

International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.103

(12/2011)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE
ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Specification and
Description Language (SDL)

**Specification and Description Language –
Shorthand notation and annotation in SDL-2010**

Recommendation ITU-T Z.103



ITU-T Z-SERIES RECOMMENDATIONS
LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
User Requirements Notation (URN)	Z.150–Z.159
Testing and Test Control Notation (TTCN)	Z.160–Z.179
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Processing environment architectures	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Z.103

Specification and Description Language – Shorthand notation and annotation in SDL-2010

Summary

Scope/Objective

Recommendation ITU-T Z.103 defines the shorthand and annotation features of the Specification and Description Language. Together with Recommendations ITU-T Z.100, ITU-T Z.101, ITU-T Z.102, ITU-T Z.104, ITU-T Z.105 and ITU-T Z.106, this Recommendation is part of a reference manual for the language. The language defined in this document covers features of the language not included in Basic SDL-2010 in Recommendation ITU-T Z.101 or Comprehensive SDL-2010 in Recommendation ITU-T Z.102. Features defined in this Recommendation either do not have their own abstract grammar and are transformed to concrete grammar defined by Recommendations ITU-T Z.101, ITU-T Z.102 and ITU-T Z.104, or are annotations with no formal meaning.

Coverage

The Specification and Description Language has concepts for behaviour, data description and (particularly for larger systems) structuring. The basis of behaviour description is extended finite state machines communicating by messages. Data description is based on data types for values and objects. The basis for structuring is hierarchical decomposition and type hierarchies. A distinctive feature of the Specification and Description Language is the graphical representation. This Recommendation covers the features of the language such as shorthand and alternative graphical concrete syntax and macros that make SDL-2010 easier and more practical to use. The concrete grammar given is the graphical representation. The alternative textual programming representation is given in Recommendation ITU-T Z.106. This Recommendation does not provide a canonical syntax, but by applying the *Model* descriptions given a specification can be transformed to Comprehensive SDL-2010 defined in ITU-T Z.102, or (if no Comprehensive SDL-2010 features are used) to Basic SDL-2010 as defined in SDL-2010. It should be noted that in both Basic and Comprehensive SDL-2010 the details of expressions, data definitions and action language, are defined in Recommendation ITU-T Z.104.

Applications

The Specification and Description Language is applicable within standard bodies and industry. The main application areas for which the Specification and Description Language has been designed are stated in ITU-T Z.100, but the language is generally suitable for describing reactive systems. The range of application is from requirement description to implementation. The features of the language defined in Recommendation ITU-T Z.103 are cumulative with, and rely on, the features defined in Recommendations ITU-T Z.101, ITU-T Z.102 and ITU-T Z.104. Therefore the language defined in Recommendation ITU-T Z.103 is practical for application to all kinds of real time systems (except if ASN.1 is also required in which case features defined in Recommendation ITU-T Z.105 should also be used, or if the specification needs to be in text only or interchange format in which case features defined in Recommendation ITU-T Z.106 should also be used).

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T Z.103	2011-12-22	17

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. 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 Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

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

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2012

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Table of Contents

	Page
1	Scope and objective 1
1.1	Objective..... 1
1.2	Application 1
2	References..... 1
3	Definitions 2
3.1	Terms defined elsewhere 2
3.2	Term defined in this Recommendation 2
4	Abbreviations and acronyms 2
5	Conventions 2
6	General rules 3
6.1	Lexical rules – text in a comment area..... 3
6.2	End terminator, comment and comment area..... 3
6.3	Text extension 4
6.4	Solid association symbol 5
6.5	The metasymbol <i>is followed by</i> and flow line symbol without arrowhead 5
6.6	Names and identifiers, name resolution and visibility – additional diagrams 6
6.7	Macro..... 6
6.8	Informal text 6
6.9	Text symbol 6
6.10	Frame symbol, page numbers and multiple page diagrams 6
6.11	Drawing rules 7
7	Organization of Specification and Description Language specifications..... 8
7.1	Framework..... 8
7.2	Package..... 8
7.3	Referenced definition 10
8	Structural concepts..... 11
8.1	Types, instances and gates..... 11
8.2	Type references and operation references 13
8.3	Context parameters 15
8.4	Specialization 15
9	Agents 15
9.1	System 19
9.2	Block..... 19
9.3	Process 19
9.4	Procedure..... 20
9.5	Agent and composite state reference 21

	Page
10	Communication 23
	10.1 Channel..... 23
	10.2 Connection..... 25
	10.3 Signal..... 26
	10.4 Signal list area 26
11	Behaviour..... 26
	11.1 Start..... 26
	11.2 State 26
	11.3 Input..... 28
	11.4 Priority input..... 29
	11.5 Continuous signal 30
	11.6 Enabling condition..... 30
	11.7 Save 30
	11.8 Implicit transition 30
	11.9 Spontaneous transition..... 31
	11.10 Label (connector name)..... 31
	11.11 State machine and composite state 32
	11.12 Transition..... 34
	11.13 Action 36
	11.14 Statement lists..... 38
	11.15 Timer 40
12	Data..... 40
	12.1 Data definitions 41
	12.2 Use of data..... 41
	12.3 Active use of data 41
13	Generic system definition 41
	13.1 Optional definition..... 41

Introduction

This Recommendation is part of the ITU-T Z.100 to ITU-T Z.106 series of Recommendations that give the complete language reference manual for SDL-2010. The text of this Recommendation is stable. For more details see Recommendation ITU-T Z.100.

Recommendation ITU-T Z.103

Specification and Description Language – Shorthand notation and annotation in SDL-2010

1 Scope and objective

This Recommendation defines features of the Specification and Description Language that make the language practical to use. The features defined in this document cover the shorthand notation and annotation for the language, which is defined further in the other Recommendations of the ITU-T Z.100 series. Together with Recommendations [ITU-T Z.100], [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.104], [ITU-T Z.105] and [ITU-T Z.106], this Recommendation forms a reference manual for the language.

1.1 Objective

The objective of this Recommendation is to define the features of the Specification and Description Language that are written using shorthand notation or are used to add annotation. The features defined in this Recommendation add to SDL-2010 features defined for Basic SDL-2010 or Comprehensive SDL-2010 or the data and action language in SDL-2010. The features do not rely on features defined for using ASN.1 or the use of the common interchange format.

Shorthand notation is a concrete grammar that is transformed by a model into further concrete grammar before the representation as abstract grammar is considered. The use of shorthand notation is important to make the use of the language practical, because the shorthand is easy to read, write and understand as well as being concise. A construct written using the concrete grammar defined by this Recommendation is allowed to contain more than one shorthand; in some cases a shorthand is transformed through one (or more) other shorthand forms before a concrete grammar that does not contain any shorthand is obtained from which the abstract grammar is determined. Where the order of transformation is important, this is defined.

An annotation adds information to a specification in the language that does not change the formal meaning of the specification. While such information does not change the formal meaning as far as the Specification and Description Language is concerned, annotations are important to aid understanding of the purpose and intention of a specification or how an SDL-2010 specification interacts with or is interpreted by another language (such as the Message Sequence Chart language).

1.2 Application

This Recommendation is part of the reference manual for the Specification and Description Language. The part of the language defined by this Recommendation includes additional concrete syntax for shorthand notations and *Model* sections for the shorthand. It also includes the concrete syntax for additional annotation not included Basic SDL-2010 or Comprehensive SDL-2010 or the data and action language in SDL-2010. The part of the language defined in this Recommendation is required to make the use of the language practical.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

- [ITU-T Z.100] Recommendation ITU-T Z.100 (2011), *Specification and Description Language – Overview of SDL-2010*.
- [ITU-T Z.101] Recommendation ITU-T Z.101 (2011), *Specification and Description Language – Basic SDL-2010*.
- [ITU-T Z.102] Recommendation ITU-T Z.102 (2011), *Specification and Description Language – Comprehensive SDL-2010*.
- [ITU-T Z.104] Recommendation ITU-T Z.104 (2011), *Specification and Description Language – Data and action language in SDL-2010*.
- [ITU-T Z.105] Recommendation ITU-T Z.105 (2011), *Specification and Description Language – SDL-2010 combined with ASN.1 modules*.
- [ITU-T Z.106] Recommendation ITU-T Z.106 (2011), *Specification and Description Language – Common interchange format for SDL-2010*.
- [ITU-T Z.111] Recommendation ITU-T Z.111 (2008), *Notations and guidelines for the definition of ITU-T languages*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere: the definitions of [ITU-T Z.100] apply.

3.2 Term defined in this Recommendation

None.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms: the abbreviations and acronyms defined in Recommendation [ITU-T Z.100] apply.

5 Conventions

The conventions defined in [ITU-T Z.100] apply, which includes the conventions defined in [ITU-T Z.111].

Where an abstract or concrete syntax rule is defined in this Recommendation with the same name as a rule in [ITU-T Z.101] or [ITU-T Z.102], the rule given here replaces the rule in [ITU-T Z.101] or [ITU-T Z.102]. Any *Abstract grammar* or *Concrete grammar* conditions, *Semantics* and *Model* defined on a named rule in [ITU-T Z.101] or [ITU-T Z.102] apply to the redefined rule, unless specifically defined otherwise in this Recommendation. Any contradiction between [ITU-T Z.101] or [ITU-T Z.102] and this Recommendation is an error in the definition of SDL-2010 that needs to be resolved by further study.

6 General rules

6.1 Lexical rules – text in a comment area

An additional lexical rule is defined for text in a <comment area>:

```
<text> ::=
    { <general text character> | <special> | <semicolon> | <apostrophe> }*
```

If the extended character set is used (see [ITU-T Z.102]), all printing characters are permitted to appear freely in <text> in a <comment area>.

A <text> sequence of lexical units is used in a <comment area> where it is treated as annotation. In a <comment area> the <text> is a single lexical unit and the sequence of characters for the <text> is terminated by reaching the end of the text contained in the <comment symbol> or <text extension symbol> of the <comment area>.

6.2 End terminator, comment and comment area

End terminator and comment are as defined in [ITU-T Z.101] and [ITU-T Z.102].

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

Concrete grammar

With text, three forms of comments are used. The first form is the <note>, which is considered only as a unit at the lexical level and therefore does not appear outside lexical rules (see [ITU-T Z.101]). The second and third forms are a <comment> in an <end> and <comment body> as defined in [ITU-T Z.101]. The places where <end> (and therefore <comment>) and <comment body> are allowed are defined in the concrete syntax rules.

With symbols a <comment area> with <text> is used.

```
<comment area> ::=
    <dashed association symbol> is connected to
    { <comment symbol> | <text extension symbol> } contains <text>
```

```
<comment symbol> ::=
```



```
<dashed association symbol> ::=
```



A <dashed association symbol> is a line symbol.

It is allowed that a <comment area> is connected to any graphical symbol except a <comment symbol> or <text extension symbol>. One end of the <dashed association symbol> shall be connected to the middle of the vertical segment of the <comment symbol>. The graphical symbol shall connect to the other end of the <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.

Model

A <text extension symbol> in a <comment area> is treated as a <comment symbol>, and the <comment area> is distinguished from a <text extension area> by the use of the <dashed association symbol>.

In principle the non-terminal <comment area> is added to the concrete syntax for pages (such as <package page> or <block type page>) so that it is possible to add <comment area> to any symbol on the page (except a <comment symbol> or <text extension symbol>). For example, a modified form of <package page> is:

```
<package page> ::=
    <frame symbol> contains
    { <package heading> <page number area>
      {
        { <package text area> [ is connected to <comment area> ] }*
        { <diagram in package> [ is connected to <comment area> ] }* } set }
    [ is associated with <package use area> ]
```


For conciseness, it is assumed that this example is used (as further extended to include the syntax for <text extension area> as given in clause 6.3) as a template to derive the modified syntax for other pages such as <block type page>.

The <comment area> connected to a symbol has no formal meaning and is therefore removed before determining the abstract grammar.

