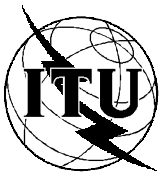


INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Z.120

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(03/93)

CRITERIA FOR THE USE AND APPLICABILITY OF FORMAL DESCRIPTION TECHNIQUES

MESSAGE SEQUENCE CHART (MSC)

ITU-T Recommendation Z.120

(Previously “CCITT Recommendation”)

FOREWORD

The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the International Telecommunication Union. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, established the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

ITU-T Recommendation Z.120 was prepared by the ITU-T Study Group X (1988-1993) and was approved by the WTSC (Helsinki, March 1-12, 1993).

NOTES

1 As a consequence of a reform process within the International Telecommunication Union (ITU), the CCITT ceased to exist as of 28 February 1993. In its place, the ITU Telecommunication Standardization Sector (ITU-T) was created as of 1 March 1993. Similarly, in this reform process, the CCIR and the IFRB have been replaced by the Radiocommunication Sector.

In order not to delay publication of this Recommendation, no change has been made in the text to references containing the acronyms "CCITT, CCIR or IFRB" or their associated entities such as Plenary Assembly, Secretariat, etc. Future editions of this Recommendation will contain the proper terminology related to the new ITU structure.

2 In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

© ITU 1994

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the ITU.

CONTENTS

			Page
1	Introduction to MSC		1
2	General rules		2
	2.1	Lexical rules	2
	2.2	Visibility and Naming Rules	3
	2.3	Comment	3
	2.4	Text symbol	4
	2.5	Drawing Rules	5
	2.6	Paging of MSCs	5
3	Message Sequence Chart document		5
4	Basic MSC		6
	4.1	Message Sequence Chart	6
	4.2	Instance	9
	4.3	Message	11
	4.4	Condition	13
	4.5	Timer	15
	4.6	Action	17
	4.7	Process creation	17
	4.8	Process stop	18
5	Structural concepts		19
	5.1	Coregion	19
	5.2	Sub-Message Sequence Chart	20
6	Message Sequence Chart examples		22
	6.1	Standard message flow diagram	22
	6.2	Message overtaking	23
	6.3	MSC basic concepts	24
	6.4	MSC with time supervision	25
	6.5	MSC-composition / MSC-decomposition	26
	6.6	Local conditions	28
	6.7	Shared condition and messages with parameters	29
	6.8	Creating and terminating processes	29
	6.9	Coregion	30
	6.10	Sub-Message Sequence Chart	31
Annex A – Index			32

SUMMARY

Scope/objective

The purpose of recommending Message Sequence Chart (MSC) is to provide a trace language for the specification and description of the communication behaviour of system components and their environment by means of message interchange. Since in MSCs the communication behaviour is presented in a very intuitive and transparent manner, particularly in the graphical representation, the MSC-language is easy to learn, use and interpret. In connection with other languages it can be used to support methodologies for system specification, design, simulation, testing, and documentation.

Coverage

This Recommendation presents a syntax definition for Message Sequence Charts in abstract, textual, and graphical representation. An informal semantics description is provided.

Application

The main area of application for MSC is an overview specification of the communication behaviour for real time systems, in particular telecommunication switching systems. By means of MSCs selected system traces, primarily "standard" cases may be specified. Non-standard cases covering exceptional behaviour may be built on them. Thereby MSCs may be used for requirement specification, interface specification, simulation and validation, test case specification and documentation of real time systems. MSC may be employed in connection with other specification languages, in particular SDL. In this context, MSCs also provide a basis for the design of SDL-systems.

Status/stability

The status of basic MSCs is fairly stable, including the constructs for instance, instance creation and termination, message exchange, action, timer handling and condition. Further developments and extensions are expected mainly for structural concepts.

Associated work

Recommendation Q. 65: Stage 2 of the method for the characterization of services supported by an ISDN.

MESSAGE SEQUENCE CHART (MSC)

(Helsinki, 1993)

1 Introduction to MSC

A Message Sequence Chart (MSC) shows sequences of messages interchanged between system components and their environment. In SDL, system components are modelled by service, process and block constructs. MSCs have been used for a long time by CCITT Study Groups in their recommendations and within industry, according to different conventions and under various names such as Signal Sequence Chart, Information Flow Diagram, Message Flow and Arrow Diagram.

The reason to standardize MSCs is to make it possible to provide tool support for them, to exchange MSCs between different tools, to ease the mapping to and from SDL specifications and to harmonize the use within CCITT.

One part of the standardization work is to provide a clear definition of the meaning of an MSC. This is done in this Recommendation by means of relating MSCs to SDL specifications, as follows: An MSC describes one or more traces of an SDL system specification.

According to the above definition, an MSC can be derived from an existing SDL system specification, and may then be used e.g. to record the result of an animation. However, an MSC is generally produced before the SDL system specification, and then may serve as

a) an overview of a service as offered by several entities;

b) a statement for requirements for SDL specifications;

c) a basis for elaboration of SDL specifications;

d) a basis for system simulation and validation;

e) a basis for selection and specification of test cases;

f) a semi-formal specification of communication;

g) an interface specification.

Since an MSC usually only covers a partial behaviour, the selection of partial behaviours is a crucial task. The candidates for MSCs are primarily the “standard” cases. Further cases are generally built on them and cover exceptional behaviours, e.g. caused by errors of various kinds.

In the following a syntax for Message Sequence Charts is presented in abstract, textual, and graphical representation. A corresponding informal (verbal) semantics description is provided.

This Recommendation is structured in the following manner: In clause 2, general rules concerning syntax, drawing and paging are outlined. In clause 3, a syntax definition for the Message Sequence Chart document which is a collection of Message Sequence Charts is provided. Clause 4 contains the syntax definition for Message Sequence Charts and the syntax rules for the basic constituents, i.e. instance, message, condition, timer, action, process creation and termination. In clause 5, higher level concepts concerning structuring and modularisation are introduced. These concepts support a top down specification and permit a refinement of individual instances by means of generalized time ordering (see 5.1) and sub-Message Sequence Chart (see 5.2). In clause 6, examples are provided for all MSC-constructs. Appendix I contains cross references for the <keyword>s and non-terminals.

2 General rules

Only rules specific for Message Sequence Charts are listed. The remaining rules are identical to those of Recommendation Z.100.

2.1 Lexical rules

Contrary to Z.100 <text> does not contain the semicolon, since <text> is used within the definition of action, and the semicolon is employed as a terminator for action. The semicolon is contained in the definition of <end> (see 2.3).

<lexical unit> ::=
<word>
| <character string>
| <special>
| <composite special>
| <note>
| <keyword>
| <semicolon>

<keyword> ::=
action
| **all**
| **block**
| **comment**
| **concurrent**
| **condition**
| **create**
| **decomposed**
| **endconcurrent**
| **endinstance**
| **endmsc**
| **endmscdocument**
| **endsubmsc**
| **endtext**
| **env**
| **from**
| **inst**
| **instance**
| **msc**
| **mscdocument**
| **in**
| **out**
| **process**
| **referenced**
| **related to**
| **reset**
| **service**
| **set**
| **shared**
| **stop**
| **submsc**
| **system**
| **text**
| **timeout**
| **to**

```
<text> ::=
                        { <alphanumeric>
                    |   <other character>
                    |   <special>
                    |   <full stop>
                    |   <underline>
                    |   <space>
                    |   <apostrophe> }*
<special> ::=
                        +
                    |   -
                    |   %
                    |   !
                    |   /
                    |   >
                    |   *
                    |   (
                    |   )
                    |   "
                    |   ,
                    |   =
                    |   :
<semicolon> ::=
                        ;
<note> ::=
                    /* <text> */
```

2.2 Visibility and naming rules

Entities are identified and referred to by means of associated names. Entities are grouped into entity classes to allow flexible naming rules. The following entity classes exist:

a) MSC document;

b) MSC;

c) sub-MSC;

d) instance;

e) condition;

f) timer;

g) message.

An entity that contains other entities according to the syntax rules forms a scope unit. The following scope units exist:

a) MSC document;

b) MSC;

c) sub-MSC;

d) instance.

No two entities within a scope unit and belonging to the same entity class can have the same name. Different occurrences of a condition name, timer name and message name within a scope unit denote the same entity. The name of an entity is visible within the enclosing scope unit, but not outside. Only visible names can be used when referencing entities.

2.3 Comment

A comment is a notation to represent comments associated with symbols or text.

In the *Concrete textual grammar* two forms of comments are used. The first form is the <note>.

The concrete syntax of the second form is:

<end> ::=

[<comment>] <semicolon>

<comment> ::=

comment <character string>

In the *Concrete graphical grammar* the following syntax is used:

<comment area> ::=

<comment symbol> ***contains*** <text>

is connected to <dashed association symbol>

<comment symbol> ::=

T1007630-93/d01

<dashed association symbol> ::=

T1007630-93/d02

One end of the <dashed association symbol> must be connected to the middle of the vertical segment of the <comment symbol>.

A <comment symbol> can be connected to any graphical symbol by means of a <dashed association symbol>. The <comment symbol> is considered as a closed symbol by completing (in imagination) the rectangle to enclose the text. It contains comment text related to the graphical symbol.

2.4 Text symbol

<text symbol> may be used in any <msc diagram> and <submsc diagram> for the purpose of general (global) comments (see 4.1 and 5.2). Text may be placed inside of the text symbol in form of <note>:

<text area> ::=

<text symbol> ***contains*** <note>

<text symbol> ::=

T1160120-94/d43

In the textual representation the following syntax is used:

<text definition> ::=

text <note> **endtext** <end>

The <text definition> is contained in the definition of <msc body> (see 4.1).

