrams might be used.* However, horizontal paging should be avoided.

If vertical paging is necessary, the instance heads and the MSC diagram name should be repeated on each page. The instance end symbols must only appear on the last page.

<sup>(62)</sup> *A clear spacing between symbols in an MSC diagram should be maintained both horizontally and vertically.* This makes it easier for each instance and message to be clearly distinguished from any others.

### 13.3.1 Annotations

There are four different annotations in MSC:

- note
  - appears between items of texts;
- comment symbol
  - can be attached to events or symbols;
- text symbol
  - may contain larger texts for documentary purposes;
- informal action
  - may be used to informally express internal behaviour of an instance (see also 13.10).

As in any formal language, <sup>(62)</sup> *Annotations help to improve the understanding of an MSC description and should be used freely.*

Another very useful practice is to annotate which scenarios (or parts of scenarios) that are normal from those that are exceptional.



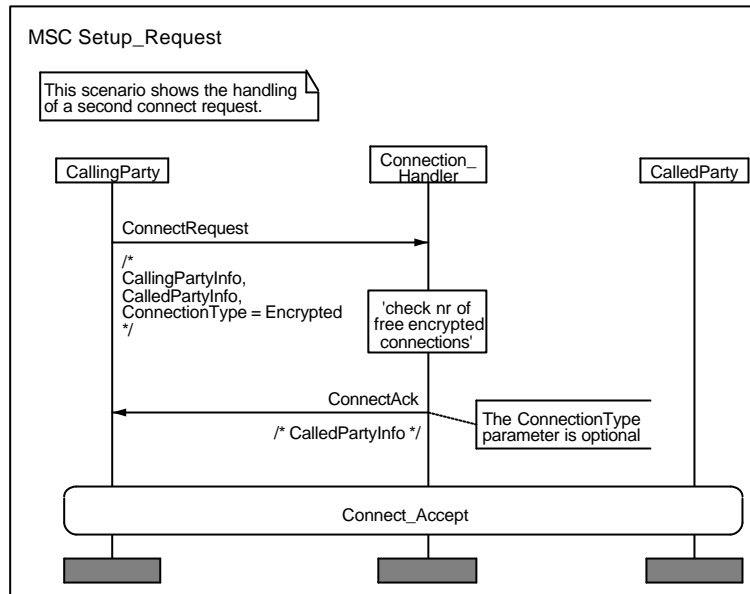


Figure 16: Annotations in MSC

## 13.4 Naming and scope

Most MSC names are globally visible within the set of MSC and HMSC diagrams defined by one MSC document specification. An instance kind name is visible outside of its MSC document. Gate names and MSC formal parameter names are visible in the scope of one MSC diagram.

As far as possible, <sup>(62)</sup>*names in an MSC should be the same as the names of corresponding entities in the SDL*. For example, an MSC message name should be the same as its corresponding SDL signal name, and an MSC instance should have the same kind name as the corresponding SDL process or block.

An entity may have the same name as another visible entity if the two entities are of different classes. A message may thus have the same name as a timer or an instance. <sup>(62)</sup>*Entity names should be unique within a specification.*

## 13.5 MSC document

An MSC document is a collection of MSCs and HMSCs (Figure 17), declaring used instances, messages, timers and MSC References. It is also the defining document for an instance kind. An MSC document might specify an inheritance relationship between two instances (instance kinds), allowing specialization of used scenarios (MSC References).

Since an MSC document is not needed unless instance decomposition, instance kind inheritance or the data concepts are used, it can often be avoided in order to reduce complexity of the specification.

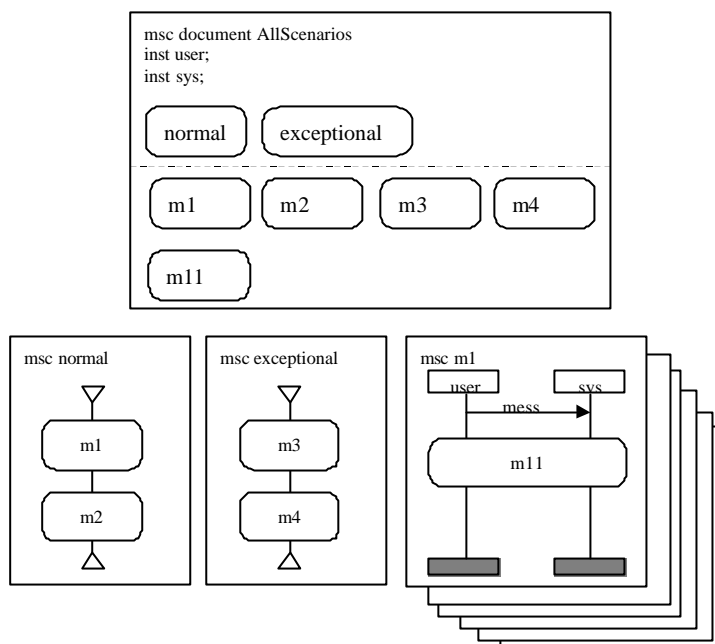


Figure 17: Collection of MSC diagrams

## 13.6 Structuring

There are two distinctive mechanisms for structuring MSC specifications. The first is related to the logical system architecture. The second is related to behaviour.

### 13.6.1 Architecture

#### 13.6.1.1 Instance

An instance is an object of an entity specifying behaviour by means of events that are ordered on the instance axis. More than one instance might be used to describe one entity. Every instance has a name associated with it and an optional kind name, e.g. process name, which indicates which entity the instance is describing. In relation to SDL, the kind name can be preceded by a kind denominator which may be one of the reserved words **system**, **block** or **process**. An instance without kind name will have its own name as an implicit kind name. <sup>(62)</sup>*If there is an associated SDL specification, each MSC instance should have a kind name and kind denominator corresponding to the name and entity kind of the equivalent entity in SDL.*

It is easy to add more and more instances to an MSC in an attempt to make it easier to understand. Unfortunately, this can have the opposite effect by adding complexity which can be an unnecessary distraction. So, <sup>(62)</sup>*the number of instances included in an MSC should be kept low to maintain a focus on the normative interface(s) and important entities in the logical or physical model.*

The instance name (together with the optional kind name) may be placed above or inside the instance head. For the sake of consistency, <sup>(62)</sup>*if the kind name is present in an MSC instance, the instance head symbol should contain the instance name with the kind name placed above the symbol*, as shown in Figure 18. Otherwise both names have to be separated by a colon symbol.