6.3 Text extension

Concrete grammar

```
<text extension area> ::=
    <solid association symbol> is connected to
    <text extension symbol> contains
    { {
      | <name>
      | <integer name>
      | <real name>
      | <character string>
      | <hex string>
      | <bit string>
      | <note>
      | <comment body>
      | <composite special>
      | <special>
      | <semicolon>
      | <other character>
      | <quoted operation name>
      | <keyword> } *
    }
```

```
<text extension symbol> ::=
    
```

It is allowed to connect a <text extension area> to any graphical symbol that contains text to be analysed as lexical units (except another <text extension symbol>). One end of the <solid association symbol> shall be connected to the middle of the vertical segment of the <text extension symbol>. The graphical symbol shall connect to the other end of the <solid association symbol>.

The <text extension symbol> is considered as a closed symbol by completing (in imagination) the rectangle.

The <text extension symbol> is allowed to contain a list of lexical items (such as <name>, <note> or <keyword>). Whether a particular lexical item is allowed will depend on the graphical symbol that connects to the <text extension area> and the model given below.

Model

In principle, the non-terminal <text extension area> is added to the concrete syntax for pages (such as <package page> or <block type page>) so that <text extension area> is allowed to be added to any symbol (except a <comment symbol> or <text extension symbol>) on the page. It is allowed for a symbol to have both a <text extension area> and a <comment area>. For example, a modified form of <package page> is:

```
<package page> ::=
    <frame symbol> contains
        { <package heading> <page number area>
            { <package text area>
                [ is connected to <text extension area> ]
                [ is connected to <comment area> ] }*
            { <diagram in package>
                [ is connected to <text extension area> ]
                [ is connected to <comment area> ] }*
            } set }
    [ is associated with <package use area> ]
```

Each <comment area> is connected to the symbol for the <package text area> or the <diagram in package> (not to the optional <text extension area>). The <comment area> is removed as described in clause 6.2 before determining the abstract grammar.

For conciseness, it is assumed that this example is used as a template to derive the modified syntax for other pages such as <block type page>.

The list of lexical items contained in the <text extension symbol> is a continuation of text within the graphical symbol. The graphical symbol is therefore transformed by adding the list of lexical items in the <text extension symbol> to the end of the items in the graphical symbol, and the <text extension symbol> is removed. The validity and meaning of any particular lexical item therefore depends on the particular graphical symbol and the context in which it is used. This transformation is applied before any transformation relating to the graphical symbol or its contents.

Sometimes a symbol contains more than one item of text. For example, the <block symbol> for a <typebased block definition> contains text for the <typebased block heading> and for each <gate> item. In this case, it is the text item that appears first in the syntax that is extended. The <text symbol> is a special case, because the contents are always treated as a single text item; therefore the text extends the whole contents of the <text symbol>.

6.4 Solid association symbol

See [ITU-T Z.101].

6.5 The metasymbol *is followed by* and flow line symbol without arrowhead

The flow line symbol without arrowhead is added to make specifications more readable by omitting unnecessary arrowheads.

Concrete grammar

The concrete syntax for <flow line symbol> is extended to allow a <flow line symbol without arrowhead> as an alternative.

```
<flow line symbol> ::=
    <flow line symbol without arrowhead>
    | <flow line symbol with arrowhead>
```

```
<flow line symbol without arrowhead> ::=
```

<flow line symbol without arrowhead> is a line symbol.

A <flow line symbol without arrowhead> shall not be used in a diagram where it merges with another <flow line symbol>, or an <out connector symbol> or a <nextstate area>.

A <flow line symbol without arrowhead> has the same representation as a <solid association symbol>. If there is any ambiguity in a diagram, the symbol is taken to be a <solid association symbol>, and if a <flow line symbol> is required instead (representing *is followed by*) a <flow line symbol with arrowhead> should be used.

Model

A <flow line symbol without arrowhead> is transformed to a <flow line symbol with arrowhead>.

6.6 Names and identifiers, name resolution and visibility – additional diagrams

An identifier is used to identify an item of a particular entity kind with a specific name defined in a specific scope unit context. As well as a *Name*, an *Identifier* has a *Qualifier* that gives the path to the definition of the identifier. In the concrete syntax the <qualifier> is usually minimized or omitted if it is possible to resolve the identity without full qualification, to avoid the specification becoming unreadable due to all the qualification. The minimization or omission of qualifiers is a shorthand notation that should in principle appear in this Recommendation, but is fully described in [ITU-T Z.101] because of the importance of this shorthand for usage of the language.

This Recommendation introduces concrete syntax for the following scope unit kinds in addition to those defined in Basic SDL-2010:

system	<system diagram>
block	<block diagram>
process	<process diagram>

6.7 Macro

See [ITU-T Z.102].

6.8 Informal text

See [ITU-T Z.101].

6.9 Text symbol

See [ITU-T Z.101].

6.10 Frame symbol, page numbers and multiple page diagrams

Basic SDL-2010 is extended to allow multiple page diagrams.

General rules are described here.

Every page of a diagram has a heading, which identifies the diagram kind (such as **process type**) and the identity (usually as an unqualified name if this is unambiguous) for the corresponding abstract grammar item. The concrete grammar for multiple pages is the same for every page, and it is required that the information given is consistent on every page. Consistency means that in a heading if an item (such as **abstract** as a <type preamble>) appears on one page, it should appear identically on all other pages, but for convenience it is permitted to omit additional information on all but one page and the combined heading formed from a combination of all pages. It is suggested that at least one page (with an appropriate <page number>) has the full heading. The formal correct presentation is for every diagram page to have the same identical full heading.

If there is a <package use area> the formal correct presentation is for every diagram page to have the same <package use area>, but for convenience it is permitted to omit information in the <package use area> of some pages or to omit the <package use area> altogether on a page, provided the package use information for the diagram is given in at least one <package use area>. It is suggested that one page with the full heading for the diagram also has a <package use area> that gives the full package use information.

Model

If a diagram has several pages, and the headings are different on different pages a consistent heading is derived (if possible; otherwise the diagram is probably invalid).

The full formal correct <package use area> is derived from combining the content of every <package use area> on all the pages. Any duplication of package use information is removed.

A diagram with several pages is logically replaced by a diagram with one larger page with the consistent heading and an arbitrary <page number>. Forming the single page diagram is a topological transformation, where to join two pages together, a break is made in the frame of each and the line of the frames are joined so there is a single line around the content of both original pages. Additional diagrams are added in the same way until a single line surrounds all the contents of all the original pages. The items related to the frames remain connected, attached or associated with the new frame, and are moved with the frame as it is reshaped to make it rectangular again. The headings and page numbers are removed and replaced by the consistent heading and arbitrary <page number> in the top left and right corners respectively.

6.11 Drawing rules

The size of the graphical symbols is chosen by the user (in principle, but tools possibly limit user choice).

Symbol boundaries shall not overlay or cross except line symbols, which are allowed to cross each other. There is no logical association between symbols that do cross. The following are line symbols:

- <channel symbol 1>
- <channel symbol 2>
- <create line symbol>
- <dashed association symbol>
- <flow line symbol>
- <flow line symbol without arrowhead>
- <solid association symbol>

Vertical mirror images of <input symbol>, <output symbol>, <internal input symbol>, <internal output symbol>, <priority input symbol>, <comment symbol> and <text extension symbol> are allowed.

Text within a graphical symbol is read from left to right, starting from the upper left corner. The right-hand edge of the symbol is interpreted as a newline character, indicating that the reading continues at the leftmost point of the next line (if any). Text within a graphical symbol should be contained within the symbol (that is, not overlapping the lines forming the border of the symbol or boundary of the symbol for unclosed symbols such as the <comment symbol>), so that no part of the text is read as associated with some other symbol. However, a symbol often contains more than one text item, for example, a <typebased block definition> contains text for both a <typebased block heading> and a number of <gate> items. Different text items should not overlap, should be sufficiently separated to be distinct, and in a diagram should be unambiguously nearer to any construct they are related to than any other text item.

7 Organization of Specification and Description Language specifications

7.1 Framework

The concrete syntax of a system specification is extended to allow agent diagrams as well as typebased agent definitions.

Concrete grammar

```
<system specification> ::=
    <agent diagram>
    | <typebased agent definition>[ is associated with <package use area> ]
```

Model

A <system specification> being an <agent diagram> that is a <process diagram> or a <typebased agent definition> that is a <typebased process definition> is derived syntax for a <system diagram> having the same name as the process, containing implicit channels and containing the <process diagram> or <typebased process definition> as the only definition. Using this model satisfies the constraint the *Agent-kind* is **SYSTEM**. The transformation of a <process diagram> into a <system diagram> containing the <process diagram> is carried out before the transformation of the contained <process diagram> into a <typebased process definition> and a corresponding <process type diagram>.

A <system specification> being an <agent diagram> that is a <block diagram> or a <typebased agent definition> that is a <typebased block definition> is derived syntax for a <system diagram> having the same name as the block, containing implicit channels and containing the <block diagram> or <typebased block definition> as the only definition. Using this model satisfies the constraint the *Agent-kind* is **SYSTEM**. The transformation of a <block diagram> into a <system diagram> containing the <block diagram> is carried out before the transformation of the contained <block diagram> into a <typebased block definition> and a corresponding <block type diagram>.

A <system diagram> is derived syntax for a <typebased system definition> and a corresponding <system type diagram>.

A <package use area> associated with a <typebased agent definition> of a <system specification> is derived syntax for a <package use area> associated with the <system diagram> derived from the <typebased agent definition>.

7.2 Package

Package is extended to allow textual references (to agents types, contained packages, procedures and interface definitions), contained procedure definitions and create line annotation. Package reference is extended to allow it to be used as annotation that references a package defined elsewhere.

Concrete grammar

```
<package reference area> ::=
    <package symbol> contains [ <qualifier> ] <package name>
```

The <package reference area> is extended compared to Basic SDL-2010 to include an optional <qualifier>. The optional <qualifier> and <package name> of a <package reference area> shall be contained in the lower rectangle of <package symbol>.

If in a <package reference area> there is a <qualifier> before the <name> of the referenced package and the <qualifier> does not identify the scope unit directly enclosing the package reference, the package reference and referenced package are in different scope units. In this case, it is possible to remove the <package reference area> from the model without changing the semantics of the model

and the reference is a form of annotation providing consistent information about the referenced package with no representation in the abstract grammar. Otherwise, for a <package reference area> where the <qualifier> of the <name> of the referenced package is omitted or identifies the scope unit directly enclosing the <package reference area>, the package reference and referenced package are logically in the same scope unit: the scope unit containing the reference.

```
<package text area> ::=
    <text symbol> contains
    {
        <signal definition list>
    |   <data definition>
    |   <remote procedure definition>
    |   <remote variable definition>
    |   <select definition>
    |   <macro definition>
    |   <agent type reference>
    |   <package reference>
    |   <procedure definition>
    |   <procedure reference> }*
```

The <package text area> is extended compared to Comprehensive SDL-2010 to include the textual references: <agent type reference>, <package reference> and <procedure reference>; and to allow a <procedure definition>.

```
<agent type reference> ::=
    <system type reference>
  | <block type reference>
  | <process type reference>

<system type reference> ::=
    system type <system type name> [ <formal context parameters> ] referenced <end>

<block type reference> ::=
    <type preamble> block type
  [ <qualifier> ] <block type name> [ <formal context parameters> ] referenced <end>

<process type reference> ::=
    <type preamble> process type
  [ <qualifier> ] <process type name> [ <formal context parameters> ] referenced <end>

<package reference> ::=
    package [ <qualifier> ] <package name> referenced <end>

<diagram in package> ::=
    <package reference area>
  | <entity in agent diagram>
  | <option area>
  | <create line area>
```

The <diagram in package> of Comprehensive SDL-2010 is extended to allow the <create line area> annotation.

Model

A <package reference> in a <package text area> is removed from the <package text area> and transformed to a <package reference area> where the <package symbol> contains the same <package identifier>. The <package reference area> is placed in the <package diagram> containing the <package text area>.

A <system type reference> in a <package text area> is removed from the <package text area> and transformed to a <system type reference area> where the <system type symbol> contains the same <system type name> and <formal context parameters>. The <system type reference area> is placed in the <package diagram> containing the <package text area>.

A <block type reference> in a <package text area> is removed from the <package text area> and transformed to a <block type reference area> where the <block type symbol> contains the same <type preamble>, <qualifier>, <block type name> and <formal context parameters>. The <block type reference area> is placed in the <package diagram> containing the <package text area>.