2.5 Drawing rules

The size of the graphical symbols can be chosen by the user. Symbol boundaries must not overlap or cross. An exception to this rule applies

a) for the crossing of message symbol with message symbol, timeout symbol, reset symbol, create symbol, instance axis symbol and dashed association symbol;

b) for the crossing of timeout symbol and reset symbol with message symbol, timeout symbol, reset symbol, create symbol and dashed association symbol;

c) for the crossing of create symbol with message symbol, timeout symbol, reset symbol, instance axis symbol and dashed association symbol;

d) for the crossing of condition symbol with instance axis symbol;

e) for the overlap of action symbol with instance axis symbol in column form.

There are two forms of the instance axis symbol and the coregion symbol: the single line form and the column form. It is not allowed to mix both forms within one instance.

If a shared condition (see 4.4) crosses an instance which is not involved in this condition the instance axis is drawn through.

In case where the instance axis symbol has the column form, the vertical boundaries of the action symbol have to coincide with the column lines.

Message lines may be horizontal or with downward slope (with respect to the direction of the arrow) and may be bended.

If there is a message arrow head and a message origin on the same point of the instance axis, then it is interpreted as if the message origin is drawn below the message arrow head.

2.6 Paging of MSCs

MSCs can be partitioned vertically over several pages. The horizontal partitioning may be handled by means of sub-MSC (see 5.2).

When an MSC is partitioned into several pages, then the <msc heading> is repeated on each page, but the instance end symbols may only appear on one page (on the "last" page for the instance in question). For each instance the <instance head area> must appear on the first page where the instance in question starts and must be repeated in dashed form on each of the following pages where it is continued. If messages or timers are continued from one page to the next page, the message or timer name has to appear on both pages.

Page numbering may be included on the pages of an MSC in order to indicate the correct sequence of pages. If page numbering is omitted, then the correct sequence of pages must be indicated by means of global conditions that act as connectors (see also 4.4).

3 Message Sequence Chart document

The Message Sequence Chart document header contains the document name and optionally, following the keyword **related to,** the identifier (pathname) of the SDL-document to which the MSCs refer. No special graphical grammar is introduced since it is assumed that the same document contains Message Sequence Charts in both graphical and textual representation.

Abstract grammar

MSC-document :: *MSC-document-name*
[Sdl-reference]
*[Message-sequence-chart-***set** *]*
*[Submsc-***set** *]*

MSC-document-name = *Name*

Sdl-reference :: *Sdl-document-identifier*

Sdl-document-identifier = *Identifier*

Identifier :: [*Qualifier*]
Name

Qualifier :: *Path-item+*

Path-item = *System-qualifier* |
Block-qualifier |
Process-qualifier

System-qualifier :: *System-name*

Block-qualifier :: *Block-name*

Process-qualifier :: *Process-name*

Concrete textual grammar

<message sequence chart document> ::=
mscdocument <document head> <document body> **endmscdocument** <end>
<document head> ::=
<msc document name> [**related to** <sdl reference>] <end>
<sdl reference> ::=
<sdl document identifier>
<identifier> ::=
[<qualifier>] <name>
<qualifier> ::=
<path item> { **/** <path item> }*
<path item> ::=
<scope unit class> <name>
<scope unit class> ::=
system
| **block**
| **process**

<document body> ::=
{ <message sequence chart> | <submsc> | <msc diagram> | <submsc diagram> }*

Semantics

A Message Sequence Chart document is a collection of Message Sequence Charts, and sub-Message Sequence Charts, optionally referring to a corresponding SDL-document.

4 Basic MSC

4.1 Message Sequence Chart

A Message Sequence Chart describes the message flow between instances, and optionally the actions triggered by the messages, depending on initial, intermediate and final conditions. One Message Sequence Chart describes a partial

behaviour of a system. Although the name *Message Sequence Chart* obviously originates from its graphical representation, it is used both for the textual and the graphical representation.

The Message Sequence Chart heading consists of the Message Sequence Chart name and (optionally) a list of the instances being contained in the Message Sequence Chart body.

Abstract grammar

Message-sequence-chart	::	*MSC-name* [*MSC-interface*] *MSC-body*
MSC-name	=	*Name*
MSC-interface	::	*Instance-list*
Instance-list	=	*Instance-declaration*+
Instance-declaration	::	*Instance-name* [*Instance-kind*]
Instance-name	=	*Name*
Instance-kind	=	*System-name* | *Block-name* | *Process-name* | *Service-name* | *Name*
System-name	::	*Name*
Block-name	::	*Name*
Process-name	::	*Name*
Service-name	::	*Name*
MSC-body	::	(*Instance-definition* | *Text-definition*)*
Text-definition	::	*Informal-text*

The *Instance-list* in the *MSC-interface*, if present, must contain the same instances as specified in the *MSC-body*.

Concrete textual grammar

<message sequence chart> ::=
msc <msc head> <msc body> **endmsc** <end>
<msc head> ::=
<message sequence chart name> <end> [<msc interface>]
<msc interface> ::=
inst <instance list> <end>
<instance list> ::=
<instance name> [**:** <instance kind>] [**,** <instance list>]
<instance kind> ::=
[<kind denominator>] <kind name>

<kind denominator> ::=

system | block | process | service

<msc body> ::=

{ <instance definition> | <text definition> }*

Concrete graphical grammar

<msc diagram> ::=

<msc symbol> ***contains*** { <msc heading> <msc body area> }

<msc body area> ::=

{ <instance area> | <external message area> | <text area> }*

<msc symbol> ::=

<frame symbol>

<frame symbol> ::=

T1007630-93/d03

<msc heading> ::=

msc <message sequence chart name>

Semantics

An MSC describes the communication between a number of system components, and between these components and the rest of the world, called environment. For each system component covered by an MSC there is an instance axis. The communication between system components is performed by means of messages. The sending and consumption of messages are two different events. It is assumed that the environment of an MSC is capable of receiving and sending messages from and to the Message Sequence Chart; no ordering of message events within the environment is assumed. Although the behaviour of the environment is non-deterministic, it is assumed to obey the constraints given by the Message Sequence Chart.

No global time axis is assumed for one Message Sequence Chart. Along each instance axis the time is running from top to bottom, however, we do not assume a proper time scale. If no coregion is introduced (see 5.1) a total time ordering of events is assumed along each instance axis. Events of different instances are ordered only via messages: a message must first be sent before it is consumed (see 4.3). No other ordering is prescribed. A Message Sequence Chart therefore imposes a partial ordering on the set of events being contained. A binary relation which is transitive, antisymmetric and reflexive is called partial order.

For the message inputs (labelled by ?mi) and outputs (labelled by !mi) of the Message Sequence Chart in Figure 1a) we derive the following ordering relation: !m2 < ?m2, !m3 < ?m3, !m4 < ?m4, ?m1 < !m2 < !m3 < ?m4, ?m2 < !m4 together with the transitive closure.

The partial ordering can be described in a minimal form (without an explicit representation of the transitive closure) by its connectivity graph [see Figure 1b)].

The semantics of an MSC can be related to the semantics of SDL by the notion of a reachability graph. Each sequentialization of an MSC describes a trace from one node to another node (or a set of nodes) of the reachability graph describing the behaviour of an SDL system specification. The reachability graph consists of nodes and edges. Nodes denote global system states. A global system state is determined by the values of the variables and the state of execution of each process and the contents of the message queues. The edges correspond to the events which are executed by the system, e.g. the sending and the consumption of a message or the execution of a task. A sequentialization of an MSC denotes one total ordering of events compatible with the partial ordering defined by the MSC.

Note that the reachability graph shows more events than the MSC. Thus, the sequentializations of the MSCs are not complete paths of the reachability graph.

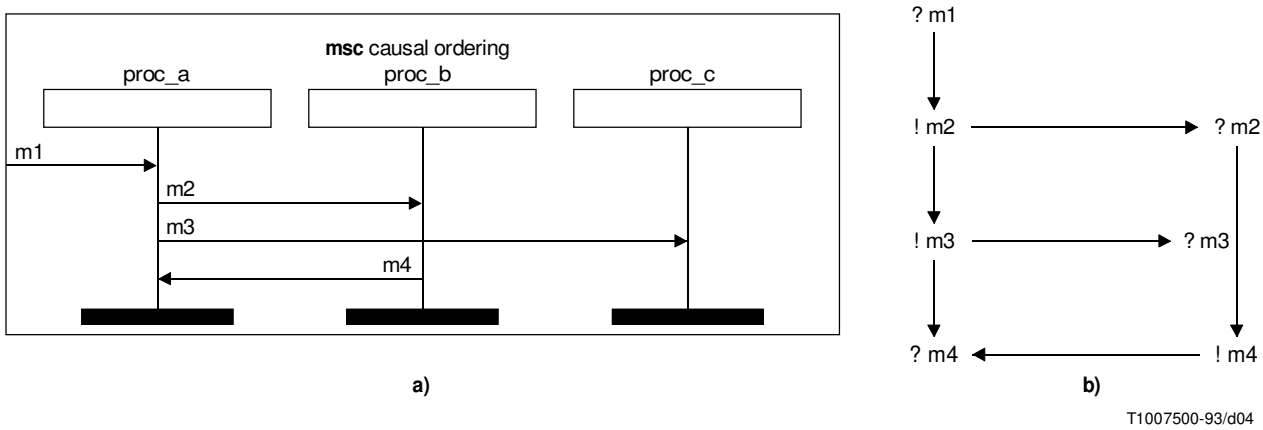


FIGURE 1/Z.120

Message Sequence Chart and corresponding connectivity graph

4.2 Instance

A Message Sequence Chart is composed of interacting instances of entities. An instance of an entity is an object which has the properties of this entity. Related to SDL, an entity may be an SDL-process, block or service. Within the instance heading the entity name, e.g. process name, may be specified in addition to the instance name. Within the instance body the ordering of events is specified. By means of the keyword **decomposed**, a sub-Message Sequence Chart with the same name may be attached to an instance.