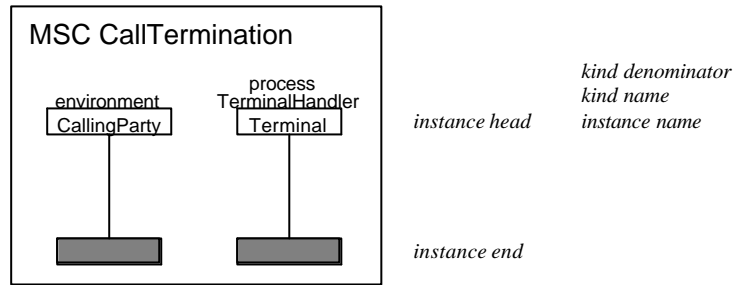


Figure 18: Placement of instance name and kind name

### 13.6.1.2 Instance decomposition

Behaviour described by several instances can be composed into one instance, hiding the intra-communication between the original instances. This means that the same part of a scenario is described in (at least) two diagrams, firstly on the higher level and secondly on a lower level, showing the internal behaviour of the decomposed instance. Furthermore, a decomposed instance needs a defining MSC document in which used instances, messages and MSC References are defined. <sup>(62)</sup>*Instance decomposition should be avoided in MSCs because of the complexity it might introduce.*

It is however good practice to represent a higher-level SDL entity with an instance, without describing the lower-level behaviour, if this abstraction improves the understanding of the overall behaviour.

### 13.6.1.3 Dynamic instances

Dynamic instances in MSC can be described by using the instance creation and instance stop concepts. Generally, standards describe a static view of the components avoiding the more complex dynamic identity relations and so <sup>(62)</sup>*dynamic instances should be avoided in MSCs.* Instance creation and instance stop should only be shown if they are a vital part of the specification.

Note the difference between the instance end and the instance stop. The instance end terminates the description of the behaviour of an instance within one MSC diagram, while the instance stop describes the termination of the entity that the instance represents.

### 13.6.1.4 Environment

In general, one MSC specifies the possible behaviour of only a part of a certain system. Everything else is referred to as "the environment" with which messages can be interchanged. The environment can be considered to be one or several instances that communicate with the instances in the MSC. Graphically the environment is represented by the diagram frame. Communication with the environment is provided by message arrows connected to the frame (see Figure 19).

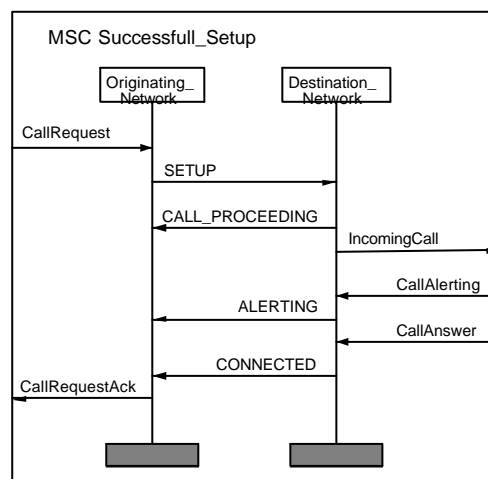


Figure 19: Messages being sent to and from the environment

There are situations when using the frame to represent the environment is counter-intuitive. In the example shown in Figure 19, a natural, but not justified interpretation would be that the message CallAlerting is sent in response to message IncomingCall. In fact, message CallAlerting might be sent before message IncomingCall, possibly from a different entity than the receiver of message IncomingCall.

As an alternative to the environment frame, specific instances may be used to describe the interaction of the system with the environment. When there is communication with more than one distinct environment entity, explicit instances for the environment enable the description of ordering, and allow a concrete behaviour description of external entities that interact with the system under consideration. *(62)Instances with instance kind name "environment" should be used to represent the environment in an MSC.*

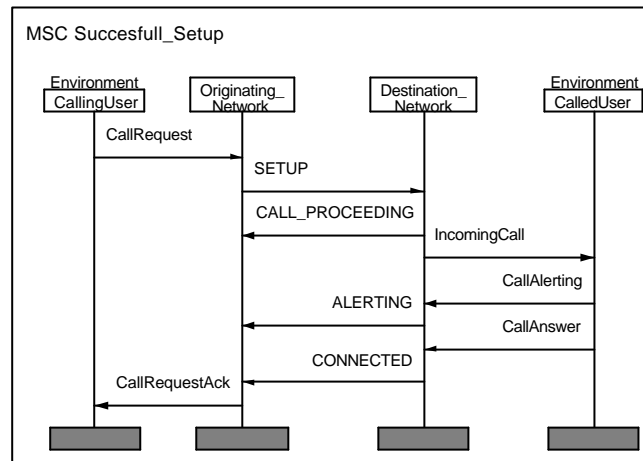


Figure 20: MSC with instances representing the environment

## 13.6.2 Behaviour

In MSC, there is a possibility to divide complex scenarios into smaller, named descriptions. There are several reasons to do this:

- making the specification easily readable and suitable for print-out;
- reuse of common behaviour parts, ensuring easier maintenance of the specification;
- hiding details while focussing on message exchange;
- keeping logically distinct parts separate.

This structuring of behaviour is realized by allowing expressions on MSC parts. The parts can be a group of events or an MSC Reference. In an expression, the following relationships between the parts might be expressed:

- sequence (seq);
- alternative (alt);
- optionality (opt);
- parallelism (par);
- Repetition (loop);
- Exception (exc).

These expressions might be used in three different ways or contexts:

- HMSC;
- MSC references in basic MSCs;
- In-line expressions in basic MSCs.

An MSC Reference is used to refer to other MSC or HMSC diagrams by means of their MSC name. MSC References may be used within basic MSCs or in HMSCs.

***HMSCs should be used to specify a high-level view of scenarios which are defined in other MSC or HMSC diagrams.***

Generally, unrestricted use of the expressions can cause an explosion of the number of scenarios, which may cause problems with validation.

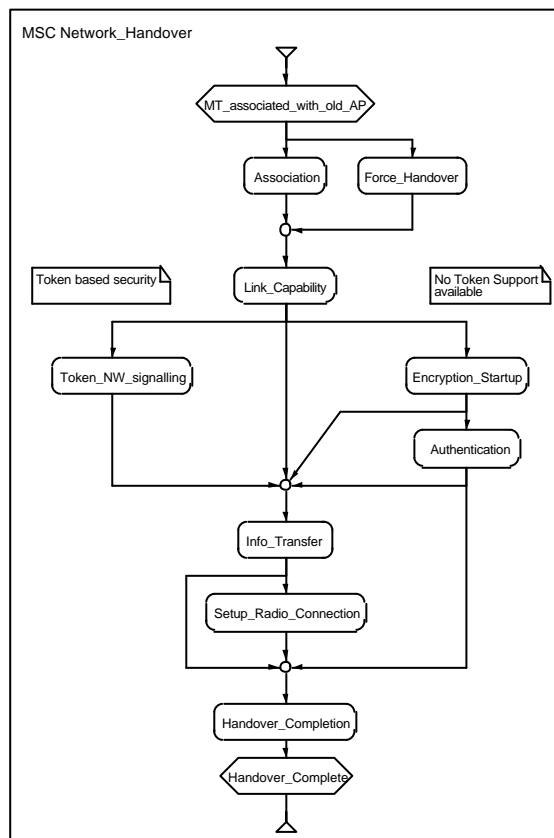
### 13.6.2.1 High-level MSC (HMSC)

The composition of a set of MSCs is specified by means of a High-level MSC (HMSC) which is a roadmap of the contained MSC References. HMSCs provide a graphical way of describing the combination of Message Sequence Charts, typically visualizing sequence, alternative and loop relationships.

Apart from MSC References, an HMSC can also contain conditions, start, stop and connection symbols.