A <process type reference> in a <package text area> is removed from the <package text area> and transformed to a <process type reference area> where the <process type symbol> contains the same <type preamble>, <qualifier>, <block type name> and <formal context parameters>. The <process type reference area> is placed in the <package diagram> containing the <package text area>.

A <procedure reference> in a <package text area> is removed from the <package text area> and transformed to a <procedure reference area> with the same <procedure reference heading>. The <procedure reference area> is placed in the <package diagram> containing the <package text area>.

7.3 Referenced definition

Comprehensive SDL-2010 is extended to include referenced definitions that are textual. The referenced definitions are extended to include agent and state diagrams.

Concrete grammar

<referenced definition> ::=
 <diagram> | <definition>

<definition> ::=
 <procedure definition>
 | <macro definition>

The <referenced definition> of Basic SDL-2010 is extended to include <definition>. Each alternative of <definition> is a textual definition. For a <macro definition> there is no corresponding reference. Each <procedure definition> shall have a corresponding <procedure reference area> or <procedure reference> in the associated <package diagram> or <system specification>.

NOTE – [ITU-T Z.104] adds operation definition to <definition>.

<diagram> ::=
 <package diagram>
 | <agent diagram>
 | <agent type diagram>
 | <composite state diagram>
 | <composite state type diagram>
 | <procedure diagram>
 | <operation diagram>

The Basic SDL-2010 <diagram> is extended to include <agent diagram> and <composite state diagram> that are shorthand notations.

8 Structural concepts

8.1 Types, instances and gates

8.1.1 Structural type definitions

8.1.1.1 Agent types

Comprehensive SDL-2010 is extended with a shorthand for an agent type diagram with a body that gives the state machine of the agent type without explicitly defining a type for the state machine.

Concrete grammar

```
<agent formal parameters> ::=  
    ( <parameters of sort> {, <parameters of sort>}* )  
    | [ <end> ] fpar <parameters of sort> {, <parameters of sort>}*
```

The Basic SDL-2010 <agent formal parameters> is extended to allow bracketed parameters (as in SDL-2000) as an alternative concrete syntax.

Model

If an <agent type diagram> (a <system type diagram>, <block type diagram> or <process type diagram>) has an <agent structure area> that contains an <agent body area> instead of a <interaction area>, this is shorthand for an agent type that has no contained agents and a state machine with its type defined by the <agent body area>. The transform is described in the next paragraph.

A <composite state type diagram> is derived from the <agent type diagram>. This diagram is given a heading with the same <virtuality>, the key words **state type**, an anonymous name, the same <formal context parameters> and same <agent formal parameters> as the <agent type diagram>. The <agent body area> is copied as the <composite state body area> and any part of a <qualifier> in the body area that refers to the enclosing agent type is changed to refer to the composite state type of the agent type. Each <gate on diagram> of the <agent type diagram> is copied as a <gate on diagram> of the <composite state type diagram> but omitting any <endpoint constraint>. A <composite state type reference area> for the derived <composite state type diagram> is placed in the <agent type diagram>. In the <agent type diagram>, the <agent body area> is replaced by a <typebased state machine> that has as its <typebased composite state>: an anonymous unique name, the <agent formal parameters> list of the agent as <actual parameters>, and as the <composite state type expression> the (anonymous) name of the composite state type with the same <actual context parameters> (if any) as the agent. Each <gate> of the <composite state type diagram> is placed inside the state symbol of the <typebased state machine> and is connected by a channel to the gate of the same name on the <agent type diagram>.

If the <agent type diagram> is defined as a specialization by inheriting another agent type (the supertype), the composite state type of the state machine of the supertype has to be **virtual** and the composite state type name is the same in both the supertype and subtype, because the copied <agent body area> adds to the supertype.

8.1.1.2 System type

Comprehensive SDL-2010 is extended to allow more than one page for a <system type diagram>.

Concrete grammar

```
<system type diagram> ::=  
    <system type page>+
```

8.1.1.3 Block type

Comprehensive SDL-2010 is extended to allow more than one page for a <block type diagram>.

Concrete grammar

```
<block type diagram> ::=
    <block type page>+
```

8.1.1.4 Process type

Comprehensive SDL-2010 is extended to allow more than one page for a <process type diagram>.

Concrete grammar

```
<process type diagram> ::=
    <process type page>+
```

8.1.1.5 Composite state type

Comprehensive SDL-2010 is extended to allow more than one page for a <composite state type diagram>.

Concrete grammar

```
<composite state type diagram> ::=
    <composite state type page>+
    | <state aggregation type page>+
```

NOTE – It is possible to specify a <composite state type diagram> that only consists of transitions associated with an asterisk state, without <start area> and without any substates. These transitions are either terminated by a <dash nextstate> or by a <return area>. These transitions apply when the agent or procedure is in the composite state. The nextstate of such a transition terminated by <dash nextstate> is the composite state; however, the *Exit-procedure-definition* and *Entry-procedure-definition* of the composite state are not called.

8.1.2 Type expression

See [ITU-T Z.101] and [ITU-T Z.102].

8.1.3 Abstract type

See [ITU-T Z.101].

8.1.4 Gates defined by interface gates

The language is extended to include <interface gate definition> as a shorthand and <signal list area> of <gate definition> to be optional.

Concrete grammar

```
<gate on diagram> ::=
    <gate definition> | <interface gate definition>
```

Basic SDL-2010 <gate on diagram> is extended to allow <interface gate definition> as a shorthand.

```
<gate definition> ::=
    {
        { <gate symbol 1> | <inherited gate symbol 1> }
        is associated with
        { <gate> [<signal list area> ] } set
    |
        { <gate symbol 2> | <inherited gate symbol 2> }
        is associated with
        { <gate> [ <signal list area> <signal list area> ] } set
    } [ is connected to <endpoint constraint> ]
```

Comprehensive SDL-2010 <gate definition> is extended to allow <signal list area> to be optional if this is derivable from the communication channels or <endpoint constraint>.

```

<interface gate definition> ::=
    { <gate symbol 1> | <inherited gate symbol 1> }
    is associated with <interface identifier>
    [ is connected to <endpoint constraint> ]

```

Model

If the set of signals carried by a gate is able to be derived from the communication path or endpoint constraint of the gate the <signal list area> is optional in the <gate definition>.

An <interface gate definition> is shorthand for a <gate definition> having the name of the interface as <gate name> and the <interface identifier> as the <gate constraint> or <signal list area>.

8.2 Type references and operation references

Type definitions such as <procedure definition> have type references. The referenced definition defines the properties of the type. The type is fully described in the referenced definition.

Type references are extended compared with Basic SDL-2010 to allow items such as the <type preamble> and <gate property area> to be placed in the type reference as a form of annotation making the items visible in the type reference.

Concrete grammar

```

<agent type reference area> ::=
    {
        <system type reference area>
        | <block type reference area>
        | <process type reference area> }
    { is connected to <gate property area> }*

```

Basic SDL-2010 <agent type reference area> is extended to allow <gate property area> items.

```

<gate property area> ::=
    <gate definition> | <interface gate definition>

```

A <gate property area> is a form of annotation about the gates of the referenced agent. If there is an <agent type reference area> for the agent defined by an <agent type diagram>, each <gate property area> associated with the <agent type reference area> corresponds to the <gate on diagram> with the same <gate> associated with the <agent type diagram>. It is optional whether there is a <gate property area> for a corresponding <gate on diagram>. If there is a <gate property area>, it is not required to include each <signal list item> of the <gate on diagram>. No <gate property area> associated with the <agent type reference area> shall contain <signal list item>s not contained in the corresponding <gate on diagram>s associated with the <agent type diagram>.

```

<system type reference area> ::=
    <system type symbol> contains
    { system <system type name> [ <formal context parameters> ] }

```

Basic SDL-2010 <system type reference area> is extended to allow <formal context parameters>.

```

<block type reference area> ::=
    <block type symbol> contains
    { <type preamble> [<qualifier>] <block type name> [ <formal context parameters> ] }

```

Basic SDL-2010 <block type reference area> is extended to include <type preamble>, <qualifier> and <formal context parameters>.

```

<process type reference area> ::=
    <process type symbol> contains
    { <type preamble> [<qualifier>] <process type name> [ <formal context parameters> ] }

```

Basic SDL-2010 <process type reference area> is extended to include <type preamble>, <qualifier> and <formal context parameters>.

```

<composite state type reference area> ::=
    <composite state type symbol> contains
    { <type preamble>
      [<qualifier>] <composite state type name> [ <formal context parameters> ] }
    { is connected to <gate property area> }*

```

Basic SDL-2010 <composite state type reference area> is extended to include <type preamble>, <qualifier>, <formal context parameters> and connected <gate property area> items.

A <gate property area> is a form of annotation about the gates of the referenced state type. If there is a <composite state type reference area> for a composite state defined by a <composite state type diagram>, each <gate property area> associated with the <composite state type reference area> corresponds to the <gate on diagram> with the same <gate> associated with the <composite state type diagram>. It is optional whether there is a <gate property area> for a corresponding <gate on diagram>. If there is a <gate property area>, it is not required to include each <signal list item> of the <gate on diagram>. No <gate property area> associated with the <composite state type reference area> is allowed to contain a <signal list item> that is not contained in the corresponding <gate on diagram> associated with the <composite state type diagram>.

```

<procedure reference heading> ::=
    <type preamble> [ exported [ as <remote procedure identifier> ] ]
    [<qualifier>] <procedure name> [ <formal context parameters> ]

```

Basic SDL-2010 <procedure reference heading> is extended to include <type preamble>, the **exported** clause, <qualifier> and <formal context parameters>.

If **exported** is given in a <procedure reference heading>, the referenced type shall be an exported procedure and if a <remote procedure identifier> is also given, the procedure shall identify the same remote procedure definition.

```

<procedure reference> ::=
    procedure <procedure reference heading> referenced <end>

```

If there is a non-empty <type preamble> in the type reference, this shall be the same as the <type preamble> of the <referenced definition>.

NOTE – A <type preamble> is optionally empty; therefore adding <type preamble> to the syntax of type references does not invalidate the syntax of Basic SDL-2010.

If in a type reference the <qualifier> of the <name> of the referenced type is omitted or identifies the scope unit directly enclosing the type reference, the type reference and <referenced definition> are logically in the same scope unit: the scope unit containing the reference.

If in a type reference there is <qualifier> for the <identifier> before the <name> of the referenced type and the <qualifier> does not identify the scope unit directly enclosing the type reference, the type reference and <referenced definition> are in different scope units. In this case, it is possible to remove the type reference from model without changing the semantics of the model, and the reference is a form of annotation providing consistent information about the referenced definition. Although the type reference is removed from the model, the additional information given by the optional items such a <type preamble> or <formal context parameters> is still required to match the referenced type. For this matching the context for these items is the context of the referenced type rather than the context of the type reference.

If there is a <formal context parameters> item in the type reference, this shall be the same as the <formal context parameters> of the <referenced definition>.

Multiple type references in the same context that refer to the same entity class and have the same qualifier and the same name are equivalent to one type reference from that context with all attribute property and behaviour property elements of all the references. Such attributes and properties do not have to be present or complete on each reference, but they all have to be consistent with the referenced type and therefore with each other.

Model

A <procedure reference> is removed from the containing text area and transformed to a <procedure reference area> with the same <procedure reference heading> in the scope unit containing the text.

8.3 Context parameters

See [ITU-T Z.102].

8.4 Specialization

Concrete grammar

Specialization is as defined in [ITU-T Z.102], but the following rules apply to shorthand notations introduced in this Recommendation.

- a) Virtual transitions or saves shall not appear in agent (set of instances) definitions, or in composite state definitions.
- b) An input or save with <virtuality> shall not contain <asterisk>.

9 Agents

An additional shorthand diagram is added to Basic SDL-2010 and Comprehensive SDL-2010 to define a set of agents without the need to explicitly define an agent type: the agent diagram. An agent diagram has an implicit anonymous agent type defined by the content of the diagram and used for the agent set.

Other shorthand forms are also added to Basic SDL-2010 and Comprehensive SDL-2010.

Concrete grammar

<agent diagram> ::=

{ <system diagram> | <block diagram> | <process diagram> }
[*is associated with* <package use area>]

<agent instantiation> ::=

[<number of instances>]
<agent additional heading>

NOTE – <agent instantiation> is used in <block diagram> and <process diagram>.