Abstract grammar

Instance-definition :: *Instance-name*
[*Instance-kind*]
[**DECOMPOSED**]
Instance-event-list
[*Stop-node*]

Instance-event-list :: *Instance-event* *

Instance-event :: *Message-input* |
Message-output |
Create-node |
Timer-statement |
Coregion |
Action |
Condition

To each instance containing the keyword **decomposed** a corresponding *Submsc* (see 5.2) with the same name has to be specified. To each *Message-output* on a decomposed instance a corresponding *Message-output*, sent to the exterior of the *Submsc* must be specified. An analogous correspondence must hold for incoming messages.

Concrete textual grammar

<instance definition> ::=
instance <instance head> <instance body> **endinstance** <end>
<instance head> ::=
<instance name> [[**:**] <instance kind>] [**decomposed**] <end>

<instance body> ::=

<instance event list> [<stop>]

<instance event list> ::=

{ <message input> | <message output> | <create> | <timer statement>
| <coregion> | <action> | <condition> }*

Concrete graphical grammar

<instance area> ::=

<instance head area> ***is followed by*** <instance body area>

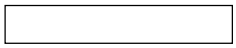
<instance head area> ::=

<instance head symbol> ***is associated with*** <instance heading>

<instance heading> ::=

<instance name> [**:** <instance kind>] [**decomposed**]

<instance head symbol> ::=



T1007630-93/d05

<instance body area> ::=

<instance axis symbol>
{ ***is followed by*** <instance event area>
is followed by <instance axis symbol> }*
is followed by { <instance end symbol> | <stop symbol> }

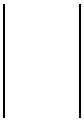
<instance axis symbol> ::=

<instance axis symbol1> | <instance axis symbol2>

<instance axis symbol1> ::=

T1007630-93/d06

<instance axis symbol2> ::=



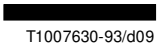
T1007630-93/d07

<instance event area> ::=

<message in area>
| <message out area>
| <create area>
| <timer area>
| <concurrent area>
| <action area>
| <condition area>

T1007630-93/d08

<instance end symbol> ::=



T1007630-93/d09

The <instance heading> may be placed above or inside of the <instance head symbol> or split such that the <instance name> is placed above the <instance head symbol> whereas the <instance kind> is placed inside. In the latter case the colon symbol is optional (and has to occur above if present) and the optional keyword **decomposed** has to occur inside if present. It is not allowed to mix <instance axis symbol1> and <instance axis symbol2>.

Semantics

Within the Message Sequence Chart body the instances are defined. The instance end symbol determines the end of the description of the instance within this MSC. It does not describe the termination of the instance (see 4.8: Process stop). Correspondingly, the instance head symbol determines the start of the description of the instance within the MSC. It does not describe the creation of the instance (see 4.7: Process creation).

In the context of SDL an instance may refer to a process (keyword **process**), service (keyword **service**) or block (keyword **block**). Outside of SDL, it may refer to any kind of entity. The instance definition provides an event description for message inputs and message outputs, actions, shared and local conditions, timer, process creation, process stop. Outside of coregions (see 5.1) a total ordering of events is assumed along each instance-axis. Within coregions no time ordering of events is assumed.

By means of the keyword **decomposed** a sub-Message Sequence Chart with the same name may be attached to an instance.

4.3 Message

A message within an MSC represents exchange of information between two instances or one instance and the environment.

A message exchanged between two instances can be split into two events: the message input and the message output; e.g. the second message in Figure 1a) can be split into !m2 (output) and ?m2 (input). Messages coming from the environment are represented by a message input, messages sent to the environment by a message output. In the textual representation the message input is represented by the keyword **in**, the message output by the keyword **out**, both followed by the message name and optionally a message instance name. To a message, parameters may be assigned between parentheses. The declaration of the parameter list is optional for the message input.

The correspondence between message outputs and message inputs has to be defined uniquely. In the textual representation normally the mapping between inputs and outputs follows from message name identification and address specification. In case where the message name and the address is not sufficient for a unique mapping the message instance name has to be employed. In the graphical representation a message is represented by an arrow.

Abstract grammar

Message-input	::	*Message-identification* *Sender-address*
Message-output	::	*Message-identification* *Receiver-address*
Message-identification	::	*Message-name* [*Message-instance-name*] [*Parameter-list*]
Message-name	=	*Name*
Message-instance-name	=	*Name*
Parameter-list	=	*Parameter-name+*
Parameter-name	=	*Name*

Sender-address = *Address*

Receiver-address = *Address*

Address = *Instance-name* |
ENVIRONMENT

It is not allowed that the *Message-output* is causally depending on its *Message-input* via other messages. This is the case if the connectivity graph (see 4.1) contains loops. If a *Parameter-list* is specified for a *Message-input* then it has to be specified also for the corresponding *Message-output*. The *Parameter-list*s have to be identical.

Concrete textual grammar

<message input> ::=
in <msg identification> **from** <address> <end>
<message output> ::=
out <msg identification> **to** <address> <end>
<msg identification> ::=
<message name> [**,** <message instance name>] [(<parameter list>)]
<parameter list> ::=
<parameter name> [**,** <parameter list>]
<address> ::=
<instance name> | **env**

For messages exchanged between instances the following rules must hold: To each <message output> one corresponding <message input> has to be specified and vice versa. In case where the <message name> and the <address> is not sufficient for a unique mapping the <message instance name> has to be employed.

Concrete graphical grammar

<message out area> ::=
<flow line symbol>
<flow line symbol> ::=

T1007630-93/d10

<message in area> ::=
<message symbol> ***is associated with*** <msg identification>
is connected to { <message out area> | <msc symbol> | <submsc symbol> }
[***is followed by*** <message out area>]
<message symbol> ::=

T1007630-93/d11

The mirror image of the <message symbol> is allowed.

<external message area> ::=
<message symbol> ***is associated with*** <msg identification>
is connected to { <message out area> { <msc symbol> | <submsc symbol> } }

NOTE – In the graphical representation the message instance name is not necessary for a unique syntax description.

Semantics

For an MSC the message-output denotes the message sending (corresponding to SDL-output), the message-input the message consumption (corresponding to SDL-input). No special construct is provided for message reception (input into the buffer). No type definition is attached to parameters within the parameter list.

If there is a message arrow head and a message origin on the same point of the instance axis then it is interpreted as if the message origin is drawn below the message arrow head.

4.4 Condition

A condition describes either a global system state (global condition) referring to all instances contained in the MSC or a state referring to a subset of instances (non-global condition). In the second case the condition may be local, i.e. attached to just one instance. In the textual representation the condition has to be defined for each instance to which it is attached using the keyword **condition** together with the condition name. If the condition refers to several instances, then the keyword **shared** together with the instance list denotes the set of instances by which the condition is shared. A global condition referring to all instances may be defined by means of the keyword **shared all**.

Abstract grammar

Condition	::	*Condition-name* [*Shared-information*]
Condition-name	=	*Name*
Shared-information	::	*Shared-instance-list* | **ALL**
Shared-instance-list	=	*Instance-name+*

Concrete textual grammar

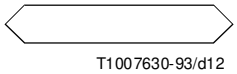
<condition> ::=
condition <condition name> [**shared** { <shared instance list> | **all** }] <end>

<shared instance list> ::=
<instance name> [**,** <shared instance list>]

To each <instance name> contained in a <shared instance list> of a <condition>, an instance with a corresponding shared <condition> must be specified. If instance *b* is contained in the <shared instance list> of a shared <condition> attached to instance *a* then instance *a* must be contained in the <shared instance list> of the corresponding shared <condition> attached to instance *b*. If instance *a* and instance *b* share the same <condition> then for each message exchanged between these instances, the <message input> and <message output> must be placed both before or both after the <condition>.

Concrete graphical grammar

<condition area> ::=
<local condition area> | <shared condition area>
<local condition area> ::=
<condition symbol> ***contains*** <condition name>
<condition symbol> ::=



T1007630-93/d12

<shared condition area> ::=

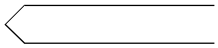
<condition left area> | <condition middle area> | <condition right area>

<condition left area> ::=

<condition left symbol> ***is associated with*** <condition name>

is connected to { <condition middle area> | <condition right area> }

<condition left symbol> ::=



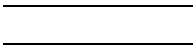
T1007630-93/d13

<condition middle area> ::=

<condition middle symbol>

is connected to { <condition middle area> | <condition right area> }

<condition middle symbol>::=

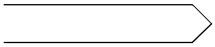


T1007630-93/d14

<condition right area> ::=

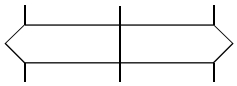
<condition right symbol>

<condition right symbol> ::=



T1007630-93/d15

A <shared condition area> is split into several areas: <condition left area>, <condition middle area>, <condition right area> which are horizontally aligned in order to form one single symbol associated with the <condition name>. The <local condition area> refers to just one instance, the <shared condition area> has a connection to other instances. If a shared <condition> crosses an <instance axis symbol> which is not involved in this condition the <instance axis symbol> is drawn through:



T1007630-93/d16

Semantics

Global conditions, representing global system states, refer to all instances involved in the MSC. For each Message Sequence Chart

- an initial global condition (global initial state);
- a final global condition (global final state); and
- intermediate global conditions (global intermediate states)

may be specified using the keyword **shared all** in the textual representation.