*(62) Connections should always be used when HMSC flow lines join or merge to distinguish them from simple crossing lines.*

Unlike plain MSCs, instances and messages are not shown within HMSCs which focus only on composition aspects.

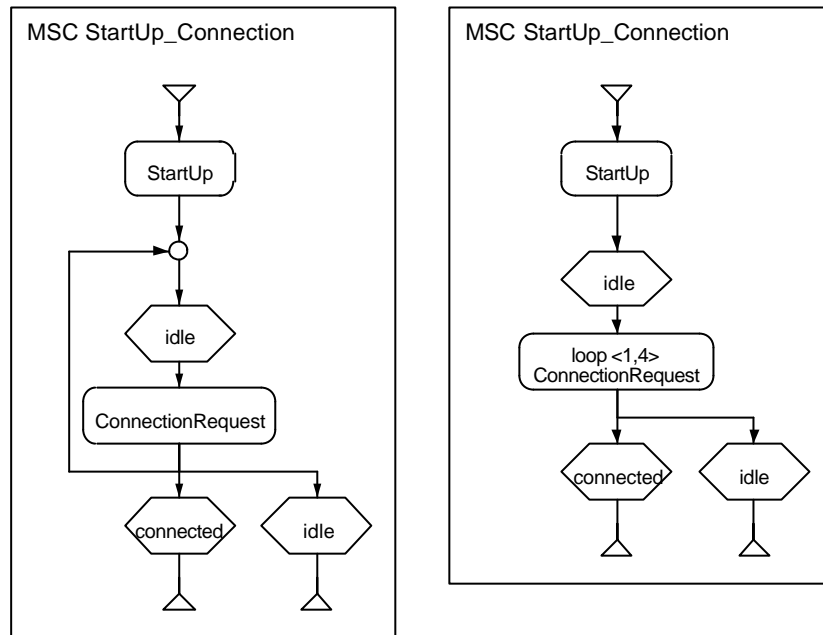


**Figure 21: Example of HMSC usage**

The annotations, "Token based security" and "No token support available" help to provide some helpful functional segregation within the HMSC shown in Figure 21. Such *(62) annotations should be used within HMSC to explain the purpose of different alternative branches.*

HMSCs are hierarchical in the sense that an MSC Reference may refer to an HMSC and, consequently support a top down design approach very well. In order to maintain sufficient transparency and manageability, <sup>(62)</sup>*References to other HMSCs should be used within HMSCs to ensure that a logical structuring of described behaviour is achieved.* This has the added advantage of keeping to a minimum the number of symbols in any one HMSC.

An MSC Reference may contain a textual operator expression instead of a single Reference name. The textual expression offers the same expressiveness as the graphical notation with the one exception that loop boundaries can be given in the textual form. MSC Reference expressions are useful for a compact representation, in particular of several alternatives, but makes the description less intuitive. To improve readability <sup>(62)</sup>*graphical HMSC expressions should be used in preference to textual Reference expressions.*



**Figure 22: HMSC with graphical relations between the references and corresponding HMSC with reference expression**

### 13.6.2.2 MSC reference in basic MSC

Behaviour parts can also be reused or abstracted in basic MSCs by using MSC References connected to the instances. In general, the number of MSC References should be kept low within a plain MSC in order to focus on the message interchange.

HMSC References may be included in basic MSCs but referring to "overview" charts from detailed sequence specifications can be confusing. Therefore, <sup>(62)</sup>*Plain MSCs should not include HMSC References.*

MSC References in basic MSCs should be used as a structuring means and for the reuse of scenarios. Figure 23 shows an example of MSC References used in the specification of a test purpose preamble and postamble. As such, the MSC Reference plays a similar role to that of a procedure in SDL. If the same scenario appears in several MSCs of an MSC document, it should be specified as an MSC of its own.

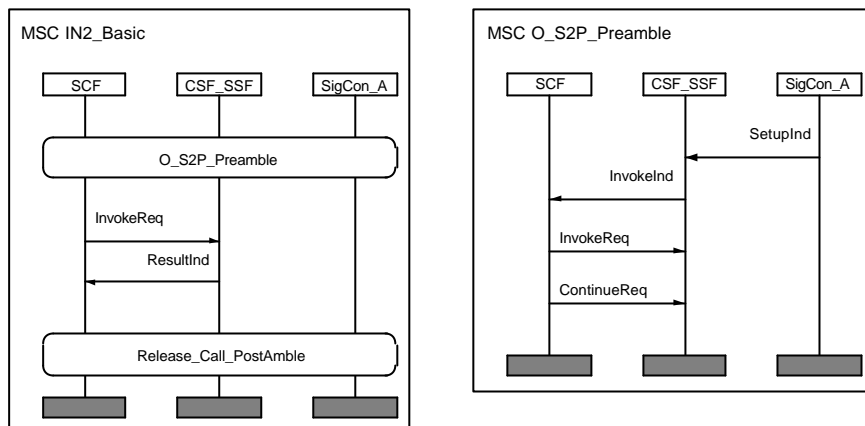


Figure 23: MSC references in basic MSC

### 13.6.2.3 Inline expression

An inline expressions can be looked upon as an expanded form of an MSC Reference expression used in a basic MSC context. They are ideally suited to the compact description of several small variants. Typically they cover only a small section of the complete MSC which means that the inline expression should contain only a few events.

Inline expressions are used to define concisely 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 can use the following operators on events:

- sequence;
- alternative;
- optional;
- loop;
- parallel;
- exception.

Inline expressions can be nested. Inline expressions give the benefit of conciseness at the expense of making a specification more complex and, thus, more difficult to read. *(62)The use of multiple inline expressions in a single MSC diagram should be limited to avoid an unnecessary explosion in the number of implicit scenarios.*

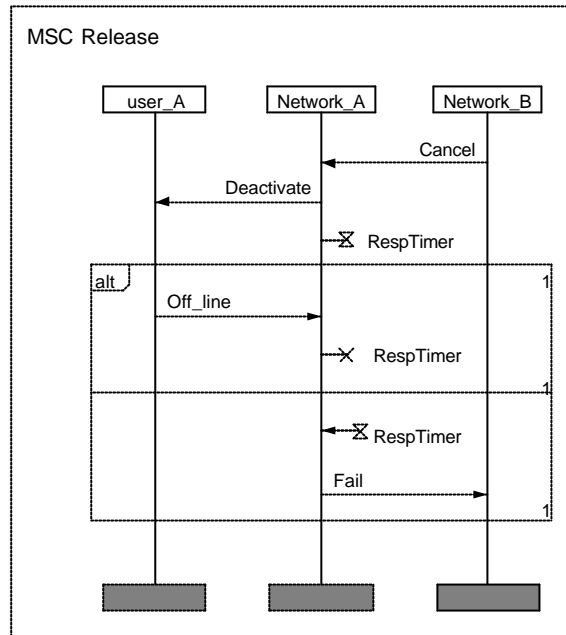


Figure 24: Usage of inline expression

In certain situations, inline expressions are the only descriptive way to illustrate a scenario. For example, after setting a timer an alternative can be used to describe both the normal course of action and the exceptional behaviour resulting from a timeout. A scenario with two or more alternative courses of action might either be described in an HMSC, where the alternative is described by different MSC References on alternative paths, or in a basic MSC, where the alternative is described by alternative inline expressions. <sup>(62)</sup>*HMSCs should be used to highlight significant alternative or optional behaviour paths but; if the differences are only minor, these could be described within an MSC using inline expressions.*

## 13.7 Data

When data (type information or values) can enhance the understanding of an MSC, this may be indicated informally by notes, comments or informal actions. To formally express data in MSC can lead to an unnecessarily complex specification that can be difficult to understand and maintain.