In <agent instantiation>, if <agent formal parameters> are present, <number of instances> shall be present, even if both <initial number> and <maximum number> are omitted. If both the <number of instances> in the <agent instantiation> of an <agent diagram> and the <number of instances> in the <agent reference area> for the <agent diagram> are specified, the two <number of instances> shall be equal lexically.

<agent structure area> ::=

{ {<agent text area>}*
{<entity in agent diagram>}*
{ <interaction area> | <agent body area> } }*set*

The Basic SDL-2010 <agent structure area> is extended to allow an <agent body area> instead of an <interaction area> for the shorthand given in clause 8.1.1.1, where an agent type that has no contained agents and a state machine with its type defined by the <agent body area>.

<agent body area> ::=

{ [<start area>]
{ <state area> <in connector area> }* }*set*

<start area> shall only be omitted in an agent type diagram.

<agent text area> ::=

```
<text symbol>
contains {
    | <valid input signal set>
    | <signal definition list>
    | <variable definition>
    | <data definition>
    | <timer definition>
    | <remote procedure definition>
    | <remote variable definition>
    | <macro definition>
    | <select definition>
    | <signal list definition>
    | <procedure definition>
    | <procedure reference>
    | <imported procedure specification>
    | <imported variable specification>
    | <block reference>
    | <process reference>
    | <block type reference>
    | <process type reference> }* }
```

The Comprehensive SDL-2010 <agent text area> is extended to allow <signal list definition>, <procedure definition>, <imported procedure specification>, <imported variable specification>, textual <block reference>, textual <process reference>, textual <block type reference> and textual <process type reference>. A <signal list definition> is an alternative syntax for an <interface definition>. A <procedure definition> is a textual alternative to a <procedure diagram>.

<imported procedure specification> ::=

```
imported procedure <remote procedure identifier> <end>
{
    [ <end> ] returns <sort> <end>
    | [ <end> ] fpar <formal parameter> {, <formal parameter> }*
    [ <end> returns <sort> ] <end> }
```

An <imported procedure specification> has no SDL-2010 meaning and is treated as a comment, though to be compatible with SDL-2000 and SDL-92 the <remote procedure identifier> should refer to a remote procedure that is consistent with the <formal parameter>s and returned <sort>.

<imported variable specification> ::=

```
imported
    <remote variable identifier> {, <remote variable identifier> }* <sort>
    {, <remote variable identifier> {, <remote variable identifier> }* <sort>}* <end>
```

An <imported variable specification> has no SDL-2010 meaning and is treated as comment, though to be compatible with SDL-2000 and SDL-92 each <remote variable identifier> should refer to a remote variable that is consistent with the <sort>.

<block reference> ::=

```
block <block name> [ <number of instances> ] referenced <end>
```

<process reference> ::=

```
process <process name> [ <number of instances> ] referenced <end>
```

<interaction area> ::=

```
{
    [ <state machine area> ]
    { <agent area> | <create line area> | <channel definition area>}*
}set
```

The Basic SDL-2010 <interaction area> is extended to allow a <create line area> as annotation, and it is allowed to omit the <state machine area> (see *Model* below).

<create line area> ::=

```
<create line symbol>
is attached to {<create line endpoint area> <create line endpoint area> }
```



```

<create line endpoint area> ::=
    <agent area> | <agent type reference area> | <state partition area>

<create line symbol> ::=
    ----->

```

The arrowhead on the <create line symbol> indicates the <agent area> or <agent type reference area> of an agent or agent type upon which a create action is performed. <create line symbol>s are optional and have no formal meaning, but if used, then there shall be a create request for the agent (or agent type) at the arrowhead end of the <create line symbol> in the agent (or agent type or state machine) at the originating end of the <create line symbol>. This rule applies after transformation of <option area>. The create action is possibly inherited and therefore need not be specified directly in the agent or agent type.

```

<state machine area> ::=
    <typebased state machine>
    | <inherited state machine>
    | <state machine reference area>

```

The <state machine area> is extended compared to Comprehensive SDL-2010 to include <state machine reference area>.

```

<state machine reference area> ::=
    <state symbol> contains { <state name> }

```

A <state machine reference area> is shorthand for a <typebased state machine> and a <composite state type reference area> (see *Model* below). The <state name> of a <state machine area> shall be the name of a <composite state diagram> defined in the same scope as the <state machine reference area>.

```

<agent area> ::=
    <typebased agent definition>
    | <inherited agent definition>
    | <agent reference area>

```

The <agent area> of Comprehensive SDL-2010 is extended to include <agent reference area>.

```

<inherited block definition> ::=
    <dashed block symbol> contains
        { <block name> [ <number of instances> ] { <gate>* } set }
        { is connected to <gate property area> }*

```

The <inherited block definition> of Comprehensive SDL-2010 is extended to allow <gate property area> items.

```

<inherited process definition> ::=
    <dashed process symbol> contains { <process name> [ <number of instances> ] { <gate>*
} set }
        { is connected to <gate property area> }*

```

The <inherited process definition> of Comprehensive SDL-2010 is extended to allow <gate property area> items.

Model

An <agent diagram> is shorthand for an <agent type diagram>. The corresponding <agent reference area> to the <agent diagram> in an <agent area> is shorthand for an <agent type reference area> to the <agent type diagram> and a <typebased agent definition> that uses the agent type. The <agent diagram> is transformed into an <agent type diagram> as described in the next paragraph. When an <agent structure area> is used in an <agent diagram>, the <agent diagram> is first replaced by an <agent type diagram>, then any <agent body area> in the <agent structure area> of <agent type diagram> is replaced as in clause 8.1.1.1.

The pages of the <agent diagram> are first combined into a single page. The page of the <agent type diagram> is formed from the page of the <agent diagram> by inserting the keyword **type** after the keyword **system**, **block** or **process** and changing the name to a unique anonymous name. The <agent structure area> is copied from the <agent diagram> to the <agent type diagram>. In the <agent structure area>, any part of a <qualifier> that refers to the agent of the <agent diagram> is changed to refer to the agent type of the <agent type diagram>. For each <external channel identifiers> list outside the frame of the agent diagram (<block diagram> or <process diagram>), there is a connected <gate on diagram> for the agent type diagram with a unique anonymous name, and any <channel definition area> attached to the <external channel identifiers> is attached to the <gate on diagram>. There is also a connected <gate on diagram> for the agent type diagram with a unique anonymous name, for each channel attached to the <agent reference area> for the <agent diagram> that does not appear in <external channel identifiers> of the <agent diagram>.

An <agent diagram> that has specialization (an <agent instantiation> with an <agent additional heading> containing a <specialization>) is shorthand for defining an implicit agent type and one typebased agent of this type.

If the <state machine area> is omitted from an <interaction area>, this is a shorthand for a <state machine area> that is a <typebased state machine> where the <composite state item> is a <typebased composite state> with a <composite state name> that has the same name as the agent name and a <composite state type expression> that is an anonymous name for the type for a minimal state machine: a *State-start-node* followed by a *Stop-node*. The start transition of the minimal state machine is virtual (in a subtype it is allowed to be redefined or finalized).

A <composite state diagram> is shorthand for a <composite state type diagram> and it has a corresponding <state machine reference area> or <state area> or <composite state reference area> referencing the <composite state diagram>. A <state machine reference area> is transformed into a <typebased state machine>, and a <state area> containing a <typebased composite state> that uses the <composite state type diagram>. The <composite state name> of the <composite state item> of the <typebased state machine> is the <state name> of the <state machine reference area>. The <composite state type expression> of the <composite state item> is the unique anonymous name of the <composite state type diagram>. Each <gate> of the <typebased state machine> identifies the anonymously named <gate on diagram> of the <composite state type diagram> that corresponds to a set of channels connected to the <state machine reference area> at the same point. The identified channels all join the <typebased state machine> at the same place close to the <gate> inside the <typebased state machine>.

A <block reference> in an <agent text area> is removed from the <agent text area> and transformed to a <block reference area> where the <block symbol> contains the same <block name> and <number of instances>. The <block reference area> is placed in the <agent structure area> containing the <agent text area>.

A <process reference> in an <agent text area> is removed from the <agent text area> and transformed to a <process reference area> where the <process symbol> contains the same <process name> and <number of instances>. The <process type reference area> is placed in the <agent structure area> containing the <agent text area>.

A <block type reference> in an <agent text area> is removed from the <agent text area> and transformed to a <block type reference area> where the <block type symbol> contains the same <type preamble>, <qualifier>, <block type name> and <formal context parameters>. The <block type reference area> is placed in the <agent structure area> containing the <agent text area>.

A <process type reference> in an <agent text area> is removed from the <agent text area> and transformed to a <process type reference area> where the <process type symbol> contains the same <type preamble>, <qualifier>, <process type name> and <formal context parameters>. The <process type reference area> is placed in the <agent structure area> containing the <agent text area>.

A <procedure reference> is removed from the <agent text area> and transformed to a <procedure reference area> with the same <procedure reference heading>. The <procedure reference area> is placed in the <agent structure area> containing the <agent text area>.

9.1 System

A system diagram is shorthand for a typebased system that is based on an implicit type described by the content of the system diagram. (See the *Model* for <agent diagram> above).

Concrete grammar

```

<system diagram> ::=
    <system page>+

<system page> ::=
    <frame symbol> contains {<system heading> <page number area> <agent structure area> }
    [ is associated with <package use area> ]

<system heading> ::=
    system <system name> <agent additional heading>

```

The <agent additional heading> in a <system diagram> shall not include <agent formal parameters>.

9.2 Block

A block diagram is shorthand for a typebased block that is based on an implicit type described by the content of the block diagram (see the *Model* for <agent diagram> above).

Concrete grammar

```

<block diagram> ::=
    <block page>+

<block page> ::=
    <frame symbol> contains {<block heading> <page number area> <agent structure area> }
    { is associated with <external channel identifiers> }*
    [ is associated with <package use area> ]

<block heading> ::=
    block [<qualifier>] <block name> <agent instantiation>

```

The <external channel identifiers> identify external channels connected to channels in the <block diagram>. It is placed outside the <frame symbol>, close to the endpoint of internal channels at the <frame symbol>.

9.3 Process

A process diagram is shorthand for a typebased process that is based on an implicit type described by the content of the process diagram (see the *Model* for <agent diagram> above).

Concrete grammar

```

<process diagram> ::=
    <process page>+

```

<process page> ::=
 <frame symbol> **contains** { <process heading> <page number area> <agent structure area> }
 { **is associated with** <external channel identifiers> }*
 [**is associated with** <package use area>]

<process heading> ::=
 process [<qualifier>] <process name> <agent instantiation>

The <external channel identifiers> identify external channels connected to channels in the <process diagram>. It is placed outside the <frame symbol>, close to the endpoint of internal channels at the <frame symbol>.

9.4 Procedure

The syntax for procedures is extended compared with Comprehensive SDL-2010 to allow procedures to be defined textually, external procedures and various extensions to the syntax of procedures.

Concrete grammar

<procedure definition> ::=
 <external procedure definition>
 | { <package use clause> }*
 <procedure heading>
 [<end> <entity in procedure>+]
 [<virtuality>] [<comment body>] <left curly bracket>
 <statements> <end>*
 <right curly bracket>

The optional <virtuality> before <left curly bracket> <statements> in <procedure definition> applies to the start transition of the procedure, which in this case is the list of statements.

The definition of **exported** variables is not allowed in a <variable definition> in a <procedure definition>.

A list of <statements> in a <procedure definition> following <virtuality> of **virtual** or **redefined** is a virtual procedure start. Virtual procedure start is further described in clause 8.4.3 of [ITU-T Z.102].

<procedure formal parameters> ::=
 (<formal variable parameters> {, <formal variable parameters> }*)
 | [<end>] **fpar** <formal variable parameters> {, <formal variable parameters> }*

Basic SDL-2010 <procedure formal parameters> syntax is extended to allow bracketed parameters (as in SDL-2000) as an alternative concrete syntax.

<parameter kind> ::=
 [**in/out** | **in** | **out**]

The Basic SDL-2010 <parameter kind> is extended to allow the <parameter kind> to be empty.

<procedure result> ::=
 <result sign> <result aggregation> [<variable name>] <sort>
 | **returns** <result aggregation> [<variable name>] <sort>

The Basic SDL-2010 <procedure result> is extended to allow a <result sign> (as in SDL-2000) as an alternative concrete syntax. Also it is allowed to name the result, in which case it is a variable available for use in the procedure.