Initial, intermediate and final global conditions are not introduced merely for documentation purposes in the sense of comments or illustrations. In the case of a whole set of Message Sequence Charts these conditions have a well defined function. Global conditions determine possible continuations of Message Sequence Charts containing the same set of instances by means of condition identification: In the case where the final global condition of MSC1 is identical with the initial global condition of MSC2, MSC2 can be looked at as a continuation of MSC1.

Global conditions also define possible compositions and decompositions of MSCs. After decomposing an MSC at an intermediate global condition into MSC1 and MSC2, the global intermediate condition becomes global final condition for MSC1 and global initial condition for MSC2.

For an extensive employment of MSC-composition, global conditions referring to the complete system state are too restrictive. A partitioning into non-global conditions referring to a subset of instances is demanded. Local conditions attached to individual instances are a special case of non-global conditions. By means of non-global conditions also

combinations of Message Sequence Charts with different sets of instances can be defined whereby the continuation only refers to a common subset of instances.

It has to be noted that an MSC ending with a global condition may be continued also by an MSC starting with a non-global condition and vice versa, if both conditions refer to the same (sub)set of instances. As a generalisation of the rules for global conditions we define the continuation of two MSCs with a non-empty common set of instances in the following way: MSC2 is a continuation of MSC1 by means of (non)global conditions if for each instance which both MSCs have in common MSC1 ends with a (non)global condition and MSC2 begins with a corresponding (non)global condition. "Corresponding" in this context means that both conditions refer to the same subset of instances and both conditions agree with respect to name identification. In addition, each (non)global condition of MSC2 must have a corresponding (non)global condition in MSC1. Accordingly, MSC1 and MSC2 can be composed.

The other way round an MSC containing intermediate non-global conditions can be decomposed into MSC1 and MSC2. After decomposition the intermediate conditions become final conditions for MSC1 and initial condition for MSC2. The obtained MSCs again can be combined according to the rules set up above.

4.5 Timer

In MSCs either the setting of a timer and a subsequent timeout due to timer expiration or the setting of a timer and a subsequent timer reset (time supervision) may be specified. In the graphical representation the set symbol has the form of a small rectangle. The timeout-symbol is represented by a message sent from an instance to itself. The reset symbol is a modified timeout-symbol with a dashed input-arrow.

The specification of timer instance name and timer duration is optional both in the textual and graphical representation.

Abstract grammar

Timer-statement	=	*Set-node* |
		Reset-node |
		Timeout
Set-node	::	*Timer-name*
		[*Timer-instance-name*]
		[*Duration-name*]
Reset-node	::	*Timer-name*
		[*Timer-instance-name*]
Timeout	::	*Timer-name*
		[*Timer-instance-name*]
Timer-name	=	*Name*
Timer-instance-name	=	*Name*
Duration-name	=	*Name*

Concrete textual grammar

<timer statement> ::=
<set> | <reset> | <timeout>
<set> ::=
set <timer name> [**,** <timer instance name>] [(<duration name>)] <end>
<reset> ::=
reset <timer name> [**,** <timer instance name>] <end>

<timeout> ::=

timeout <timer name> [**,** <timer instance name>] <end>

For <set> and <timeout> the following rules must be obeyed: To each <set> a corresponding <timeout> or <reset> has to be specified and vice versa. In case where the <timer name> is not sufficient for a unique mapping the <timer instance name> has to be employed.

Concrete graphical grammar

<timer area> ::=

<timer set area> | <timer reset area> | <timeout area>

<timer set area> ::=

<set symbol>

<set symbol> ::=

T1007630-93/d17

<timer reset area> ::=

<reset symbol> ***is associated with*** <timer name> [(<duration name>)]

is connected to <timer set area>

<reset symbol> ::=

<reset symbol1> | <reset symbol2>

<reset symbol1> ::=

T1007630-93/d18

<reset symbol2> ::=

T1007630-93/d19

<timeout area> ::=

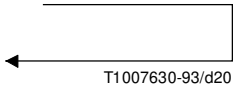
<timeout symbol> ***is associated with*** <timer name> [(<duration name>)]

is connected to <timer set area>

<timeout symbol> ::=

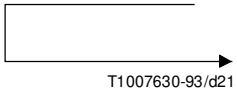
<timeout symbol1> | <timeout symbol2>

<timeout symbol1> ::=



T1007630-93/d20

<timeout symbol2> ::=



T1007630-93/d21

One side of the <set symbol> must coincide with the <instance axis symbol>. In case of the <instance axis symbol2>, the <set symbol> must be external to the column formed by <instance axis symbol2>.

It is the starting end of <reset symbol> and <timeout symbol> that should be connected to the <set symbol>, at the middle of that side which is opposite to the side that coincides with the <instance axis symbol>.

Semantics

Set and reset are timer constructs taken over from SDL. Set denotes setting the timer and reset denotes resetting of the timer. Timeout corresponds to the consumption of the timer signal in SDL.

4.6 Action

In addition to message exchange the actions may be specified in MSCs. An informal text is attached to the actions.

Abstract grammar

Action :: *Informal-Text*

Concrete textual grammar

<action> ::=

action <action text> <end>

Concrete graphical grammar

<action area> ::=

<action symbol> ***contains*** <action text>

<action symbol> ::=

T1007630-93/d22

In case where the instance axis has the column form, the width of the <action symbol> must coincide with the width of the column.

Semantics

An action describes an internal activity of an instance.

4.7 Process creation

Analogously to SDL, creation and termination of process instances may be specified within MSCs. A process instance may be created by another process instance. No message events before the creation must refer to the created instance.

Abstract grammar

Create-node :: *Instance-name*
*Parameter-name**

Concrete textual grammar

<create> ::=

create <instance name> [(<parameter list>)] <end>

To each <create> there must be a corresponding instance with the specified name. The <instance name> has to refer to an instance with type process if its type is specified. An instance can be created only once, i.e. within one MSC two or more <create>s with the same name must not appear. No message events before the creation must refer to the created instance.

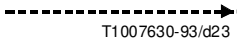
Concrete graphical grammar

<create area> ::=

<createline symbol> [***is associated with*** <parameter list>]

is connected to <instance head symbol>

<createline symbol> ::=



T1007630-93/d23

The mirror image of the <createline symbol> is allowed.

Semantics

Create defines the dynamic creation of a process instance by another. A create is immediately executed.

4.8 Process stop

The process stop in a sense is the counterpart to the process creation. However, a process instance can only stop itself whereas a process instance is created by another process instance.

Abstract grammar

Stop-node :: *( )*

The *Stop-node* at the end of an *Instance-definition* is allowed only for instances of type process.

Concrete textual grammar

<stop> ::=

stop <end>

Concrete graphical grammar

<stop symbol> ::=

T1007630-93/d24

Semantics

The stop at the end of an instance body causes the termination of this process instance.

5 Structural concepts

In this section, higher level concepts are introduced referring to generalized time ordering (coregion) and composition and decomposition of instances.

Instances in MSCs may refer to entities of different level of abstraction as indicated already by the keywords (**block**, **service**, **process**). Corresponding decomposition operations on instances can be defined determining the transition between different levels of abstraction. When instances are composed into one instance, then the total ordering of events along this instance must be relaxed in order to preserve the externally observable behaviour.

5.1 Coregion

The total ordering of events along each instance (see 4.1) in general may be not appropriate for entities referring to a higher level than SDL-processes.

Therefore a coregion is introduced for the specification of unordered events on an instance. Such a coregion in particular covers the practically important case of two or more incoming messages where the ordering of consumption may be interchanged.

Abstract grammar

Coregion :: *Coevent**

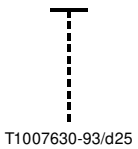
Coevent = *Message-input* |
Message-output

Concrete textual grammar

<coregion> ::=
concurrent { <coevent> }* **endconcurrent** <end>
<coevent> ::=
<message input> | <message output>

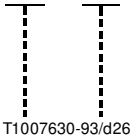
Concrete graphical grammar

<concurrent area> ::=
<coregion start symbol>
is followed by <coevent area>
is followed by <coregion end symbol>
<coregion start symbol> ::=
<coregion start symbol1> | <coregion start symbol2>
<coregion start symbol1> ::=



T1007630-93/d25

<coregion start symbol2> ::



T1007630-93/d26

<coevent area> ::=
{ { <message in area> | <message out area> } ***is followed by*** <coregion symbol> }*

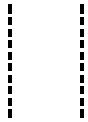
<coregion symbol> ::=

<coregion symbol1> | <coregion symbol2>

<coregion symbol1> ::=

T1007630-93/d27

<coregion symbol2> ::=



T1007630-93/d28

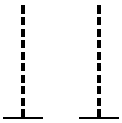
<coregion end symbol> ::=

<coregion end symbol1> | <coregion end symbol2>

<coregion end symbol1> ::=

T1007630-93/d29

<coregion end symbol2> ::=



T1007630-93/d30

<coregion start symbol1>, <coregion symbol1>, <coregion end symbol1> and <instance axis symbol1> must not be mixed with <coregion start symbol2>, <coregion symbol2>, <coregion end symbol2> and <instance axis symbol2> within one instance.

Semantics

For MSCs a total time ordering of events is assumed within each instance. By means of a coregion an exception to this can be made: events contained in the coregion are not ordered in time.

5.2 Sub-Message Sequence Chart

An instance of an MSC may be decomposed in form of a sub-Message Sequence Chart (sub-MSC), thus allowing a top-down specification.

A sub-MSC essentially has a structure analogous to an MSC. It is distinguished from the MSC by the keyword **submsc.** Characteristic for a sub-MSC is its relation to a decomposed instance containing the keyword **decomposed** and having the same name as the sub-MSC. The relation is provided by the messages connected to the exterior of the sub-MSC and the corresponding messages sent and consumed by the decomposed instance.