*Data types and expressions introduce unnecessary complexity to a specification and should be avoided.*

## 13.8 Message

An MSC message describes two asynchronous events: a sending event that is performed by the sending instance and a receive event that is handled by the receiving instance. The receive event is optional (see 13.8.1).

Messages may cross instances that are placed between the sender and receiver. By rearranging the order of the instances, instance crossing messages can be minimized. <sup>(62)</sup>*The crossing of MSC instances by messages should be minimised by placing frequently communicating instances close to each other wherever possible.* However, the natural and logical ordering of entities should be considered to be more important than strict adherence to this guideline.

A message arrow may be drawn either horizontally or with a downward slope. Both forms are equivalent but the downward slope is sometimes used informally to indicate the passage of time. Since this is prone to misinterpretation, <sup>(62)</sup>*delay or the passage of time should be described by the time concepts in MSC* (see 13.13).

Messages with downward slopes can also be used to describe the overtaking of messages. However, <sup>(62)</sup>*the unnecessary crossing of messages should be avoided since it obscures the meaning of an MSC.*

<sup>(62)</sup>*Message overtaking should be avoided, except in MSCs explicitly describing the behaviour when overtaking takes place.*



In general, two or more events may not be attached to the same point or at the same level on an instance axis. There is one exception to this rule. An incoming event and an outgoing event may be attached to the same point or at the same height. This is interpreted as if the incoming event is drawn above the outgoing event.

Although both representations are equivalent, within a standard, <sup>(62)</sup>*an MSC should show an outgoing event below the incoming event that preceded it* as this presentation gives a clearer description of the ordering relationships.

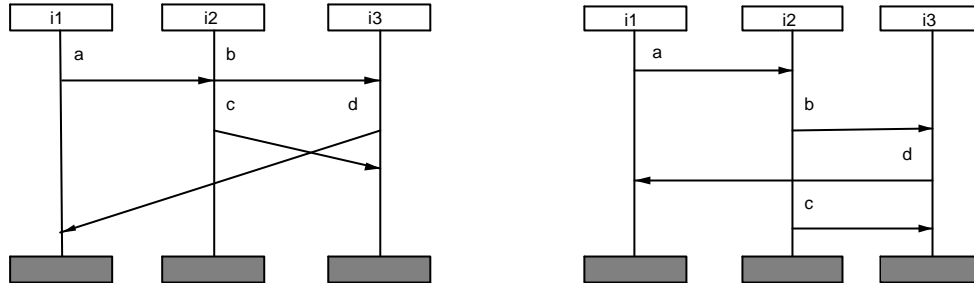


Figure 25: Clear ordering of events

Note The two examples are semantically equivalent, but the layout of the second example makes the scenario easier to comprehend.

MSC does not require the parameters of the message to be described. However, providing an informal type name in a message is often useful when creating an SDL specification with an MSC model as input. In some cases, it might also be of value to indicate that a parameter has a certain value if this improves the understanding of the scenario. In protocol standards it is not unusual for a message to have an extensive parameter list defined and the inclusion of such lists with all messages can make an MSC very difficult to read.

The description of MSC message parameters may differ from SDL signal parameters regarding the level of detail. In a single scenario, it is common to highlight only the interesting aspects of the message parameters i.e.; the part that affects the further behaviour of the scenario. This abstraction is a very useful mechanism that ensures that the scenarios are not too detailed and complex.

MSC message parameters have a formal meaning in that they illustrate how values are transmitted together with the message. According to the language definition, these values must conform to the corresponding parameter data type in the message declaration. However, in the interests of clarity, <sup>(62)</sup>*only those parameters that are absolutely necessary for the understanding of the message sequence should be included with an MSC message*. In order to be able to do this while still complying with the MSC syntax, <sup>(57)</sup>*if incomplete message parameter information is to be shown in an MSC, this should be given in a note, following the message name* as shown in Figure 26.

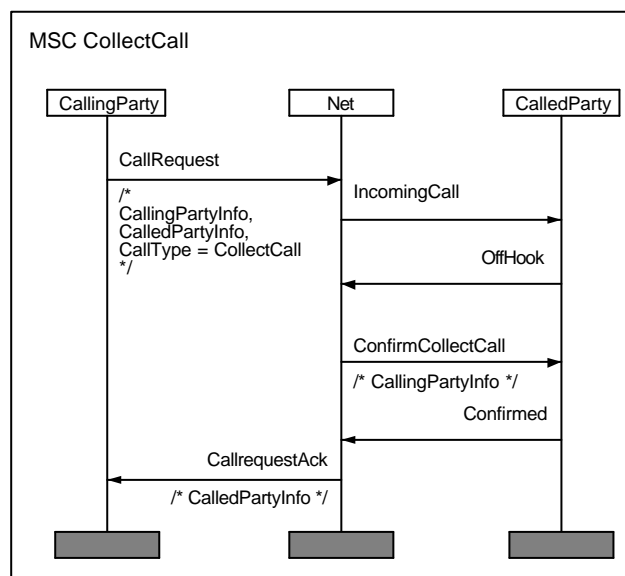


Figure 26: Indication of message parameter information

## 13.8.1 Incomplete messages

Besides the specification of successful transmission of messages, incomplete messages can be described in MSC. An incomplete message communication is represented by a lost message symbol or a found message symbol. A lost message is a message output for which the message input is unknown. A found message is a message input for which the message output is unknown. Lost messages may be used to describe the reaction of a system in error cases such as in case of an unreliable transmitter (see Figure 27).

*(62) Lost and found message should normally not be used in MSCs* because they 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.

A situation where a lost message may be used is in a scenario that describes how re-sending of lost messages is handled. Found messages may be used when a message can be sent by several possible instances, and the sending identity is not relevant to the scenario.

In SDL, unsuccessful signal transmission can only be described in an indirect manner.

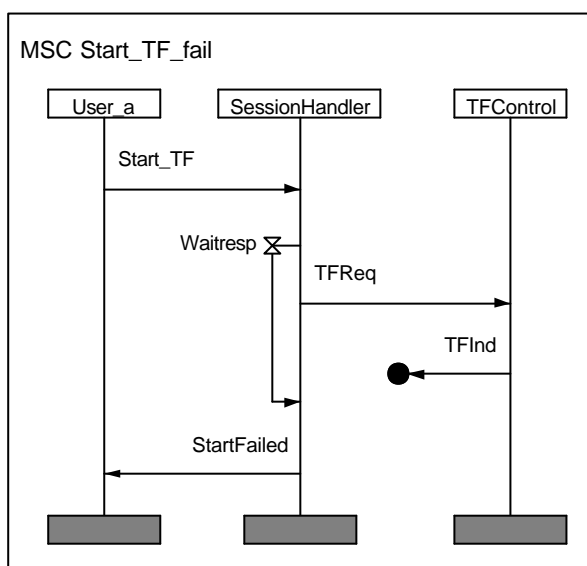


Figure 27: Lost and found message

## 13.9 Condition

MSC conditions can be used in two different ways:

- as setting conditions;
- as guarding conditions.