<entity in procedure> ::=
 <variable definition>
 | <data definition>
 | <select definition>
 | <macro definition>
 | <procedure definition>
 | <procedure reference>

Comprehensive SDL-2010 <entity in procedure> is extended to allow <procedure definition> and <procedure reference> items in the procedure.

<external procedure definition> ::=
 procedure <procedure name> <procedure signature> **external** <end>

An external procedure shall not be mentioned in a <type expression>, in a <formal context parameter> or in a <procedure constraint>.

Semantics

An external procedure is a procedure whose <procedure body area> is not included in the SDL description (for further information see clause 13 of [ITU-T Z.102]).

Model

A formal parameter with no explicit <parameter kind> has the implicit <parameter kind> **in**.

When a <variable name> is present in <procedure result>, each <return area> within the procedure graph without an <expression> is replaced by a <return area> containing <variable name> as the <expression>.

A <procedure result> with <variable name> is derived syntax for a <variable definition> (that is an <entity in procedure> of the procedure) with <variable name> and <sort> in <variables of sort>. If there is a <variable definition> involving <variable name> in a <variable definition> it shall have the same sort as the result, but no further <variable definition> is added.

A <procedure definition> (other than an <external procedure definition>) is derived syntax for a <procedure diagram> that is a single <procedure page>, having the same <procedure preamble> and <procedure body area> that is a single <procedure start area> with the same <virtuality> as given in the <procedure definition>. The <transition area> of the <procedure start area> consists of a <task area> containing the <statements> of the <procedure definition> followed by an unlabelled <return area>. The <entity in procedure> items of the <procedure definition> are inserted into a <procedure text area> of the <procedure diagram> and allow local items (such as variables) of the procedure to be defined. This transformation takes place after handling any <select definition> or <macro definition> in the <procedure definition>.

A <procedure reference> as an <entity in procedure> of a <procedure text area> is removed from the <procedure text area> and transformed to a <procedure reference area> with the same <procedure reference heading> in the procedure containing the <procedure text area>. The <procedure reference> transformations take place after transformation of each <procedure definition> to a <procedure diagram>.

9.5 Agent and composite state reference

Basic SDL-2010 is extended to include references to agent diagrams and composite state diagrams of a state aggregation. Both the references and the diagram are shorthand notations.

Concrete grammar

<agent reference area> ::=
 { <block reference area>
 | <process reference area> }

<block reference area> ::=
 <block symbol> **contains**
 { { <block name> [<number of instances>] } }

<process reference area> ::=
 <process symbol> **contains**
 { { <process name> [<number of instances>] } }

<composite state reference area> ::=
 <state symbol> *contains* { <state name> }

NOTE – A <composite state reference area> is used only in a <state partition area> of a <state aggregation body area> to reference a <composite state diagram> for the partition. A <composite state diagram> that is used by a composite state application has a <composite state item> with a <composite state name> that identifies the <composite state diagram>.

Model

The <agent diagram> referenced by an <agent reference area> is a shorthand that is transformed into an <agent type diagram> (see the transformation of <agent diagram> under *Model* in clause 9). An <agent reference area> is transformed to a <typebased agent definition> that uses the <agent type diagram> and an <agent type reference area> for the <agent type diagram>. The <typebased agent definition> has the same symbol (<block symbol> or <process symbol>) as the <agent reference area>, which contains a typebased heading (<typebased block heading> or <typebased process heading>) and a <gate> for each <external channel identifiers> list on the <agent diagram>. Each <gate> of the <typebased agent definition> is the anonymous unique name of the <gate on diagram> of the <agent type diagram> formed from the <external channel identifiers> on the <agent diagram> or a channel connected to <agent reference area> for the <agent diagram> that does not appear in the <external channel identifiers>. The <gate> is placed so that the set of channels (for the <external channel identifiers>) or channel (not in <external channel identifiers>) is associated with this <gate>. In the <typebased agent definition>, the <block name> or <process name> is the same as the <block name> or <process name> of the <agent reference area>. The <number of instances> of the <typebased agent definition> is the same as the <number of instances> present in the <agent reference area> or <agent diagram> (if both are given or one is omitted), or is empty if both are omitted. In the <typebased agent definition>, the <block type expression> or <process type expression> is the anonymous unique name of the <agent type diagram>.

A <composite state diagram> referenced by <composite state reference area> as a <state partition area> is a shorthand that is transformed into a <composite state type diagram> (see the transformation of <composite state diagram> in *Model* under clause 11.11). A <composite state reference area> is transformed to a <typebased state partition definition> that uses the <composite state type diagram>, and a <composite state type reference area> for the <composite state type diagram>. The <state symbol> of the <typebased state partition definition> contains the same <composite state name> and <actual parameters> as the <composite state reference area>, and the <composite state type expression> is the anonymous unique name of the <composite state type diagram>. The <typebased state partition definition> replaces the <composite state reference area> as the <state partition area> so that each <state partition connection area> attached to the <state partition area> is unchanged.

10 Communication

10.1 Channel

Concrete grammar

```
<channel definition area> ::=
    <channel symbol 1>
    is associated with
        { [<channel name>] [ <signal list area> ] }set
    is attached to {
        {
            <agent area> | <state machine area> | <gate on diagram>
            | <external channel identifiers> }
        {
            <agent area> | <state machine area> | <gate on diagram>
            | <external channel identifiers> } }set
    |
    <channel symbol 2>
    is associated with
        { <channel name> [ <signal list area> ] [ <signal list area> ] }set
    is attached to {
        {
            <agent area> | <state machine area> | <gate on diagram>
            | <external channel identifiers> }
        {
            <agent area> | <state machine area> | <gate on diagram>
            | <external channel identifiers> } }set
```

Basic SDL-2010 <channel definition area> is extended to allow the <channel name> and each <signal list area> to be optional. Basic SDL-2010 <channel definition area> is also extended to allow the <channel definition area> to be attached to <external channel identifiers> (rather than a <gate on diagram>) in the case of an <agent diagram> or <composite state diagram> for the state machine of an agent.

For the end of the <channel symbol 1> or <channel symbol 2> that is attached directly to an <agent area> or <state machine area> where the agent or state machine contains the <channel identifier>s for the channel in <external channel identifiers>, the channel is attached to the implicit gate introduced by the <external channel identifiers>. Otherwise, in the case of no <external channel identifiers> match, there is an implicit gate on the agent or state attached to the <channel definition area>. This gate obtains the <signal list> of the respective <channel definition area> as its corresponding gate constraint. The channel is attached to that gate. This <gate> represents either the *Destination-gate* or *Originating-gate*, with the other gate determined by the other end of the channel.

If any associated <signal list area> is omitted from a <channel definition area>, the corresponding set of signals shall be derivable. Derivation is possible if at least one attached <agent area>, <state machine area> or <gate on diagram> has a defined set of signals for the gate to which the channel is connected in the direction for the <signal list area>. The set of signals is defined if the channel is connected to an <agent area> or <state machine area> and the set is defined for the gate to which the channel is connected, or if the set is defined for all internal channels or agents connected to this gate. The set of signals is defined if the channel is connected to a <gate on diagram>. The set is defined for a gate connected to <external channel identifiers> if for each external channel either no <signal list area> is omitted or the set for that external channel is derivable.

Model

If the <channel name> is omitted from a <channel definition area>, the channel is implicitly and uniquely named.

A channel with both endpoints being gates of one <typebased agent definition> represents individual channels from each of the agents in this set to all agents in the set, including the originating agent. Any resulting bidirectional channel connecting an agent in the set to the agent itself is split into two unidirectional channels.

If an associated <signal list area> is omitted from a <channel definition area>, the corresponding *In-signal-identifier-set* or *Out-signal-identifier-set* for the *Destination-gate* or *Originating-gate* of the channel is derived from the channel connection, if necessary.

If an agent or agent type contains explicit or implicit gates not attached by explicit channels, implicit channels are derived according to the following three transforms, which are applied after the transform for typebased creation is applied.

Transform 1:

Insertion of channels between instance sets inside the agent or agent type and between the instance sets and the agent state machine.

Transform 2:

Insertion of channels from a gate on the agent or agent type to gates on instance sets inside the agent or agent type and to gates on the agent state machine.

Transform 3:

Insertion of channels from gates on instance sets inside the agent or agent type and from gates on the agent state machine to gates on the agent or agent type.

In the transforms, one signal list element (interface, signal, remote procedure or remote variable) matches another signal list element if:

- a) both denote the same interface, signal, remote procedure or remote variable; or
- b) the first denotes a signal or remote procedure or remote variable, and the second denotes an interface and the interface includes the signal or remote procedure or remote variable; or
- c) both denote interfaces, and the second signal list element inherits the first signal list element.

Transform 1: Insertion of implicit channels between entities inside one agent or agent type

- a) If an element of the outgoing signal list associated with a gate of an instance in an agent (or agent type) matches an element of an incoming signal list associated with a gate of another instance in the same agent (or agent type respectively); and
- b) if neither of these gates has an explicit channel attached to it,

then

- a) if no implicit channel exists between the two gates, a unidirectional implicit channel is created from the gate where the element is outgoing to the gate where the element is incoming, and this channel is non-delaying if it is within a process (or process type) and otherwise it is delaying; and
- b) the element is added to the signal list of the implied channel.

Transform 2: Insertion of implicit channels from the gates on an agent or agent type

- a) If an element of the incoming signal list associated with a gate outside an agent (or agent type) matches an element of an incoming signal list associated with a gate of an instance in the agent (or agent type respectively); and
- b) if there is no explicit channel inside the agent (or agent type respectively) attached to the gate outside the agent (or agent type respectively) and no explicit channel attached to the gate of the instance inside the agent (or agent type respectively),

then

- a) if no implicit channel exists between the two gates, a unidirectional implicit channel is created from the gate outside the agent (or agent type respectively) to the gate of the instance inside the agent (or agent type respectively), and this channel is non-delaying if it is within a process (or process type) and otherwise it is delaying; and

- b) the element is added to the signal list of the implied channel.

Transform 3: Insertion of implicit channels from the gates on instances

The following is applied for insertion of implicit channels from the gates on instance sets within the agent or agent type to the gates on the agent or agent type:

- a) If an element of the outgoing signal list associated with a gate outside an agent (or agent type) matches an element of an outgoing signal list associated with a gate of an instance in the agent (or agent type respectively); and
- b) if there is no explicit channel attached to the gate outside the agent (or agent type respectively) and no explicit channel connected to the gate of the instance inside the agent (or agent type respectively),

then

- a) if no implicit channel exists between the two gates in the direction to the gate outside the agent (or agent type respectively), a unidirectional implicit channel is created from the gate of the instance inside the agent (or agent type respectively) to the gate outside the agent (or agent type respectively), and this channel is non-delaying if it is within a process (or process type) and otherwise it is delaying; and
- b) the element is added to the signal list of the implied channel.

10.2 Connection

A connection is the point where a channel inside a frame symbol for an agent diagram is connected to names of one or more channels outside a frame symbol.

Concrete grammar

<external channel identifiers> ::=
 <channel identifier> { , <channel identifier> }

For a connection, a <channel symbol 1> or <channel symbol 2> in a <channel definition area> is attached to <external channel identifiers> outside the enclosing <frame symbol> of a page: <block page> of a <block diagram> or <process page> of a <process diagram> or <composite state graph page> of the state machine of an agent. The <frame symbol> is associated with the <external channel identifiers>.

The <channel identifier>s in the <external channel identifiers> shall denote channels of the agent or agent type enclosing the block or process that is associated with the <external channel identifiers>. Each channel attached to an <agent area> that is an <agent reference area> shall be mentioned in at least one <external channel identifiers> of the <agent diagram> referenced by the <agent reference area>: in the <external channel identifiers> of a <block page> of a <block diagram> or in the <external channel identifiers> of a <process page> of a <process diagram>.

Model

A connection is part of the shorthand for an agent diagram or the composite state diagram of a state machine. A connection for an agent diagram is transformed to a <gate on diagram> of the <agent type diagram> derived from the agent diagram, and the corresponding <gate> within the <typebased agent definition> that replaces the <agent reference area>. A connection for the composite state diagram of a state machine is transformed to a <gate on diagram> of the <composite state type diagram> derived from the composite state diagram, and the corresponding <gate> within the <typebased state machine> that replaces the <state machine reference area>.