Abstract grammar

Submsc :: *Message-sequence-chart*

The name of a *Submsc* must be the same as the name of a corresponding *Instance-definition* containing the keyword **decomposed** within another *Message-sequence-chart* or *Submsc*. To each *Message-output,* sent to

the exterior of a *Submsc*, one corresponding *Message-output* on the decomposed instance has to be specified. An analogous correspondence must hold for incoming messages.

Concrete textual grammar

<submsc> ::=

submsc <msc head> <msc body> **endsubmsc** <end>

Concrete graphical grammar

<submsc diagram> ::=

<submsc symbol> ***contains*** { <submsc heading> <msc body area> }

<submsc symbol> ::=

<frame symbol>

<submsc heading> ::=

submsc <submsc name>

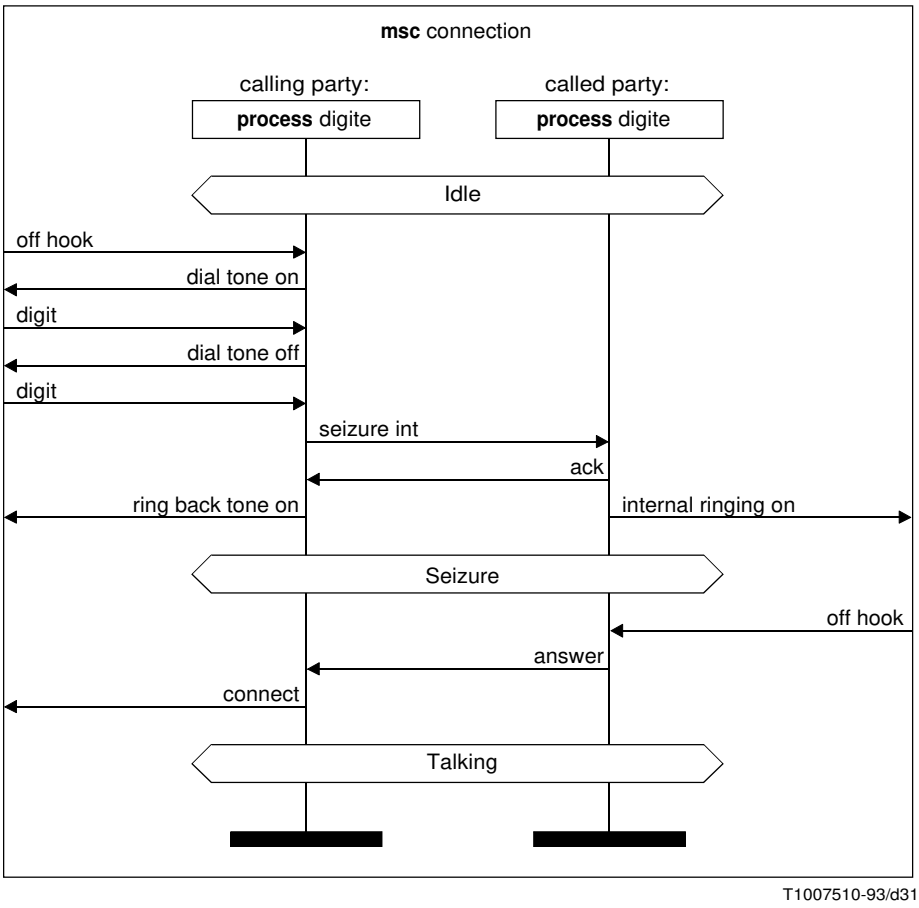
Semantics

A sub-Message Sequence Chart may be attached to an instance by means of the keyword **decomposed.** The sub-MSC represents a decomposition of this instance without affecting its observable behaviour. In the textual representation the messages addressed to and from the exterior of the sub-MSC are characterized by the address **env,** in the graphical representation by the connection with the sub-MSC border (frame symbol). Their connection with the external instances is provided by the messages sent and consumed by the decomposed instance, using message name identification. It must be possible to map the external behaviour of the sub-MSC to the messages of the decomposed instance. The ordering of message events specified on a decomposed instance must be preserved in the sub-MSC. Actions and conditions within the sub-MSC may be looked at as a refinement of actions and conditions in the decomposed instance. Contrary to messages, however, no formal mapping to the decomposed instance is assumed, i.e. the refinement of actions and conditions need not obey formal rules.

6 Message Sequence Chart examples

6.1 Standard message flow diagram

Example 6.1 shows a simplified connection set up within a switching system. The example shows the most basic MSC-constructs: (process) instances, environment, messages, global conditions.



msc connection; **inst** calling party: **process** digite, called party: **process** digite;

instance calling party: **process** digite;

condition Idle **shared all**;

in off hook **from env**;

out dial tone on **to env**;

in digit **from env**;

out dial tone off **to env**;

in digit **from env**;

out seizure int **to** called party;

in ack **from** called party;

out ring back tone on **to env**;

condition Seizure **shared all**;

in answer **from** called party;

out connect **to env**;

condition Talking **shared all**;

endinstance;

instance called party: **process** digite;

condition Idle **shared all;**

in seizure int **from** calling party;

out ack **to** calling party;

out internal ringing on **to env**;

condition Seizure **shared all**;

in off hook **from env**;

out answer **to** calling party;

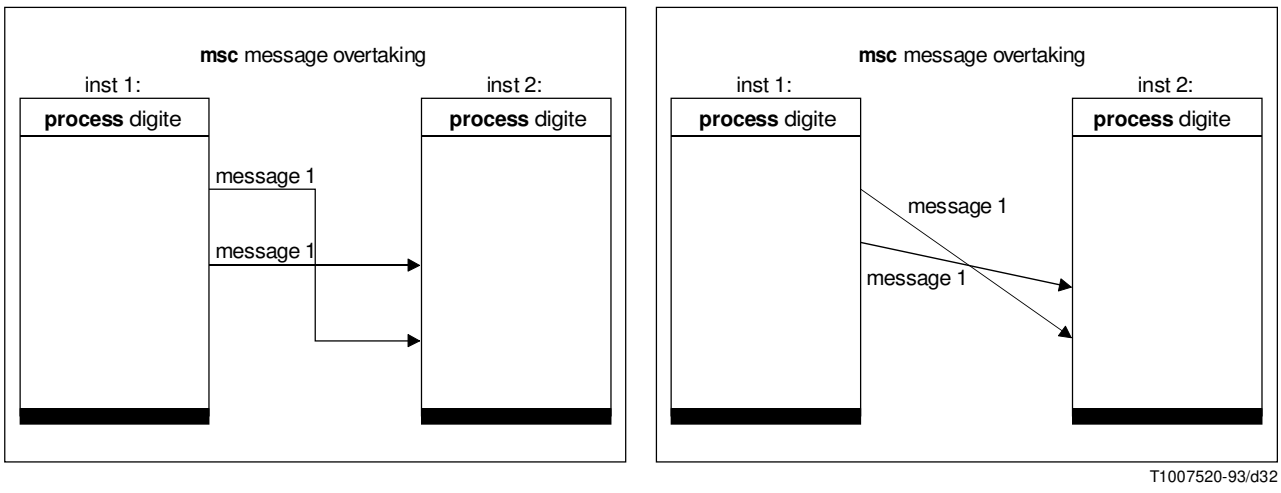
condition Talking **shared all**;

endinstance;

endmsc;

6.2 Message overtaking

Example 6.2 shows the overtaking of two messages with the same message name "message 1". In the textual representation the message instance names (a, b) are employed for a unique correspondence between message input and output. In the graphical representation messages either are represented by horizontal arrows, one with a bend to indicate overtaking or by crossing arrows with a downward slope.



msc message overtaking; **inst** inst 1, inst 2;

instance inst 1**: process** digite;

out message 1, a **to** inst 2;

out message 1, b **to** inst 2;

endinstance;

instance inst 2: **process** digite;

in message 1, b **from** inst 1;

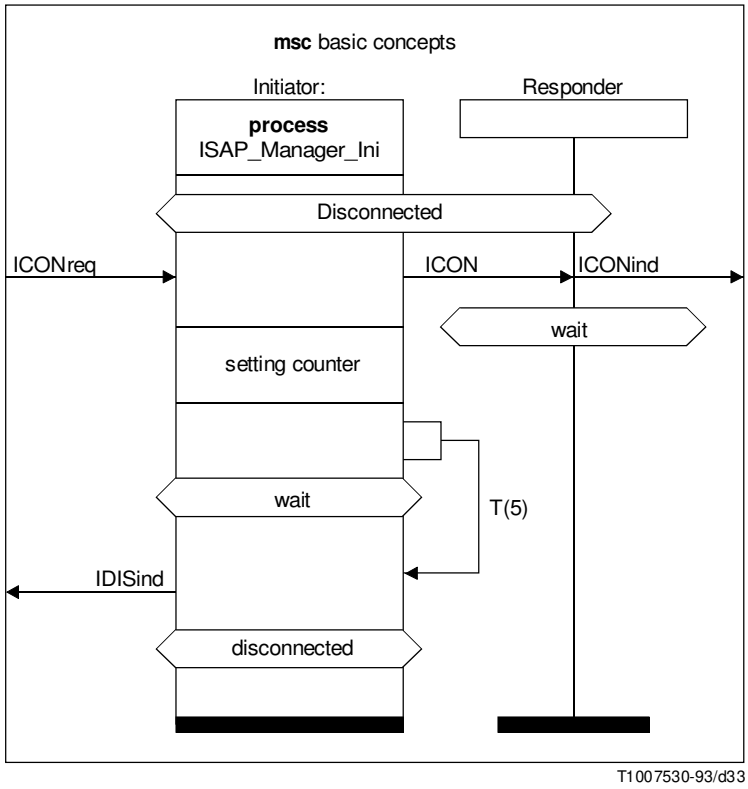
in message 1, a **from** inst 1;

endinstance;

endmsc;