Setting conditions define the actual system state of the instance(s) that share the condition. Guarding conditions are used to restrict the possible ways in which an MSC can continue.

Local setting conditions can be used to indicate system states corresponding to states in SDL. The number of used local conditions should be minimized in order to not obscure the primary function described by the MSC. Local guarding conditions may contain a boolean expression where variables are allowed. To make the description easy to understand, *(62) logical names should be used in MSC guarding conditions instead of variable expressions.*

Conditions have no further meaning. They are not events and a global condition does not imply synchronization between the shared instances.

Global conditions are attached to all instances contained in an MSC and denote global system states. For standardization, one important use of global conditions is as connection points between different MSCs within a set of MSCs. Conditions are used in HMSCs to indicate global system states or guards and impose restrictions on the MSCs

that are referenced in the HMSC. Conditions also give extra context information for the basic MSC and makes the MSC specification model easier to maintain. An example of an MSC with a global initial condition (guard) and a global final condition is shown in Figure 28.

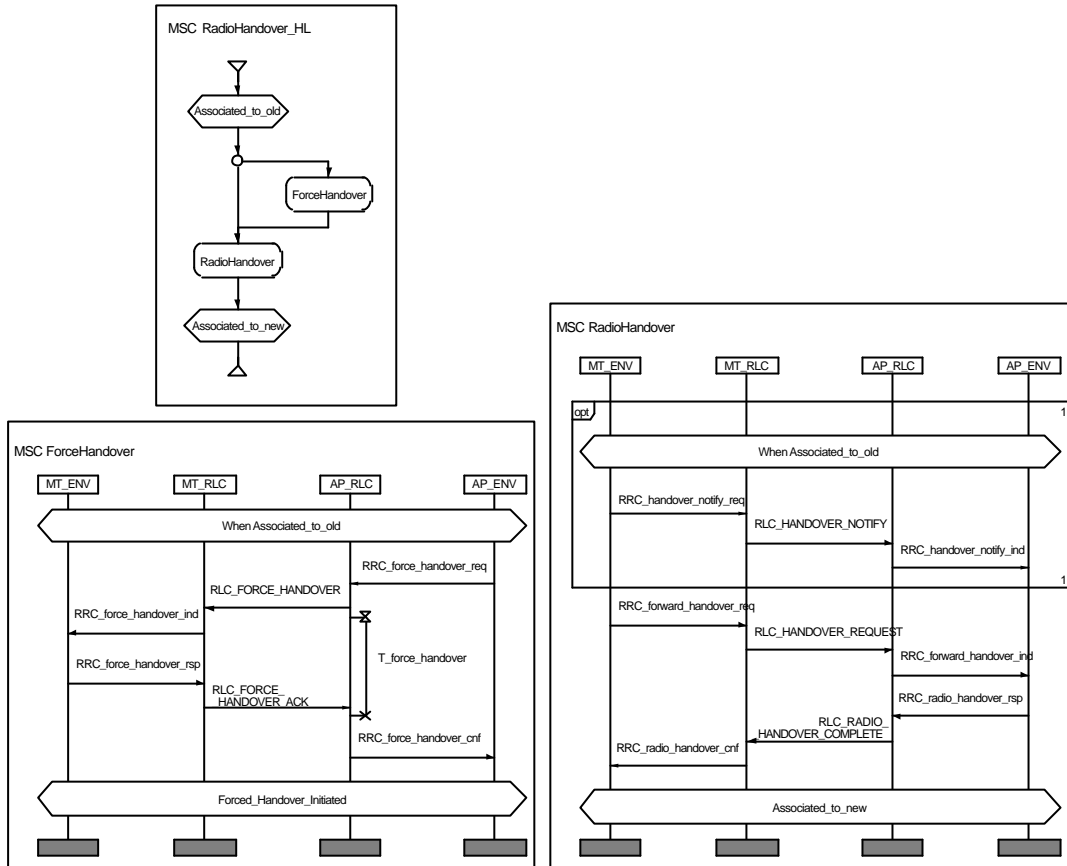


Figure 28: Global conditions used to restrict composition of MSCs

## 13.10 Action

In some situations, it can be useful to indicate informally the action that is performed after a message is received (see Figure 16). This is possible by using an informal action. <sup>62</sup>*Use of the MSC action symbol should be limited to the informal expression of a specific aspect of behaviour which helps to clarify the surrounding message sequence and to data assignments.*

## 13.11 Timer

Timers may be used informally to indicate delays or time constraints on event sequences. Since there is an explicit notation in MSC for time constraints and measurements (see 13.13), this should be used instead of timers as the notion of a timer entity may be too precise for most standard specifications.

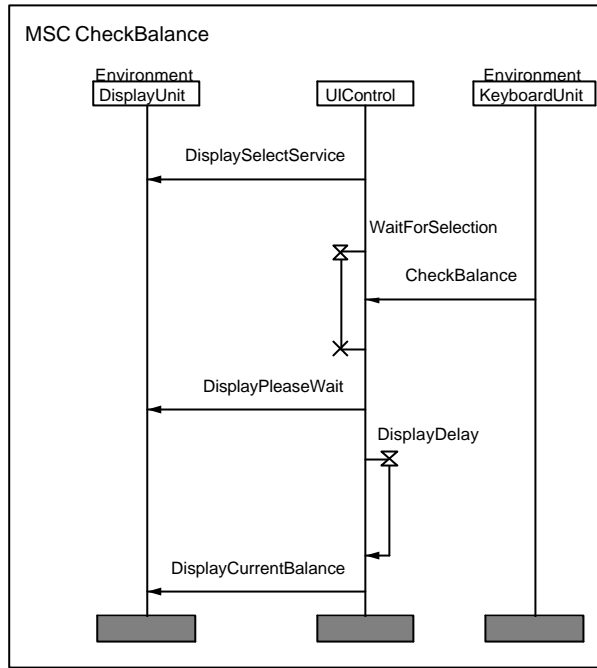


Figure 29: Timer usage

Note The WaitForSelection timer is used to restrict the waiting time for a response signal. The DisplayDelay timer is used as a delay in the execution.

In certain situations when using separated timer symbols, it is necessary to add an extra timer identifier in order to have an unambiguous scenario.