The name of each gate is a unique and unambiguous derived name. In the surrounding scope unit, each <channel definition area> that is identified by a <channel identifier> of <external channel identifiers> is attached to the <typebased agent definition> that contains the <gate>. Inside an <agent type diagram> for an agent type formed from the <agent diagram>, each <channel definition

area> that was attached to the <external channel identifiers> in the <agent diagram> is attached to the <gate on diagram> for the gate. Inside a <composite state type diagram> for a composite state type formed from the <composite state diagram>, each <channel definition area> that was attached to the <external channel identifiers> in the <agent diagram> is attached to the <gate on diagram> for the gate.

10.3 Signal

Signal is as defined in [ITU-T Z.102].

10.4 Signal list area

Concrete grammar

```

<signal list item> ::=
    | <signal identifier>
    | <timer identifier>
    | ( <interface identifier> )
    | [ procedure ] <remote procedure identifier>
    | [ remote ] <remote variable identifier>
    | [ interface ] <interface identifier>

```

The Comprehensive SDL-2010 <signal list item> is extended to allow the keywords **procedure**, **remote**, and **interface** to be omitted. A <signal list item> which is an <identifier> denotes a <signal identifier> or <timer identifier> or <interface identifier>, or else a <remote procedure identifier> if this is possible according to the visibility rules, or else a <remote variable identifier>. To force a <signal list item> to denote a <remote procedure identifier>, an <interface identifier>, or <remote variable identifier> the keyword **procedure**, **interface** or **remote**, respectively, is used.

In Comprehensive SDL-2010 a <signal list item> is allowed to contain an <interface identifier> when it occurs in a <input list> or <priority input list> is a shorthand for the set of signals of the interface.

11 Behaviour

11.1 Start

Start is as defined in [ITU-T Z.102].

11.2 State

Concrete grammar

Comprehensive SDL-2010 <state area> is extended to allow the same <state name> to appear in more than one <state area> of a body, as described in *Model* below where the <state area> items are transformed to have only one state name in each <state area>.

```

<state list> ::=
    { <basic state name> | <composite state item> }
    | { , { <basic state name> | <composite state item> } } *
    | <asterisk state list>

```

Basic SDL-2010 <state list> is extended to allow multiple items, and an <asterisk state list> as an alternative.

```

<asterisk state list> ::=
    <asterisk> [ ( <state name> { , <state name> } * ) ]

```

The <state name>s in an <asterisk state list> shall be distinct and shall be contained in other <state list>s in the enclosing body or in the body of a supertype.

<composite state item> ::=
 <composite state name> <nextstate parameters>
 |
 <typebased composite state>

Basic SDL-2010 <composite state item> is extended to allow a <composite state name> (with <nextstate parameters>) that references a <composite state diagram> as an alternative to a <typebased composite state>. The alternative <composite state name> with <nextstate parameters> uses a <composite state name> that references a <composite state diagram>, which is a shorthand that is transformed to a <typebased composite state> that defines the name and a <composite state type reference area> that references a <composite state type diagram> transformed from the <composite state diagram>.

<typebased composite state> ::=
 <composite state name> <nextstate parameters>
 <colon> <composite state type expression>

Basic SDL-2010 <typebased composite state> is extended to allow <nextstate parameters>.

A <composite state item> of a <state list> shall only contain non-empty <nextstate parameters> if it is in a <state area> that coincides with a <nextstate area>. A <state area> that is a <terminator area> of <transition area> is a shorthand for a <state area> that coincides with a <nextstate area> and shall only contain one state name (a <basic state name> or a <composite state name>).

Model

Any <state area> as the <terminator area> of a <transition area> is transformed to a <nextstate area> as the <terminator area> of the <transition area> and a <state area> that is not a <terminator area> of a <transition area> (see *Model* in clause 11.12.1). This is done before combining <state area> items as described below.

When the <state list> of a <state area> contains more than one <state name> item, a copy of that <state area> is created for each such <state name>. Then the <state area> is replaced by these copies.

When several <state area> items contain the same <state name>, these <state area>s are combined into one <state area> having that <state name>. If any <composite state item> of combined states is a <typebased composite state>, the <state list> of the combined <state area> is the <typebased composite state>.

A <composite state diagram> referenced by <composite state item> of a <state area> is a shorthand that is transformed into a <composite state type diagram> (see the transformation of <composite state diagram> in *Model* under clause 11.11). A <state area> that contains a <composite state item> that references a <composite state diagram> is transformed to a <state area> that contains a <typebased composite state> that uses the <composite state type diagram>, and a <composite state type reference area> for the <composite state type diagram>. The <typebased composite state> has the same <composite state name> as the <composite state item>, and the <composite state type expression> is the anonymous unique name of the <composite state type diagram>. The <state area> is associated with the same <input association area>, <priority input area>, <continuous signal association area>, <spontaneous transition area>, <save association area> and <connect association area> items.

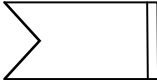
A <state area> with an <asterisk state list> is transformed to a set of <state area>s, one for each <state name> of the body in question, except for those <state name>s and <composite state name>s contained in the <asterisk state list>.

11.3 Input

Concrete grammar

<input symbol> ::=
 <plain input symbol>
 | <internal input symbol>

Basic SDL-2010 <input symbol> is extended to allow <internal input symbol> as an alternative to <plain input symbol>.

<internal input symbol> ::=


NOTE – There is no difference in meaning between a <plain input symbol> and an <internal input symbol>, but the difference enables annotation, for example, to distinguish signals received from within an agent from signals received from outside the agent.

<input list> ::=
 <stimulus> [<priority clause>] { , <stimulus> [<priority clause>] }*
 | <asterisk input list> [<priority clause>]

Basic SDL-2010 <input list> is extended to allow more than one <stimulus> in an <input list>, an <asterisk input list> as an alternative, and <priority clause> to specify that the input is a priority input. Compared with Basic SDL-2010, a <stimulus> is allowed to represent an interface. An <interface identifier> in a stimulus is a model, as described below.

When the <input list> contains one <stimulus> without a <priority clause>, then the <input area> represents an *Input-node*; otherwise the <input list> is shorthand that is transformed as in *Model* below so that each resulting <input list> contains one <stimulus> as in Basic SDL-2010.

<asterisk input list> ::=
 <asterisk>

A <state area> shall contain at most one <asterisk input list>. A <state area> shall not contain both <asterisk input list> and <asterisk save list>.

<via path> ::=
 via { <channel identifier> | <gate identifier> }

Basic SDL-2010 <via path> is extended to allow <channel identifier> as a shorthand for the gate. The <channel identifier> for a <via path> of a <stimulus> or <save area> shall identify a channel connected to the enclosing state machine. A <channel identifier> for a <via path> of <communication constraints> shall identify a channel reachable from the enclosing state machine.

Model

A <stimulus> whose <signal list item> is an <interface identifier> is derived syntax for a list of <stimulus> items, each without parameters that replaces the <interface identifier> in the enclosing <input list> or <priority input list>. In the list, there is a one-to-one correspondence between the <stimulus> items and the set of signals for the interface.

An <asterisk input list> in an <input list> is transformed to a <stimulus> list, one for each member of the complete valid input signal set of the enclosing agent type, except for any <signal identifier> of an implicit input signal (for a remote procedure, a remote variable, a priority input, a continuous signal or an enabling condition) or any <signal identifier> contained in any other <input list> and <save list> of the <state area>. If the <asterisk input list> is followed by a <priority clause>, each <stimulus> is followed by this <priority clause>.

NOTE – If the <asterisk input list> is applied to a substate within a composite state graph, the substate has an input or save for every receivable signal, therefore in this substate it is not possible to trigger a transition of an input on a composite state application that uses the composite state.

When the <stimulus> list of an <input area> of an <input association area> contains more than one <stimulus>, a copy of the <input association area> is created for each such <stimulus>. Then the <input association area> is replaced by these copies.

In an <input area> with one <stimulus>, if the <stimulus> is followed by a <priority clause>, the input symbol is replaced by a <priority input symbol> containing the <stimulus> followed by the <priority clause> so that the <input association area> becomes a <priority input association area>.

When one or more of the stimulus variable items of a <stimulus> is an <extended variable> (an <indexed variable> or <field variable>), then each <extended variable> is replaced by a unique, new, anonymous implicitly declared <variable identifier> of the same sort as the original <extended variable>. Directly following the input area (<input area> or <priority input area>), a <task area> is inserted which in its <task body> contains an <assignment statement> for each of the <extended variable> items, assigning the result of the corresponding new variable to the <extended variable>. The results are assigned in the order from left to right of the <extended variable> list. This <task area> becomes the first <action area> in the <transition area> of the input area.

A <via path> of a <stimulus> with a <channel identifier> is transformed to a <via path> with a (possibly anonymous) <gate identifier> for the gate where the channel connects (directly or indirectly) to the enclosing state machine.

A <channel identifier> as a <via path> of <communication constraints> is transformed to the (possibly anonymous) <gate identifier> for the gate of the identified channel nearest to the agent containing the <communication constraints>.

11.4 Priority input

Concrete grammar

```
<priority input list> ::=
    <priority stimulus> {, <priority stimulus>}*
    | <asterisk input list> [ <priority clause> ]
```

Comprehensive SDL-2010 <priority input list> is extended to allow more than one <priority stimulus> and an <asterisk input list>.

```
<priority stimulus> ::=
    <stimulus> [ <priority clause> ]
```

Comprehensive SDL-2010 <priority stimulus> is extended to allow the <priority clause> to be omitted.

```
<priority clause> ::=
    priority [ <priority name> ]
```

Comprehensive SDL-2010 <priority clause> is extended to allow the <priority name> to be omitted.

Model

An <asterisk input list> in a <priority input list> is transformed to a <priority stimulus> list in the same way as an <asterisk input list> in an <input list> is transformed to a <stimulus> list (see *Model* in clause 11.3).

When the <priority stimulus> list of a <priority input area> of a <priority input association area> contains more than one <priority stimulus>, a copy of the <input association area> is created for each such <priority stimulus>. Then the <priority input association area> is replaced by these copies.

A <priority stimulus> without a <priority clause> is transformed into a <priority stimulus> with a <priority clause> that is **priority** without a <priority name>.

A <priority clause> without a <priority name> is transformed into a <priority clause> that is **priority n**, where n is one greater than the highest explicit <priority name> for a <priority stimulus> of the same state. The same value is used for all such transformations of the state, so that each priority stimulus without an explicit <priority name> has the same priority.

A <stimulus> in a <priority input area> with an <indexed variable> item or <field variable> item is transformed in the same way as for an <input area>.

11.5 Continuous signal

Continuous signal is as defined in [ITU-T Z.102].

11.6 Enabling condition

Enabling condition is as defined in [ITU-T Z.102].

11.7 Save

Concrete grammar

```
<save list> ::=
    <save item> {, <save item> }*
    | <asterisk save list>
```

Basic SDL-2010 <save list> is extended to allow more than one <save item> and an <asterisk save list>.

```
<asterisk save list> ::=
    <asterisk>
```

A <state area> is allowed to contain at most one <asterisk save list>. A <state area> shall not contain both <asterisk input list> and <asterisk save list>.

Model

An <asterisk save list> is transformed to a list of <stimulus> items containing the complete valid input signal set of the enclosing <agent diagram>, except for any <signal identifier> of an implicit input signal (for a remote procedure, a remote variable, a priority input, a continuous signal or an enabling condition) or any <signal identifier> contained in any other <input list> and <save list> of the <state area>.

NOTE – If the <asterisk save list> is applied to a substate within a composite state graph, the substate has an input or save for every receivable signal, therefore in this substate it is not possible to trigger a transition of an input on a composite state application that uses the composite state.

11.8 Implicit transition

Any signal not handled by an explicit input or save is consumed by an implicit transition (a transition of an implicit <input association area> – see below) without a change of state.