6.3 MSC basic concepts

Example 6.3 contains the basic MSC constructs: instances, environment, messages, conditions, actions and timeout. In the graphical representation both types of instance symbols are used: the single line form and the column form.



msc basic concepts; **inst** Initiator: **process** ISAP_Manager_Ini, Responder;

instance Initiator: **process** ISAP_Manager_Ini;

condition Disconnected **shared all**;

in ICONreq **from env**;

out ICON **to** Responder;

action setting counter;

set T (5);

condition wait;

timeout T;

out IDISind **to env**;

condition disconnected;

endinstance;

instance Responder;

condition Disconnected **shared all**;

in ICON **from** Initiator;

out ICONind **to env**;

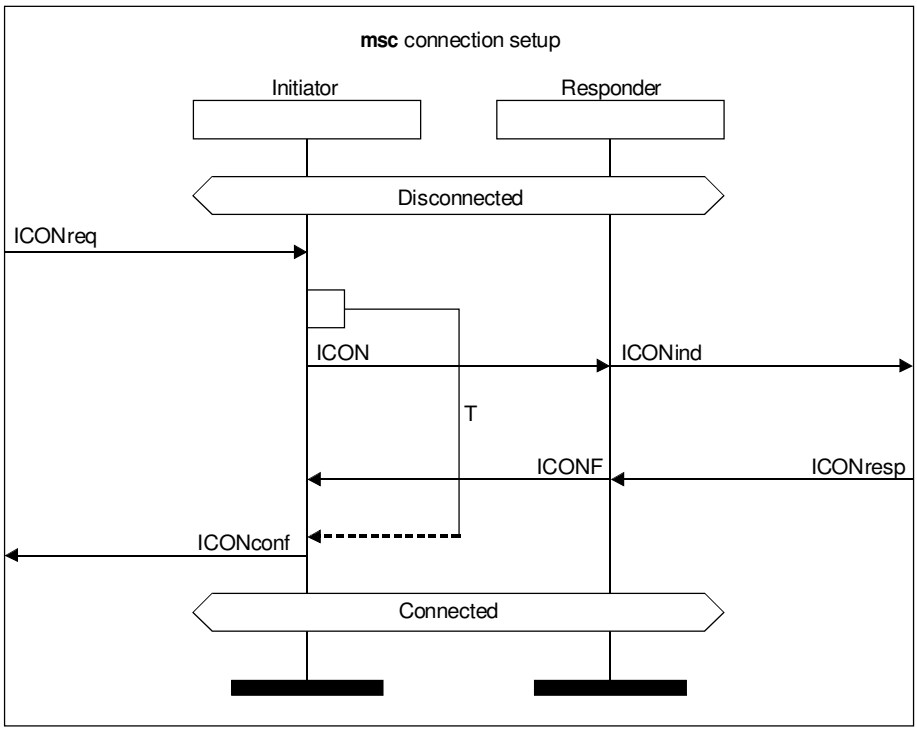
condition wait;

endinstance;

endmsc;

6.4 MSC with time supervision

The MSC connection set-up in example 6.4 contains a timer reset.



msc connection setup; **inst** Initiator, Responder;

instance Initiator;

condition Disconnected **shared all**;

in ICONreq **from env**;

set T;

out ICON **to** Responder;

in ICONF **from** Responder;

reset T;

out ICONconf **to env**;

condition Connected **shared all**;

endinstance;

instance Responder;

condition Disconnected **shared all**;

in ICON **from** Initiator;

out ICONind **to env**;

in ICONresp **from env**;

out ICONF **to** Initiator;

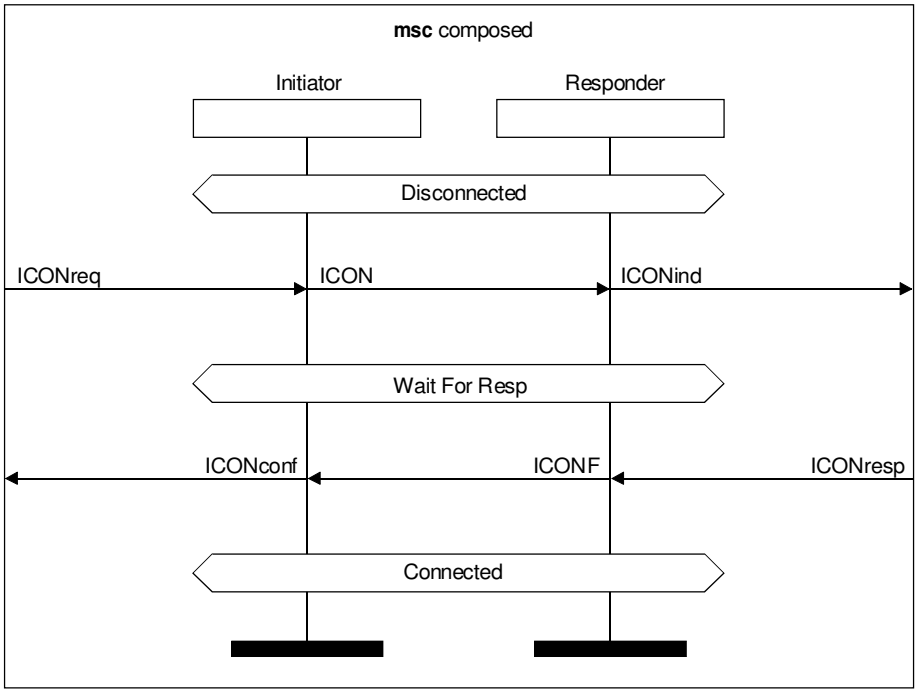
condition Connected **shared all**;

endinstance;

endmsc;

6.5 MSC-composition / MSC-decomposition

In example 6.5 the composition of MSCs by means of global conditions is demonstrated. The final global condition "Wait For Resp" of MSC connection request is identical with the initial global condition of MSC connection confirm. Therefore both MSCs may be composed to the resulting MSC composed.



T1007550-93/d35

msc composed; **inst** Initiator, Responder;

instance Initiator;

condition Disconnected **shared all**;

in ICONreq **from env;**

out ICON **to** Responder;

condition Wait For Resp **shared all**;

in ICONF **from** Responder;

out ICONconf **to env**;

condition Connected **shared all**;

endinstance;

instance Responder;

condition Disconnected **shared all**;

in ICON **from** Initiator;

out ICONind **to env;**

condition Wait For Resp **shared all**;

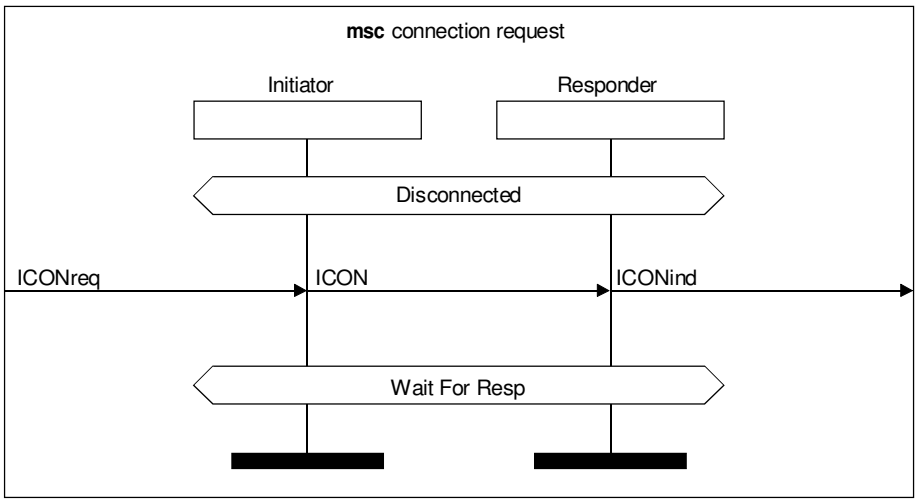
in ICONresp **from env**;

out ICONF **to** Initiator;

condition Connected **shared all**;

endinstance;

endmsc;

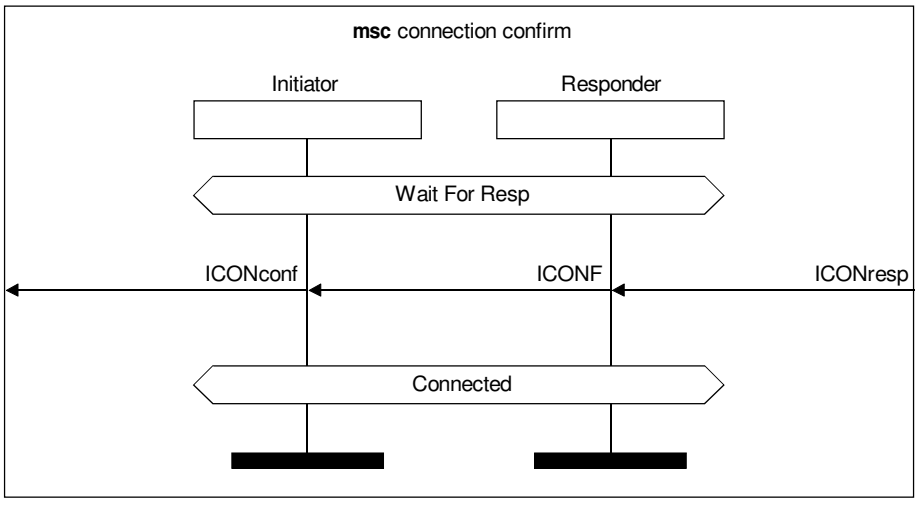


T1007560-93/d36

msc connection request; **inst** Initiator, Responder;

instance Initiator;
condition Disconnected **shared all**;
in ICONreq **from env**;
out ICON **to** Responder;
condition Wait For Resp **shared all**;
endinstance;
instance Responder;
condition Disconnected **shared all**;
in ICON **from** Initiator;
out ICONind **to env**;
condition Wait For Resp **shared all**;
endinstance;

endmsc;