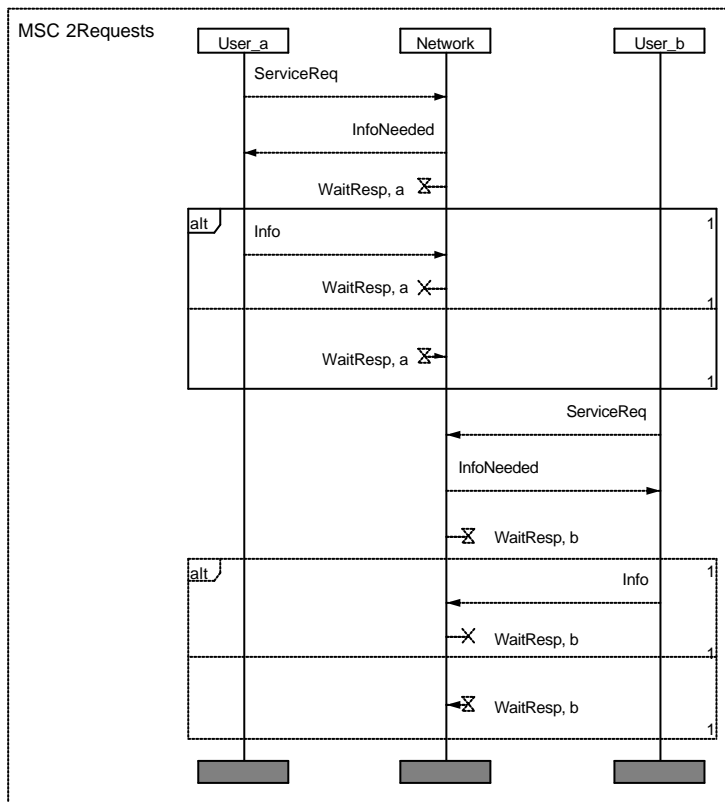


Figure 30: Separated timer symbols and timer identifier

## 13.12 Control Flow

In specifying distributed systems, all communication is normally described by asynchronous messages. It is however often the case that communication is by signal pairs, a call message and a corresponding reply message, together making a synchronous communication.

A logically connected signal pair might be high-lighted in an MSC specification by using the special symbols for reply, method and suspend.

The control flow concepts are:

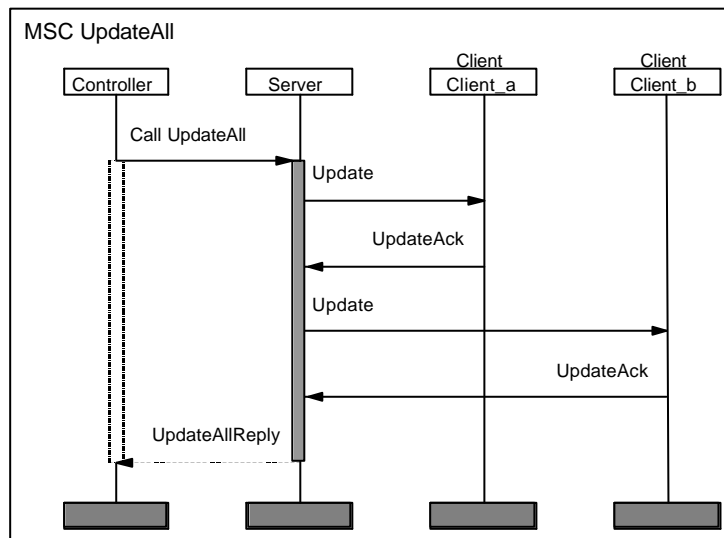
- method call;
- reply symbol;
- method symbol;
- suspend symbol.

A method call is represented by a message symbol with the CALL keyword before the message name. For a method call, there must always be a corresponding reply, and vice versa.

The method symbol is used to indicate that an instance is active. The suspension symbol is used to indicate that an instance is suspended, typically waiting for the reply of a blocking method call. The normal instance axis means that the instance is inactive, waiting for an activating event or a task to perform.

A method call followed by a suspension region is a synchronous method call.

If MSC Instances are used to represent entities that are not independent (asynchronously parallel), then the method and suspend symbols can be used to indicate how each active object gets the flow of control from the CPU.



**Figure 31: Specification of synchronous communication utilizing the suspend symbol and the method symbol.**

## 13.13 Time

The time concepts can be used for:

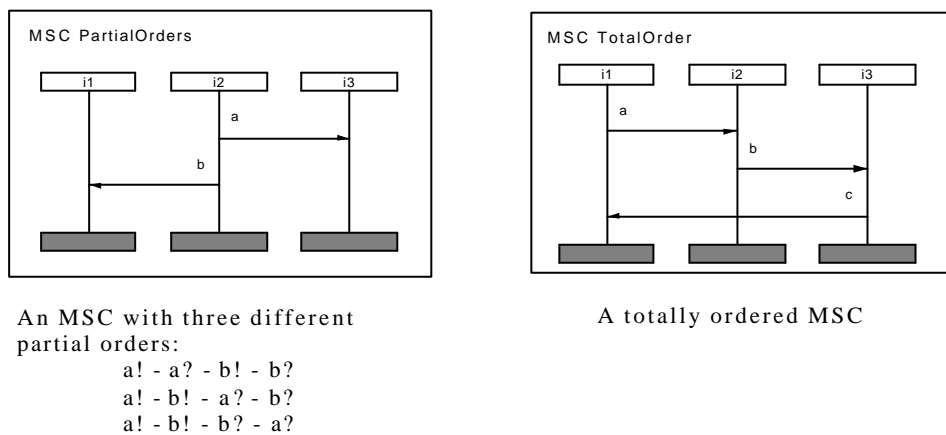
- Time measurements
- Timing constraint on events

Time constraints are useful for stating time requirements without adding behaviour to the model (compare with the use of timers). Using the time concepts assumes that a data type for handling time expressions is available.

**Figure 32: Time constraints between events**

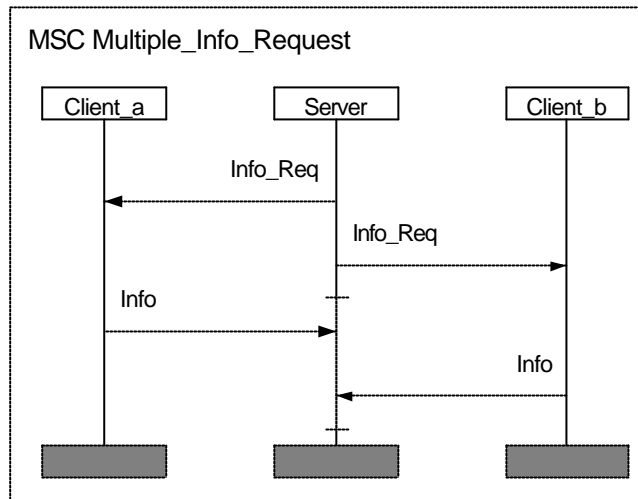
### 13.14 General ordering and coregion

Although an instance describes a total order of its events, an MSC normally describes only a set of partial event orders. This is because instances are independent, since each MSC instance is asynchronously parallel. Synchronization between instances is normally achieved by message passing.