Model

The models are applied for any <state area> as a <terminator area>, any asterisk states, any <state area> with multiple state names, and combining every <state area> with the same name, before the model for implicit transition is applied. After applying these models, the set of signals are identified that are contained in the <input list>s, <priority input list>s and the <save list>s of the <state area> (explicitly or via <asterisk input list> or <asterisk save list>), and in the following this is called the local signal set. If the local signal set is the same as the valid input signal set (see below), there are no implicit <input association area> items; otherwise these items are derived as described below. Each implicit <input association area> has a <transition area> that only contains a <nextstate area> leading back to the <state area>.

The valid input signal set for determining implicit transitions is the complete valid input signal set of the agent type or agent type enclosing the <state area>, or if the <state area> is within a state partition, the complete valid input signal set of the state partition.

If the <state area> is in an <agent body area> for an <agent diagram>, or is in a <composite state body area> for a <composite state diagram> referenced by a <state machine reference area>, the <state area> is for a state of the state machine of an agent instance, and there is an implicit <input association area> for each <signal identifier> contained in the valid input signal set that is not in the local signal set.

If the <state area> is in an <agent body area> for an <agent type diagram>, the <state area> is for a state of the state machine of an agent type and there is an implicit <input association area> for each <signal identifier> contained in the valid input signal set that is not in the local signal set.

If the <state area> is in a <procedure body area>, there is an implicit <input association area> for each <signal identifier> contained in the valid input signal set that is not in the local signal set.

A signal is in the input of a composite state if it is in the local signal set for the <state area> of the composite state, or in the local signal input set of any enclosing composite state.

If the <state area> is in a <composite state body area> for a <composite state diagram> referenced by a <composite state name> of a <composite state item>, the <state area> is for a state within the composite state. In this case there is an implicit <input association area> for each <signal identifier> contained in the valid input signal set that is not in the local signal set of the <state area> and not in any input of the composite state.

If the <state area> is in a <composite state body area> for a <composite state type diagram> identified by a <composite state type expression> of a <composite state item>, the <state area> is for a state within the composite state type. In this case, there is a different derivation of implicit transitions for each composite state application that uses the <composite state type diagram> and has a different local signal input set. Consequently, the derived <composite state body area> is (probably) different for each composite state application. Where there is at least one signal for a composite state application (a <state area> with a <composite state item> that is a <typebased composite state>), the <typebased composite state> is transformed to reference a different implied <composite state type diagram> that is a copy of the original <composite state type diagram> but with the implicit transitions expanded. In this <composite state type diagram>, for each <state area> there is an implicit <input association area> for each <signal identifier> contained in the complete valid input signal set for the agent or agent type that is not in the local signal set of the <state area> and not in any input of the composite state application.

NOTE – If composite state (type) or procedure is defined outside a state partition and is used within a state partition, the model produces implicit transitions for signals that are not allowed in the partition; therefore these constructs are not valid.

11.9 Spontaneous transition

Spontaneous transition is as defined in [ITU-T Z.102].

11.10 Label (connector name)

Concrete grammar

In addition to Comprehensive SDL-2010 the term body includes <agent body area>, and therefore all the <connector name>s defined in such a body shall be distinct.

11.11 State machine and composite state

11.11.1 Composite state graph

An additional shorthand diagram is added to Basic SDL-2010 and Comprehensive SDL-2010 to define a composite state without the need to explicitly define a composite state type: the composite state diagram. A composite state diagram has an implicit anonymous composite state type defined by the content of the diagram and used for the composite state.

Concrete grammar

```
<composite state diagram> ::=
    <composite state graph page>+ | <state aggregation page>+
<composite state graph page> ::=
    <frame symbol> contains {
        { <composite state heading> <composite state structure area> }
        { is connected to <state connection point area> }*
        [ is associated with <package use area> ]
```

The <package use area> shall be placed on the top of the <frame symbol>.

```
<composite state heading> ::=
    <virtuality> state [<qualifier>] <state name>
    [<agent formal parameters>] [<specialization>]
```

```
<composite state text area> ::=
    <text symbol> contains
    {
        <valid input signal set>
        | <variable definition>
        | <data definition>
        | <procedure definition>
        | <procedure reference>
        | <select definition>
        | <macro definition> }*
```

Comprehensive SDL-2010 <composite state text area> is extended to allow a <procedure definition> and <procedure reference>.

Model

A <composite state diagram> is shorthand for a <composite state type diagram> transformed as described in the next paragraph. The corresponding reference references the <composite state diagram> by its <state name> and is:

- a) a <state area> with a <state list> item that is a <composite state item>; or
- b) a <composite state reference area> in a <state partition area>; or
- c) a <state machine reference area>.

A corresponding <state area> is shorthand for a <composite state type reference area> to the <composite state type diagram>, and a <state area> containing a <typebased composite state> that uses the <composite state type diagram>. A corresponding <composite state reference area> is shorthand for a <composite state type reference area> to the <composite state type diagram> and a <typebased state partition definition> that uses the <composite state type diagram>. A corresponding <state machine reference area> is shorthand for a <composite state type reference area> to the <composite state type diagram>, and a <state area> containing a <typebased composite state> that uses the <composite state type diagram>.

The pages of the <composite state diagram> are first combined into a single page. If the composite state is not an aggregation, the <composite state type diagram> is formed from the <composite state diagram> by changing the <composite state graph page> into a <composite state type page> by inserting the keyword **type** after the keyword **state** in the heading and changing the name to an

anonymous name. The <composite state type heading> therefore has the <virtuality> given in the <composite state heading> of the <composite state diagram>. The anonymous name is generated in the same way for redefinitions, so that the names match for the virtual type and redefinitions. The <composite state structure area> is copied from the <composite state diagram> to the <composite state type diagram>. In the <composite state body area>, any part of a <qualifier> that refers to the composite state of the <composite state diagram> is changed to refer to the composite state type of the <composite state type diagram>. If the <composite state diagram> is for a state machine, there is also a connected <gate on diagram> for the composite state type diagram with a unique anonymous name, for each channel attached to the <state machine reference area> for the <composite state diagram>.

The model for a composite state diagram that is an aggregation is described in clause 11.11.2.

If a composite state consists of no <state area>s with <state name>s but only a <state area> with an <asterisk>, the asterisk state is transformed into a <state area> with an anonymous <state name> and a <start area> leading to this <state area>.

A <composite state diagram> that has a specialization (with a <composite state heading> that contains a <specialization>) is shorthand for defining an implicit composite state type and one typebased composite state of this type.

A <procedure reference> is removed from the <composite state text area> transformed to a <procedure reference area> with the same <procedure reference heading> in the composite state or composite state type containing the <composite state text area>.

11.11.2 State aggregation

Comprehensive SDL-2010 is extended to allow a composite state diagram that is an aggregation.

Concrete grammar

```

<state aggregation page> ::=
    <frame symbol> contains {
        <state aggregation heading>
        <aggregation structure area> }
    is connected to {<state connection point area>* } set
    [ is associated with <package use area> ]

<state aggregation heading> ::=
    <virtuality> state aggregation [<qualifier>] <state name>
    [<agent formal parameters>][<specialization>]

<state partition area> ::=
    <composite state reference area>
    | <typebased state partition definition>
    | <inherited state partition definition>

```

Comprehensive SDL-2010 <state partition area> is extended to allow <composite state reference area> for reference to a <composite state diagram>.

Model

If an entry point of the state aggregation is not connected to any entry point of a state partition, an implicit connection to the unlabelled entry is added. Likewise, if an exit point of a partition is not connected to any exit point of the state aggregation, a connection to the unlabelled exit is added.

The pages of the <composite state diagram> for an aggregation are first combined into a single page. The <composite state type diagram> for the aggregation is formed from the <composite state diagram> by changing the <state aggregation page> into a <state aggregation type page> by inserting the keyword **type** after the keyword **aggregation** in the heading and changing the name to an anonymous name. The <state aggregation type heading> therefore has the <virtuality> given in the <state aggregation heading> of the <composite state diagram>. The anonymous name is

generated in the same way for redefinitions, so that the names match for the virtual type and redefinitions. The <aggregation structure area> is copied from the <composite state diagram> to the <composite state type diagram>. In the <composite state structure area>, any part of a <qualifier> that refers to the composite state of the <composite state diagram> is changed to refer to the composite state type of the <composite state type diagram>.

11.11.3 State connection point

State connection point is as defined in [ITU-T Z.102].

11.11.4 Connect

```
<connect list> ::=
    <state exit point list>
    | <asterisk connect list>
```

Comprehensive SDL-2010 <state partition area> is extended to include an <asterisk connect list>, which is a shorthand.

```
<state exit point list> ::=
    { <state exit point> | default } { , { <state exit point> | default } }*
```

Comprehensive SDL-2010 <state exit point list> is extended to allow more than one state exit point in the list and the keyword **default**. The keyword **default** represents a *Connect-node* without a *State-exit-point-name* (and is the same as omitting the <connect list>) and corresponds to an unlabelled <return area>. It is used when a <state exit point list> should contain the unnamed exit as well as one or more named exits.

NOTE – It is permitted to have the same state exit point more than once in the <state exit point list>, but this has the same meaning as one appearance.

```
<asterisk connect list> ::=
    <asterisk> [ ( <state exit point list> ) ]
```

Model

When the <connect list> of a certain <connect association area> contains more than one <state exit point name>, a copy of the <connect association area> is created for each such <state exit point name>. Then the <connect association area> is replaced by these copies.

A <connect list> that contains an <asterisk connect list> is transformed into a list of <state exit point>s, one for each <state exit point> of the <composite state diagram> in question (including the unlabelled <return area>) except those mentioned in parentheses after the <asterisk>. The list of <state exit point>s is then transformed as described above.

11.12 Transition

11.12.1 Transition body

Concrete grammar

```
<terminator area> ::=
    <state area>
    | <merge area>
    | <nextstate area>
    | <decision area>
    | <stop symbol>
    | <out connector area>
    | <return area>
    | <transition option area>
```

Comprehensive SDL-2010 <terminator area> is extended to include <state area> and <merge area>.

A <state area> as a <terminator area> is a shorthand combination where a <nextstate area> coincides with a <state area>, and in this case the <state list> shall contain only one item (one <basic state name> or one <composite state item>). A <state area> as a <terminator area> shall not have an <asterisk state list>.

A <merge area> is a shorthand for two transitions each ending in an <out connector area> that contains the same <connector name> as the other transition.

Model

When the <terminator area> of a <transition area> is a <state area>, this is transformed: the <state area> remains (but not as the <terminator area> and with the <state symbol> content updated), and a <nextstate area> is introduced as the <terminator area> of the <transition area> instead of the <state area>. If the original <state area> has a <basic state name> introduced, <nextstate area> contains the <basic state name>. If the original <state area> has a <composite state name>, the <nextstate area> contains the <composite state name> and the <nextstate parameters>. The modified <state area> has the same contents as the original <state area> except the <nextstate parameters>, which are removed.

11.12.2 Transition terminator

11.12.2.1 Nextstate

Nextstate is as defined in [ITU-T Z.102].

11.12.2.2 Join

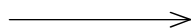
The Basic SDL-2010 join is extended to include a merge area.

Concrete grammar

<merge area> ::=

<merge symbol> *is attached to*
 { <transition string area> | <terminator area> }

<merge symbol> ::=



The <merge symbol> has the same form as a <flow line symbol with arrowhead>, but is distinguished from a <flow line symbol> by the end with the arrowhead joining a <flow line symbol> leading to the attached <transition string area> or <terminator area> that is not a <merge area>, or by joining the <merge symbol> of an attached <terminator area> that is a <merge area>.

Model

If a <merge area> is included in a <transition area>, it is equivalent to specifying an <out connector area> in the <transition area> which contains a unique <connector name> and attaching an <in connector area>, with the same <connector name> to the <flow line symbol> in the <merge area>.

A <merge area> is transformed as follows. If it joins a <flow line symbol>, this <flow line symbol> is split into two <flow line symbol>s by adding an <in connector area> with a unique anonymous <connector name> that has a <flow line symbol> leading to the original right hand side of the corresponding *is followed by*. The original left hand side of the *is followed by* has a <flow line symbol> leading to an <out connector area> with the same unique anonymous <connector name>. The <merge symbol> is replaced by a <flow line symbol> leading to an <out connector area> with the same unique anonymous <connector name>.

If the <merge symbol> joins the <merge symbol> of an attached <terminator area> that is another <merge area>, this other <merge area> is transformed first, so that the <merge symbol> then joins a <flow line symbol> leading to a <terminator area> that is an <out connector area>.