T1007570-93/d37

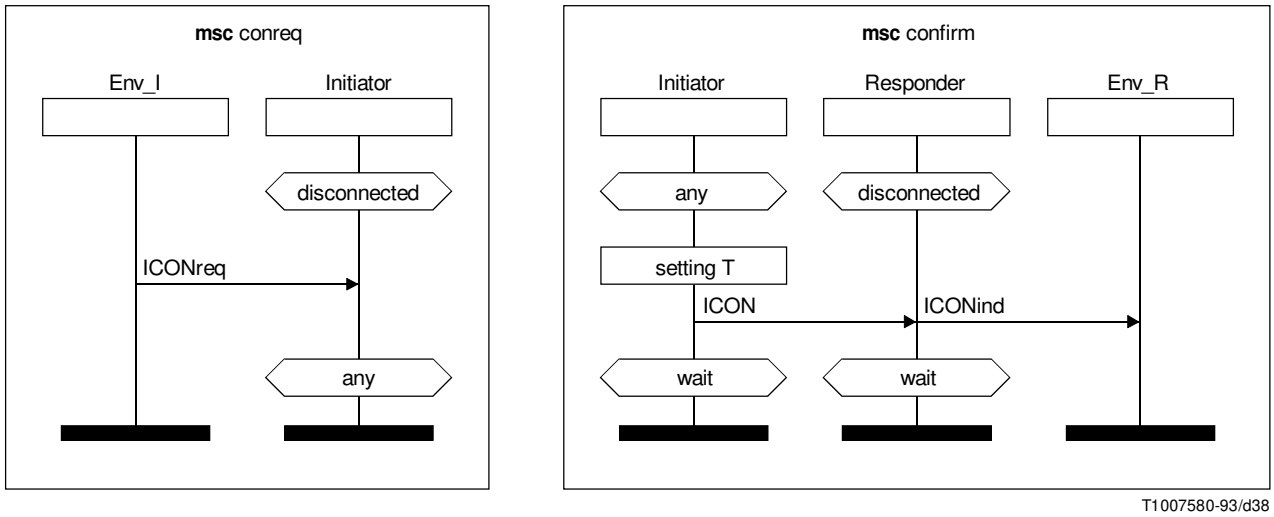
msc connection confirm; **inst** Initiator, Responder;

instance Inititator;
condition Wait For Resp **shared all**;
in ICONF **from** Responder;

out ICONconf **to env**;
condition Connected **shared all**;
endinstance;
instance Responder;
condition Wait For Resp **shared all**;
in ICONresp **from env**;
out ICONF **to** Initiator ;
condition Connected **shared all**;
endinstance;
endmsc;

6.6 Local conditions

In example 6.6 local conditions referring to one instance are employed to indicate possible continuations of this instance. Note that the two conditions with the same name "wait" in MSC "confirm" are discriminated by the instances to which they are attached.



T1007580-93/d38

msc conreq; **inst** Env_I, Initiator;

instance Env_I;
out ICONreq **to** Initiator;
endinstance;
instance Initiator;
condition disconnected;
in ICONreq **from** Env_I;
condition any;
endinstance;
endmsc;

msc confirm; i**nst** Initiator, Responder, Env_R;

instance Initiator;
condition any;
action setting T;
out ICON **to** Responder;
condition wait;
endinstance;
instance Responder;
condition disconnected;
in ICON **from** Initiator;
out ICONind **to** Env_R;
condition wait;
endinstance;

instance Env_R;

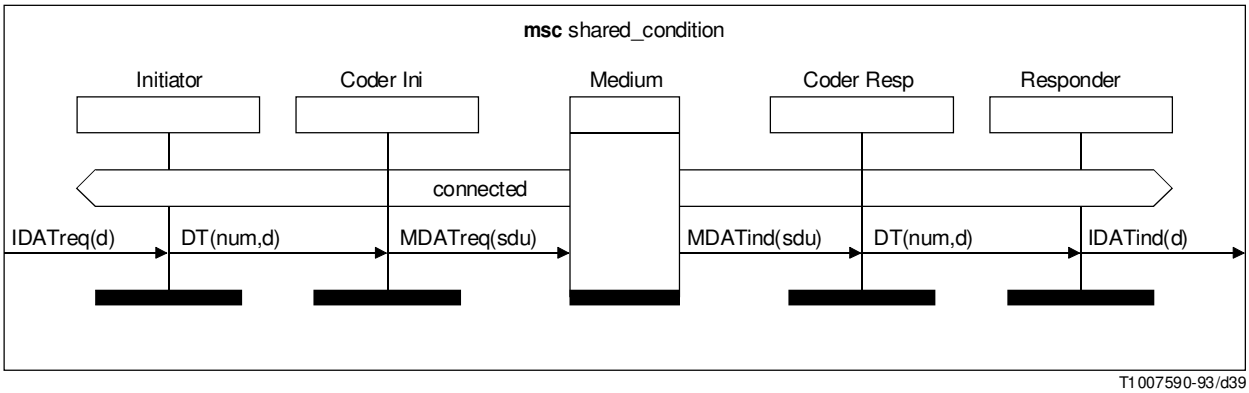
in ICONind **from** Responder;

endinstance;

endmsc;

6.7 Shared condition and messages with parameters

Example 6.7 contains the shared condition "connected". This condition is shared by the instances "Initiator" and "Responder". The instances "Coder Ini", "Medium", "Coder Resp" are not involved. In the textual representation the keyword **shared** together with a list of instances indicates the instances to which the condition is attached.



T1007590-93/d39

msc shared_condition; **inst** Initiator, Coder Ini, Medium, Coder Resp, Responder;

instance Initiator;

condition connected **shared** Responder;

in IDATreq(d) **from env**;

out DT(num,d) **to** Coder Ini;

endinstance;

instance Coder Ini;

in DT(num,d) **from** Initiator;

out MDATreq(sdu) **to** Medium;

endinstance;

instance Medium;

in MDATreq(sdu) **from** Coder Ini;

out MDATind(sdu) **to** Coder Resp;

endinstance;

instance Coder Resp;

in MDATind(sdu) **from** Medium;

out DT(num,d) **to** Responder;

endinstance;

instance Responder;

condition connected **shared** Initiator;

in DT(num,d) **from** Coder Resp;

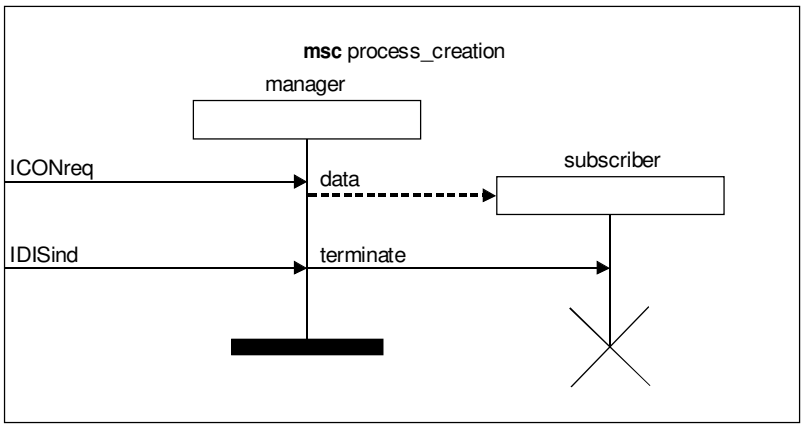
out IDATind(d) **to env**;

endinstance;

endmsc;

6.8 Creating and terminating processes

Example 6.8 shows the dynamic creation of the instance "subscriber" due to a connection request and corresponding termination due to a disconnection request.



T1007600-93/d40

msc process_creation; **inst** manager, subscriber;

instance manager;

in ICONreq **from env**;

create subscriber(data);

in IDISind **from env**;

out terminate **to** subscriber;

endinstance;

instance subscriber;

in terminate **from** manager;

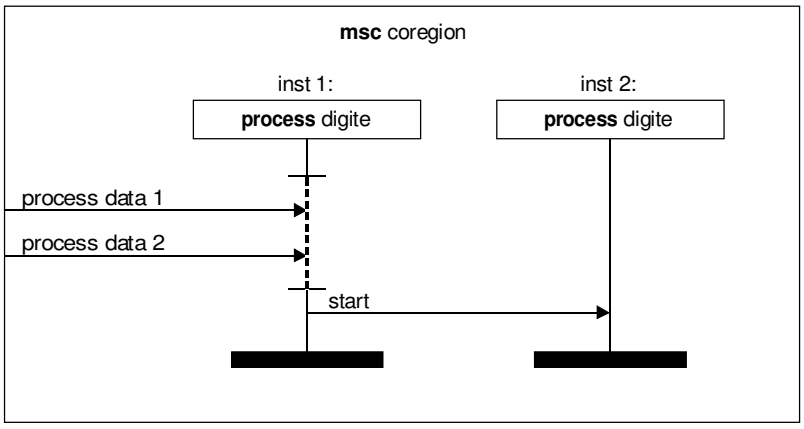
stop;

endinstance;

endmsc;

6.9 Coregion

Example 6.9 shows a concurrent region which shall indicate that the consumption of "process data 1" and the consumption of "process data 2" are not ordered in time, i.e. "process data 1" may be consumed before "process data 2" or the other way round.



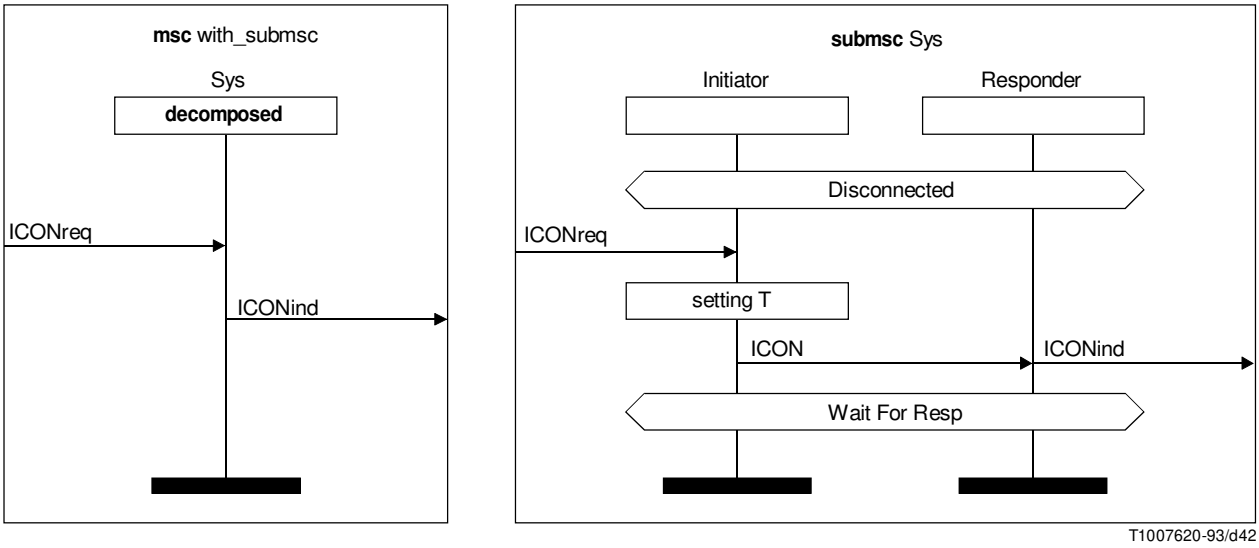
T1007610-93/d41

msc coregion; **inst** inst1, inst2;

instance inst1**: process** digite;
concurrent
in process data 1 **from env**;
in process data 2 **from env**;
endconcurrent;
out start **to** inst2;
endinstance;
instance inst2**: process** digite;
in start **from** inst1;
endinstance;
endmsc;

6.10 Sub-Message Sequence Chart

Example 6.10 contains the sub MSC "Sys". This sub MSC is attached to the instance "Sys" thus representing a decomposition of this instance.



msc with_submsc; **inst** Sys;

instance Sys **decomposed**;
in ICONreq **from env**;
out ICONind **to env**;
endinstance;
endmsc;

submsc Sys; **inst** Initiator, Responder;

instance Initiator;
condition Disconnected **shared all**;
in ICONreq **from env**;
action setting T;
out ICON **to** Responder;
condition Wait For Resp **shared all**;
endinstance;
instance Responder;
condition Disconnected **shared all**;
in ICON **from** Initiator;
out ICONind **to env**;
condition Wait For Resp **shared all**;
endinstance;
endsubmsc;

Annex A

Index

(This annex forms an integral part of this Recommendation)

The entries are the **<keyword>**s and the non-terminals from *Abstract grammar*, *Concrete textual grammar* and *Concrete graphical grammar*. **Bolded** page numbers refer to definitions of non-terminals.

<action area> 10; **17**
<action symbol> **17**
<action text> 17
<action> 10; **17**
<address> **12**
<alphanumeric> 3
<apostrophe> 3
<character string> 2; 4
<coevent area> **19**
<coevent> **19**
<comment area> **4**
<comment symbol> **4**
<comment> **4**
<composite special> 2
<concurrent area> 10; **19**
<condition area> 10; **13**
<condition left area> **14**
<condition left symbol> **14**
<condition middle area> **14**
<condition middle symbol> **14**
<condition name> 13; 14
<condition right area> **14**
<condition right symbol> **14**
<condition symbol> **13**
<condition> 10; **13**; 14
<coregion end symbol1> **20**
<coregion end symbol2> **20**
<coregion end symbol> 19; **20**
<coregion start symbol1> **19**; 20
<coregion start symbol2> **19**; 20
<coregion start symbol> **19**
<coregion symbol1> **20**
<coregion symbol2> **20**
<coregion symbol> **20**
<coregion> 10; **19**
<create area> 10; **18**
<create> 10; **17**; 18
<createline symbol> **18**
<dashed association symbol> **4**
<document body> **6**
<document head> **6**
<duration name> 15; 16
<end> **4**; 6; 7; 9; 12; 13; 15; 16; 17; 19; 21
<external message area> 8; **12**
<flow line symbol> **12**
<frame symbol> **8**; 21
<full stop> 3

<identifier> **6**
<instance area> 8; **10**
<instance axis symbol1> **10**; 11; 20
<instance axis symbol2> **10**; 11; 16; 20
<instance axis symbol> **10;** 14; 16; 17
<instance body area> **10**
<instance body> 9; **10**
<instance definition> 8; **9**
<instance end symbol> **10**
<instance event area> **10**
<instance event list> **10**
<instance head area> **10**
<instance head symbol> **10**; 11; 18
<instance head> **9**
<instance heading> **10**; 11
<instance kind> **7**; 9; 10; 11
<instance list> **7**
<instance name> 7; 9; 10; 11; 12; 13; 17; 18
<keyword> **2**
<kind denominator> 7; **8**
<kind name> 7
<lexical unit> **2**
<local condition area> **13**; 14
<message in area> 10; **12**
<message input> 10; **12**; 13; 19
<message instance name> 12
<message name> 12
<message out area> 10; **12**
<message output> 10; **12**; 13; 19
<message sequence chart document> **6**
<message sequence chart name> 7; 8
<message sequence chart> 6; **7**
<message symbol> **12**
<msc body area> **8**; 21
<msc body> 4; 7; **8**; 21
<msc diagram> 4; **8**
<msc document name> 6
<msc head> **7**; 21
<msc heading> **8**
<msc interface> **7**
<msc symbol> **8**; 12
<msg identification> **12**
<name> 6
<note> 2; 3; 4
<other character> 3
<parameter list> **12**; 17; 18
<parameter name> 12
<path item> **6**
<qualifier> **6**
<reset symbol1> **16**
<reset symbol2> **16**
<reset symbol> **16**; 17
<reset> **15**; 16
<scope unit class> 6
<sdl document identifier> 6
<sdl reference> **6**
<semicolon> 2; **3**; 4
<set symbol> **16**; 17
<set> **15**; 16

<shared condition area> 13; **14**
<shared instance list> **13**
<space> 3
<special> 2; 3
<stop symbol> 10; **18**
<stop> 10; **18**
<submsc diagram> 6; **21**
<submsc heading> **21**
<submsc name> 21
<submsc symbol> 12; **21**
<submsc> 6; **21**
<text area> **4**; 8
<text definition> **4**; 8
<text symbol> **4**
<text> **3**; 4
<timeout area> **16**
<timeout symbol1> **16**
<timeout symbol2> **16**
<timeout symbol> **16**; 17
<timeout> 15; **16**
<timer area> 10; **16**
<timer instance name> 15; 16
<timer name> 15; 16
<timer reset area> **16**
<timer set area> **16**
<timer statement> 10; **15**
<underline> 3
<word> 2
action 2; 17
Action 9; **17**
Address **12**
all 2; 13
block 2; 6
Block-name 6; **7**
Block-qualifier **6**
Coevent **19**
comment 2; 4
concurrent 2; 19
condition 2; 13
Condition 9; **13**
Condition-name **13**
Coregion 9; **19**
create 2, 17
Create-node 9; **17**
decomposed 2; 9; 10; 11; 20; 21
Duration-name **15**
endconcurrent 2; 19
endinstance 2; 9
endmsc 2; 7
endmscdocument 2; 6
endsubmsc 2; 21
endtext 2; 4
env 2; 21
from 2; 12
Identifier **6**
in 2; 12
Informal-text 7; 17
inst 2; 7

instance 2; 9
Instance-declaration **7**
Instance-definition 7; **9**; 18; 20
Instance-event **9**
Instance-event-list **9**
Instance-kind **7**; 9
Instance-list **7**
Instance-name **7**; 9; 12; 13; 17
Message-identification **11**
Message-input 9; **11**; 12; 19
Message-instance-name **11**
Message-name **11**
Message-output 9; **11**; 12; 19; 20; 21
Message-sequence-chart 5; **7**; 20
msc 2; 7; 8
MSC-body **7**
MSC-document **5**
MSC-document-name **5**
MSC-interface **7**
MSC-name **7**
mscdocument 2; 6
Name 5; 6; 7; 11; 13; 15
out 2; 12
Parameter-list **11**; 12
Parameter-name **11**; 17
Path-item **6**
process 2; 6; 8
Process-name 6; **7**
Process-qualifier **6**
Qualifier **6**
Receiver-address 11; **12**
referenced 2
related to 2; 6
reset 2; 15
Reset-node **15**
Sdl-document-identifier **6**
Sdl-reference 5; **6**
Sender-address 11; **12**
service 2; 8
Service-name **7**
set 2; 15
Set-node **15**
shared 2; 13
Shared-information **13**
Shared-instance-list **13**
stop 2; 18
Stop-node 9; **18**
submsc 2; 5; 9; 20; 21
system 2; 6; 8
System-name 6; **7**
System-qualifier **6**
text 2; 4
Text-definition **7**
timeout 2; 15; 16
Timer-instance-name **15**
Timer-name **15**
Timer-statement 9; **15**
to 2; 12