**Figure 33: Event orders of MSCs**

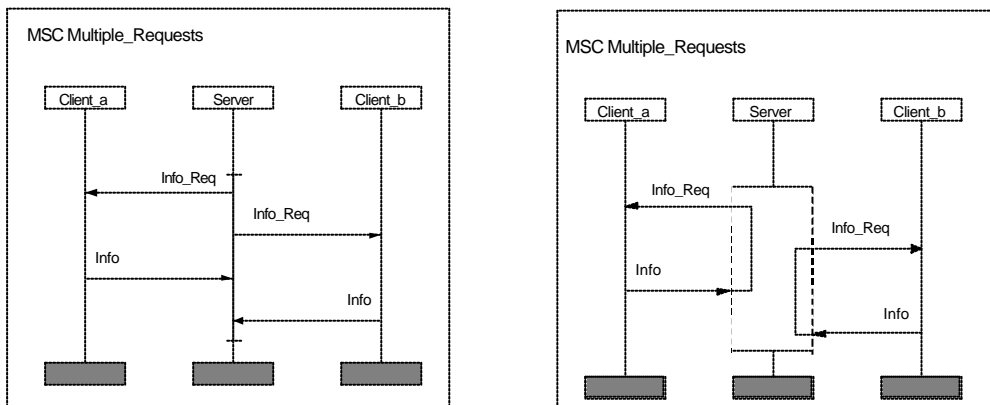
Coregions are useful for describing situations where two or several events might happen in an arbitrary order on one instance. They are also commonly used on decomposed instances to relax the total ordering imposed to the contained instances by the decomposed instance. However, large coregions, covering many events might be very hard to interpret. Thus, *the number of events shown in an MSC coregion should be limited.*



**Figure 34: Use of coregion**

Note The two Info messages can be received in any order by the Server instance. However, the first Info message must arrive after the second Info\_Req is sent and the two Info\_Req messages must be sent in a specific order.

General ordering can be used within a coregion to specify partial orders in an otherwise completely unordered region. However, <sup>(62)</sup>*an inline alternative expression should be used in an MSC instead of general ordering within a coregion.*



**Figure 35: General ordering within a coregion reduces the number of orders**

Note In the first MSC, the order restrictions that existed in the example in Figure 34 are released. On the other hand, there are a number of unwanted event orders in this MSC.

In the second MSC, each request and response message pair is now ordered, but the two events related to the communication with Client\_a is unordered with respect to the events related to the Client\_b communication.

---

## Annex A (informative): Reserved words

---

### A.1 SDL

#### A.1.1 Keywords

The following words are keywords in SDL and cannot be used as names.

NOTE: The list of keywords shows only the lower-case presentation. The upper-case equivalent of each is also an SDL reserved word.

abstract	active	adding	aggregation
alternative	and	any	as
association	atleast	block	break
call	channel	choice	comment
composition	connect	connection	constants
continue	create	dcl	decision
default	else	endalternative	endblock
endchannel	endconnection	enddecision	endexceptionhandler
endinterface	endmacro	endmethod	endobject
endoperator	endpackage	endprocedure	endprocess
endselect	endstate	endsubstructure	endsyntype
endsystem	endtype	endvalue	env
exception	exceptionhandler	export	exported
external	fi	finalized	for
from	gate	handle	if
import	in	inherits	input
interface	join	literals	macro
macrodefinition	macroid	method	methods
mod	nameclass	nextstate	nodelay
none	not	now	object
offspring	onexception	operator	operators
optional	or	ordered	out
output	package	parent	priority
private	procedure	protected	process
provided	public	raise	redefined
referenced	rem	remote	reset
return	save	select	self
sender	set	signal	signallist
signalset	size	spelling	start
state	stop	struct	substructure
synonym	syntype	system	task
then	this	timer	to
try	type	use	value
via	virtual	with	xor



## A.1.2 Predefined words

The following words are defined in ITU-T Recommendation Z.100 [4] in the SDL package "Predefined" and should not be redefined or used for any other purposes:

ACK	Array	Bag	BEL
bit	Bit	bitstring	Bitstring
Boolean	BS	CAN	Character
Charstring	chr	CR	DC1
DC2	DC3	DC4	del
DEL	DivisionByZero	DLE	Duration
EM	empty	Empty	emptystring
ENQ	EOT	ESC	ETB
ETX	extract	false	FF
first	fix	float	HT
incl	Integer	InvalidIndex	InvalidReference
IS1	IS2	IS3	IS4
last	length	LF	make
mkstring	modify	NAK	Natural
NoMatchingAnswer	NUL	num	Octet
octetstring	Octetstring	OutOfRange	power
Powerset	Predefined	Real	remove
SI	SO	SOH	String
STX	SUB	substring	SYN
take	Time	true	UndefinedField
UndefinedVariable	Vector	VT	

---

## A.2 MSC

The following words are keywords in MSC and cannot be used as names.

action	after	all	alt
as	before	begin	block
by	call	comment	concurrent
condition	connect	create	data
decomposed	def	empty	end
endafter	endbefore	endconcurrent	endexpr
endinstance	endmethod	endmsc	endsuspension
env	equalpar	escape	exc
expr	external	finalized	found
from	gate	in	inf
inherits	inline	inst	instance
int_boundary	label	language	loop
lost	method	msc	mscdocument
msg	nestable	nonnestable	offset
opt	order	otherwise	out
par	parenthesis	process	receive
redefined	reference	related	replyin
replyout	seq	service	shared
startafter	startbefore	starttimer	stop
stoptimer	suspension	system	text
time	timeout	timer	to
undef	using	utilities	variables
via	virtual	when	wildcards

## A.3 ASN.1

The following words are keywords in ASN.1 and cannot be used as names.

ABSENT	ABSTRACT-SYNTAX	ALL	APPLICATION
AUTOMATIC	BEGIN	BIT	BMPString
BOOLEAN	BY	CHARACTER	CHOICE
CLASS	COMPONENT	COMPONENTS	CONSTRAINED
DEFAULT	DEFINITIONS	EMBEDDED	END
ENUMERATED	EXCEPT	EXPLICIT	EXPORTS
EXTENSIBILITY	EXTERNAL	FALSE	FROM
GeneralizedTime	GeneralString	GraphicString	IA5String
IDENTIFIER	IMPLICIT	IMPLIED	IMPORTS
INCLUDES	INSTANCE	INTEGER	INTERSECTION
ISO646String	MAX	MIN	MINUS-INFINITY
NULL	NumericString	OBJECT	ObjectDescriptor
OCTET	OF	OPTIONAL	PDV
PLUS-INFINITY	PRESENT	PrintableString	PRIVATE
REAL	SEQUENCE	SET	SIZE
STRING	SYNTAX	T61String	TAGS
TeletexString	TRUE	TYPE-IDENTIFIER	UNION
UNIQUE	UNIVERSAL	UniversalString	UTCTime
UTF8String	VideotexString	VisibleString	WITH

## A.4 UML

The following words are keywords in UML and cannot be used as names.

«access»	association	«association»	«become»
«call»	complete	«copy»	«create»
«derive»	derived	«destroy»	destroyed
«document»	documentation	«executable»	«facade»
«file»	«framework»	«friend»	Generalization
global	«global»	«implementation»	«implementationClass»
implicit	«import»	incomplete	«instantiate»
«invariant»	«library»	local	«local»
«metaclass»	«metamodel»	new	overlapping
parameter	«parameter»	persistence	persistent
«postcondition»	«powertype»	«precondition»	«process»
«realize»	«refine»	«requirement»	«responsibility»
self	«self»	semantics	«send»
«signalflow»	«stub»	«systemModel»	«table»
«thread»	«topLevel»	«trace»	transient
«type»	«utility»	xor	

## Annex B (informative): Summary of guidelines

Table B.1 provides a summary of the guidelines for the use of SDL for descriptive purposes. This summary should be read in conjunction with the main body of text in the present document.

**Table B.1: Summary of guidelines**

Identifier	Guideline
<b>NAMING CONVENTIONS</b>	
1	A naming convention that can be applied consistently to each notation used should be chosen
2	The general use of names which differ only in character case to distinguish between entities should be avoided.
3	Care should be taken to ensure the consistent use of character case within names throughout an ASN.1, SDL, MSC or UML specification
4	Names of less than 6 characters may be too cryptic and names of more than 30 characters may be too difficult to read and assimilate.
5	The reserved words of all notations used within a standard should be avoided as defined names in each of the individual parts
6	Readability is improved if the same convention for separating words within names is used throughout a specification
7	In most cases an underscore character between each word removes any possibility of misinterpretation and this is the approach that is recommended
8	The use of a single name for multiple purposes should be avoided wherever possible
9	The addition of project-specific prefixes or suffixes can make meaningful names appear cryptic and should be used with great care
10	By giving blocks, processes and MSC instances names that represent the overall role that they play within the system, it is possible to distinguish process names from procedure names. If carefully chosen, they can help to link the SDL and MSC back to the corresponding subclauses in the text description
11	The name chosen for an SDL operation should indicate the specific action taken by the operation
12	If possible, it is advisable to leave at least one significant word in the name unabbreviated as this can help to provide the context for interpreting the remaining abbreviations
13	The name chosen for an interface or signal list should indicate the general function of the grouped signals
14	Where all signals between one block or process and another can be logically grouped together, signal list names can be chosen to indicate the origin and the destination of the associated signals
15	A state name should clearly and concisely reflect the status of the process while in that state
16	If it is important to number states then this should be done in conjunction with meaningful names
17	The name chosen for a variable should indicate in general terms what it should be used for
18	Names used to identify constants can be more specific by indicating the actual value assigned to the constant
19	The names of SDL data types should be capitalized while the names of literals and synonyms should begin with a lower-case character
<b>PRESENTATION AND LAYOUT OF DIAGRAMS</b>	
20	The general flow of SDL process diagrams and UML statechart and activity diagrams should be from the top of the page towards the bottom
21	The flow on a page of an SDL process should end in a NEXTSTATE symbol rather than a connector
22	States that are entered from NEXTSTATE symbols on other pages should always be placed at the top of the page.
23	Where transitions are short and simple they can be arranged side-by-side on a single page
24	When two or more transitions are shown on one page, there should be sufficient space between them to make their separation clear to the reader
25	Connector symbols should generally only be used to provide a connection from the bottom of one page to the top of another
26	Activity diagrams or statechart diagrams should use text boxes indicate what functions are specified in other diagrams or in which diagram the behaviour continues
27	All reference symbols and text boxes containing common declarations should be collected together at a single point within the process chart.
28	Separate text box symbols should be used for each different type of declaration
29	When the text associated with a task symbol overflows its symbol boundaries, a text extension should be used to carry the additional information
30	Symbols that terminate the processing on a particular page should be aligned horizontally
31	In simple systems where each process communicates with only one or two other processes, the orientation of INPUT and OUTPUT symbols can be used to improve the readability of the SDL. However, to avoid possible specification errors and mis-interpretation, explicit methods of identifying the source and destination of signals should be used

Identifier	Guideline
32	If used, the significance of the orientation of SDL symbols should be clearly explained in the text introducing each process diagram
<b>STRUCTURING BEHAVIOUR DESCRIPTIONS</b>	
<b>USING PROCEDURES, OPERATORS METHODS AND MACROS</b>	
<b>USING DECISIONS</b>	
<b>SYSTEM STRUCTURE, COMMUNICATION AND ADDRESSING</b>	
<b>SPECIFICATION AND USE OF DATA</b>	
<b>USING MESSAGE SEQUENCE CHARTS (MSC)</b>	
33	If splitting a scenario into several distinct MSC diagrams is not feasible, vertical paging of diagrams might be used.
34	A clear spacing between symbols in an MSC diagram should be maintained both horizontally and vertically
35	Annotations help to improve the understanding of an MSC description and should be used freely.
36	Names in an MSC should be the same as the names of corresponding entities in the SDL
37	Entity names should be unique within a specification.
38	If there is an associated SDL specification, each MSC instance should have a kind name and kind denominator corresponding to the name and entity kind of the equivalent entity in SDL
39	The number of instances included in an MSC should be kept low to maintain a focus on the normative interface(s) and important entities in the logical or physical model
40	If the kind name is present in an MSC instance, the instance head symbol should contain the instance name with the kind name placed above the symbol
41	Instance decomposition should be avoided in MSCs because of the complexity it might introduce
42	Dynamic instances should be avoided in MSCs.
43	Instances with instance kind name "environment" should be used to represent the environment in an MSC.
44	Connections should always be used when HMSC flow lines join or merge to distinguish them from simple crossing lines.
45	Annotations should be used within HMSC to explain the purpose of different alternative branches
46	References to other HMSCs should be used within HMSCs to ensure that a logical structuring of described behaviour is achieved.
47	Graphical HMSC expressions should be used in preference to textual Reference expressions
48	Plain MSCs should not include HMSC References
49	The use of multiple inline expressions in a single MSC diagram should be limited to avoid an unnecessary explosion in the number of implicit scenarios
50	HMSCs should be used to highlight significant alternative or optional behaviour paths but; if the differences are only minor, these could be described within an MSC using inline expressions
51	The crossing of MSC instances by messages should be minimised by placing frequently communicating instances close to each other wherever possible
52	Delay or the passage of time should be described by the time concepts in MSC
53	The unnecessary crossing of messages should be avoided since it obscures the meaning of an MSC
54	Message overtaking should be avoided, except in MSCs explicitly describing the behaviour when overtaking takes place
55	An MSC should show an outgoing event below the incoming event that preceded it
56	Only those parameters that are absolutely necessary for the understanding of the message sequence should be included with an MSC message
57	If incomplete message parameter information is to be shown in an MSC, this should be given in a note, following the message name
58	Lost and found message should normally not be used in MSCs
59	Logical names should be used in MSC guarding conditions instead of variable expressions
60	Use of the MSC action symbol should be limited to the informal expression of a specific aspect of behaviour which helps to clarify the surrounding message sequence and to data assignments.
61	The number of events shown in an MSC coregion should be limited.
62	An inline alternative expression should be used in an MSC instead of general ordering within a coregion

---

## History

<b>Document history</b>		
V1.1.1	June 2001	1 <sup>st</sup> draft, Scope & TOC
V1.1.2	August 2001	Addition of Naming chapter

---