11.12.2.3 Stop

Stop is as defined in [ITU-T Z.101] and [ITU-T Z.102].

11.12.2.4 Return

The Basic SDL-2010 <procedure result> is extended to allow the result to be a named variable, and it is allowed to omit the <expression> that is a <return body> in a <return area>.

Concrete grammar

If the <procedure result> has a <variable name>, the <expression> that is a <return body> in a <return area> is optional.

Model

If the <expression> is omitted in an operator or method with an <operation result> or a value returning procedure with a named <procedure result>, the procedure result variable is used as the <expression>.

11.13 Action

11.13.1 Task

Concrete grammar

<task area> ::=
 { <task symbol> **contains** <task body> | <start timer area> | <stop timer area> } |
 <macro symbol> **contains** { <macro name> [<macro call body>] }


Basic SDL-2010 is extended to allow timer set and reset to be shown with <start timer area> and <stop timer area>, respectively, and the call of a macro to be shown in a <macro symbol>.

<task body> ::=
 [<variable definitions> <end>] <non terminating statements> <end>*
 | <informal text>
 | <legacy task body>

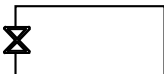
Comprehensive SDL-2010 is extended to allow <variable definitions> in a <task body> and a <legacy task body>.

<legacy task body> ::=
 <assignment> { , <assignment> }*

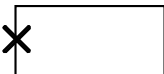
NOTE – The <legacy task body> syntax where a comma rather than <end> is used to separate one <assignment> from another is provided only so that legacy descriptions are valid, and should not be used in new descriptions.

<macro symbol> ::=


<start timer area> ::=
 <start timer symbol> **contains** <set body>

<start timer symbol> ::=


<stop timer area> ::=
 <stop timer symbol> **contains** <reset body>

<stop timer symbol> ::=


Model

If a <task body> is empty, the <task area> is removed. Any syntactic item leading to such an empty <task area> shall then lead directly to the item following the <task area>.

A <task area> defined by a <macro symbol> is transformed into a <task area> defined by a <task symbol> containing a <macro call> with the same <macro name> and <macro call body>, if one was present.

A <start timer area> is transformed into a <task area> defined by a <task symbol> containing a <set statement> with the same <set body> as the <start timer area>. A <stop timer area> is transformed into a <task area> defined by a <task symbol> containing a <reset statement> with the same <reset body> as the <stop timer area>.

11.13.2 Create

Concrete grammar

<create body> ::= { <agent identifier> | <agent type identifier> | **this** } [<actual parameters>]

Basic SDL-2010 <create body> is extended to allow <agent type identifier> as a shorthand.

If an <agent type identifier> is used in a <create body>, then the corresponding agent type shall not be defined as <abstract> or contain formal context parameters.

Model

If <agent type identifier> is used in a <create body> of a <create request area>, the following models apply.

- If there is one instance set (explicit or implicit) of the indicated agent type in the agent containing the instance that performs the create, the <agent type identifier> is derived syntax denoting this instance set.
- If there is more than one (explicit) instance set, it is determined at interpretation time in which set the instance will be created. The <create request area> is in this case replaced by a non-deterministic decision using **any** followed by one branch for each instance set. In each of the branches, a create request for the corresponding instance set is inserted.
- If there is no explicit instance set of the indicated agent type in the containing agent, the <agent type identifier> in the <create request area> is derived syntax for the implicit instance set in this context, which has a unique name.

NOTE – A context never has more than one implicit instance set of a given agent type.

11.13.3 Procedure call

Procedure call is as defined in [ITU-T Z.102].

11.13.4 Output

Concrete grammar

<output symbol> ::= <plain output symbol>
| <internal output symbol>

<internal output symbol> ::=



NOTE 1 – There is no difference in meaning between a <plain output symbol> and an <internal output symbol>.

<output body> ::=
 <output body item> { , <output body item> } *
 <communication constraints>

Basic SDL-2010 <output body> is extended to allow more than one <output body item>.

Comprehensive SDL-2010 <communication constraints> of an <output body> of an <output area> is extended to allow more than one **to** <destination> clause.

Model

The model for a <channel identifier> as a <via path> of <communication constraints> is described under *Concrete grammar* in clause 11.3.

If there is more than one <output body item> specified in an <output body> of an <output area>, the <output area> is transformed to an <output area> sequence each with a single <output body item> in the same order as specified in the original <output area>. The <communication constraints> are repeated in each <output body>.

If <communication constraints> of an <output body> of an <output area> contains more than one **to** <destination> clause, this is a shorthand for replacing the <output area> by an <output area> sequence, one for each **to** <destination>. Each <output area> has the same original <output body item> list, but in each case the <communication constraints> contains only one **to** <destination> taken in order from the original <communication constraints>.

11.13.5 Decision

Concrete grammar

<graphical answer> ::=
 <answer> | (<answer>)

Basic SDL-2010 <graphical answer> is extended to allow the round brackets to be omitted. The two alternatives of <graphical answer> have the same meaning.

11.14 Statement lists

A <statements> (a list of statements) is used in a <procedure definition> to textually define the actions to be interpreted in the procedure.

Concrete grammar

<statements> ::=
 <non terminating statements> [<end>+ [<connector name> :] <terminating statement>]
 | [<connector name> :] <terminating statement>

The Comprehensive SDL-2010 <statements> list is extended to allow a <connector name> for a <terminating statement>.

NOTE – If local variables are required in a <procedure definition>, it is possible to define them as an <entity in procedure> or alternatively by using a <compound statement> containing <variable definitions> as the <statements> list.

<non terminating statement> ::=
 [<connector name> :] <statement>
 | <compound statement>
 | <loop statement>
 | <decision statement>

The <non terminating statement> of Comprehensive SDL-2010 is extended to allow a <connector name> for a <statement>.

```

<statement> ::=
    <assignment statement>
    | <set statement>
    | <reset statement>
    | <output statement>
    | <create statement>
    | <export statement>
    | <call statement>
    | <expression statement>

```

The <statement> of Comprehensive SDL-2010 is extended to include <expression statement>, which is an operation application used as a statement where the <operation application> is interpreted and the result is ignored.

```

<expression statement> ::=
    <operation application>

```

A <return statement> is allowed in a <procedure definition>.

Model

A <non terminating statements> list with a <connector name> before a <terminating statement> is transformed into <non terminating statements> list, where the <terminating statement> is replaced by a <compound statement> formed by adding a <left curly bracket> after the <connector name> and associated colon before the <terminating statement>, and adding a <right curly bracket> after the <terminating statement>.

A <non terminating statement> with a <connector name> is transformed into a <compound statement> by adding a <left curly bracket> after the <connector name> and associated colon before the <statement>, and adding a <right curly bracket> after the <statement>.

An <expression statement> is transformed into a <call statement>, where the <procedure call body> is constructed from the <operation identifier> and the <actual parameters> of the <operation application>.

11.14.1 Compound node

Compound node is as defined in [ITU-T Z.102].

11.14.2 Decision statement – if statement

An if statement is a limited form of decision statement: a <Boolean expression> is interpreted and if it returns the predefined Boolean value true, a consequence statement is interpreted; otherwise, an alternative statement, if present, is interpreted.

Concrete grammar

```

<decision statement> ::=
    <if statement>
    | [ <connector name> : ] decision ( <question> ) [ <comment body> ]
    <left curly bracket>
        <decision statement body>
    <right curly bracket>

```

Comprehensive SDL-2010 <decision statement> is extended to include <if statement> as a shorthand.

```

<if statement> ::=
    [ <connector name> : ]
    if ( <Boolean expression> ) [ <non terminating statement> ]
    [ else [ <alternative statement> ] ]

```

The <non terminating statement> (or the absence of a <non terminating statement>) of an <if statement> is a consequence statement. The **else** and <alternative statement> associates with the closest preceding consequence statement.

```
<loop decision statement> ::=
    <loop if statement>
  | [ <connector name> : ] decision ( <question> ) [ <comment body> ]
    <left curly bracket>
      <loop decision statement body>
    <right curly bracket>
```

Comprehensive SDL-2010 <loop decision statement> is extended to include <loop if statement> as a shorthand. A <loop if statement> is essentially the same as an <if statement> except the possible kinds of consequence and alternative statement are different.

```
<loop if statement> ::=
    [ <connector name> : ]
    if ( <Boolean expression> )
      [ <statement in loop> | <loop terminating statement> ]
    [ else [ <loop alternative statement> ] ]
```

Model

An <if statement> (or <loop if statement>) is transformed to the following <decision statement> (or <loop decision statement> respectively):

```
decision boolean_expression {
    ( true ) : consequence_statement
    ( false ) : alternative_statement
}
```

where *boolean_expression*, *consequence_statement* and *alternative_statement* represent actual text for the <Boolean expression>, consequence statement and alternative statement, respectively.

11.15 Timer

Concrete grammar

```
<reset body> ::=
    ( <reset clause> { , <reset clause> }* )
```

Basic SDL-2010 <reset body> is extended to allow more than one <reset clause>.

```
<set body> ::=
    <set clause> { , <set clause> }*
```

Basic SDL-2010 <set body> is extended to allow more than one <set clause>.

Model

A <task area> is allowed to contain several <reset clause>s or <set clause>s. This is derived syntax for specifying a sequence of <task area>s, one for each <reset clause> or <set clause> such that the original order in which they were specified in <task area> is retained. This shorthand is expanded before shorthands in the contained expressions are expanded.

12 Data

See Basic SDL-2010, the active use of data as defined in Comprehensive SDL-2010 for any imperative expression that is an import expression, and [ITU-T Z.104], if any feature defined in that Recommendation applies. The <signal list definition> shorthand is introduced for interface definitions below.

12.1 Data definitions

As defined in Basic SDL-2010 and [ITU-T Z.104], if any feature defined in that Recommendation applies, except for interface definition which is extended for the shorthand <signal list definition>.

12.1.1 Data type definition

As defined in Basic SDL-2010 and [ITU-T Z.104], if any feature defined in that Recommendation applies.

12.1.2 Interface definition

As defined in Basic SDL-2010 and [ITU-T Z.104], if any feature defined in that Recommendation applies, plus extension of the shorthand <signal list definition>.

Concrete grammar

```
<signal list definition> ::=  
    signallist <interface name> <equals sign> <signal list> <end>
```

Model

A <signal list definition> is an alternative concrete syntax, and is transformed into an <interface definition> using the keyword **interface** with no <package use clause>, no <virtuality>, an <interface heading> containing the <interface name> (but no <formal context parameters> or <virtuality constraint>), no <interface type expression> specialization and no <entity in interface>. The <interface definition> has an <interface use list> with a <signal list> that is the same as the <signal list> of the <signal list definition>.

12.2 Use of data

As defined in Basic SDL-2010 and [ITU-T Z.104], if any feature defined in that Recommendation applies.

12.3 Active use of data

As defined in Basic SDL-2010 and [ITU-T Z.104], if any feature defined in that Recommendation applies.

13 Generic system definition

13.1 Optional definition

Concrete grammar

```
<select definition> ::=  
    select if ( <Boolean simple expression> ) [ <end> ]  
    {  
        | <signal definition list>  
        | <data definition>  
        | <variable definition>  
        | <timer definition>  
        | <macro definition>  
        | <remote variable definition>  
        | <remote procedure definition>  
        | <select definition>  
        | <operation definitions>  
        | <block reference>  
        | <process reference>
```

```

| <agent type reference>
| <procedure definition>
| <imported variable specification>
| <imported procedure specification>
| <signal list definition> }+
endselect <end>

```

Comprehensive SDL-2010 <select definition> is extended to include <block reference>, <process reference>, <agent type reference>, <procedure definition>, <imported variable specification>, <imported procedure specification> and <signal list definition>.

<option area> ::=

```

<option symbol> contains
{ select if ( <Boolean simple expression> ) [ <end> ]
  {
    <agent type reference area>
    | <agent area>
    | <channel definition area>
    | <package text area>
    | <agent text area>
    | <procedure text area>
    | <composite state type reference area>
    | <state partition area>
    | <procedure reference area>
    | <option area>
    | <create line area> } + }

```

Comprehensive SDL-2010 <option area> is extended to include <create line area>.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems