INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Z.109

(11/99)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

SDL combined with UML

ITU-T Recommendation Z.109

(Previously CCITT Recommendation)

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	Z.120–Z.129
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.399
QUALITY OF TELECOMMUNICATION SOFTWARE	Z.400–Z.499
METHODS FOR VALIDATION AND TESTING	Z.500–Z.599

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

ITU-T RECOMMENDATION Z.109

SDL COMBINED WITH UML

Summary

Objective

This Recommendation defines the specialized subset of UML that maps directly to SDL and that can be used in combination with SDL.

Coverage

This Recommendation presents a definition of the UML-to-SDL mapping for use in the combination of SDL and UML.

Application

The main area of application of this Recommendation is the specification of telecommunication systems. The combined use of SDL and UML permits a coherent way to specify the structure and behaviour of telecommunication systems, together with data.

Status/stability

This Recommendation is the complete reference manual describing the UML to SDL mapping for use in the combination of SDL and UML.

Associated work

Recommendation Z.100: SDL.

Source

ITU-T Recommendation Z.109 was prepared by ITU-T Study Group 10 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on 19 November 1999.

FOREWORD

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

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

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 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.

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The 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, the ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Recommendation Z.109

SDL COMBINED WITH UML

(Geneva, 1999)

1 Introduction

SDL (Specification and Description Language) combined with UML (Unified Modelling Language) is defined by this Recommendation together with Recommendation Z.100. This Recommendation defines an *SDL UML Profile* based upon UML [1], and SDL-2000 [2]. The relevant parts of the graphical grammar of SDL and other notation elements in SDL are defined in [2].

This Recommendation does not cover combination of UML and MSC [3].

1.1 Objective

The objective is to take advantage of the formal basis of SDL and the expressiveness of UML, in particular the use of UML class diagrams with associations. By using the specialized subset of UML defined by this Recommendation, it is possible to express parts of an SDL specification in UML.

1.2 Principles of the SDL-UML combination

This Recommendation defines a *UML Specialization and Restriction for combination with SDL* (an *SDL UML Profile*). It ensures a well-defined mapping between parts of a UML model and an SDL model. For each of the UML ModelElements that are included in *SDL UML*, there is a *one-to-one* mapping to the corresponding SDL concepts. The mapping is based upon the UML meta-model and upon the abstract grammar of SDL. A tool that implements *SDL UML* must support these specializations and restrictions and be able to provide this one-to-one mapping.

UML specializations and restrictions are defined in terms of the UML meta-model and the abstract grammar of SDL, i.e. independent of notation.

This Recommendation gives no notation guidelines for *SDL UML*. For some of the UML elements, SDL contains elements that have a UML-like graphical notation. A tool for the combined use of UML and SDL may use a *de facto* UML graphical notation standard for the UML covered by this Recommendation, but the SDL specific part of this tool must provide the graphical grammar for these elements as defined by Recommendation Z.100.

1.3 Restrictions on SDL and UML

There are no restrictions on SDL. However, not all of SDL is covered by *SDL UML*.

A general restriction on *SDL UML* is that only the ModelElements defined in the *SDL UML Profile* ensure a one-to-one mapping. In a combined use of UML and SDL, more parts of UML can be used, but the mapping of these cannot be guaranteed to work the same with different tools.

1.4 SDL UML mapping

The SDL UML mapping is defined by means of the UML extension mechanisms: Stereotypes, Tagged Values, and Constraints, by restricting the use of UML, and by attaching more specific meanings to UML.

UML *classes* generally represent entity *types* of SDL. In most cases the entity *kind* is represented by *stereotypes*. Where predefined model elements, stereotypes or keywords exist in UML that have a similar meaning as in SDL, they have been used.

1.5 Well-formedness rules

Two general well-formedness rules on *SDL UML* are:

- Only model elements that have mappings to SDL (and defined by this Recommendation) can be used;
- The *SDL UML* model shall adhere to the (static) semantics of SDL.

An implication of these rules is e.g. that the *ownedElement* of a class that represents an SDL type must be an *SDL UML* class, which represent a type defined in the scope unit of the SDL type.

For each of the model elements of *SDL UML*, specific well-formedness rules are described.

1.6 Conventions

The definition of *SDL UML* follows the same organization as the definition of the UML Semantics. For each UML ModelElement, a table describes the mapping to SDL, followed by possible restrictions and specializations. Each table follows the generalization hierarchy of the meta-model, and has an entry for each model element, attribute and association that are considered or has a mapping. Elements of the abstract grammar of SDL are hyphenated and in italics, e.g. *Variable-definition.*

Notation

not applicable	not used in *SDL UML*
no concept	no corresponding concept in SDL
default	default mapping
mandatory	mandatory mapping
^	the super-classes of the model element for which special mapping applies
.	attribute of the model element
=	possible value of attribute
->	association of the model element

1.7 The structure of this Recommendation

This Recommendation is organized according to the meta-model of UML. The reason for this is the intended use of this Recommendation: "Which parts of UML can be used in combination with SDL and how they would map to SDL". The mapping between UML and SDL could as well have had SDL as the starting point.

In case of differences between this Recommendation and Recommendation Z.100 [2] on the description of SDL, the definition in Recommendation Z.100 [2] applies.

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; all 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.

[1] Unified Modelling Language (UML) (OMG UML Specification v. 1.3: OMG document ad/99-06-08).

[2] ITU-T Recommendation Z.100 (1999), *Specification and Description Language (SDL)*.

[3] ITU-T Recommendation Z.120 (1999), *Message Sequence Charts (MSC)*.

3 *SDL UML* ModelElements

The main structure of *SDL UML Profile* follows the meta-model of UML. Correspondence to SDL is defined by mapping the UML meta-model to the grammar of SDL.

3.1 Summary of *SDL UML* Model Elements

SDL UML provides a specialization and restrictions of the following model elements:

– Model management:
  - Package
  - Model

– Classes, with stereotypes:
  - «system»
  - «block»
  - «process»
  - «procedure»
  - «interface»
  - «object»
  - «value»
  - «state»

– «signal» Classifier

– Associations:
  - composition as a partial representation of containment between agents
  - association stereotype with gate representing a gate with endpoint constraint
  - other associations representing the corresponding associations in SDL

– Generalization

– Dependencies:
  - «import»
  - «create»

– State machine

3.1.1 Stereotypes

The following UML standard elements are used:

Name	Applies to	Description
model	Package::Model	SDL specification (from Model Management)
interface	Classifier::Interface	Interface
signal	Classifier	Signal definition
create	Dependency::Usage	Create
import	Dependency::Permission	Package reference clause

This Recommendation defines the following stereotypes:

Name	Applies to	Description
system	Class	System (is also stereotype of Model in UML, but here Class is used)
block	Class	Block
process	Class	Process
procedure	Class	Procedure
object	Class	Object Type
value	Class	Value Type
state	Class	Composite state type
gate	Association	Gate with endpoint constraint
signature	Operation	Formal parameters of agents

3.1.2 TaggedValues

This Recommendation defines the following tagged values:

Name	Applies to	Description
encloser	Class	Specifies the enclosing type (scope unit). The value is the type identifier.
• virtual • redefined • finalized	Class	Specifies the <virtuality> of a type, procedure or operator. • **virtual** • **redefined** • **finalized**
remote	• Attribute • Procedure	• Remote variable • Remote procedure Values are Boolean.

3.1.3 Constraints

Name	Applies to	Description
atleast	Class	Represents a virtuality constraint on a virtual type
atleast	Parameter	Represents a virtuality constraint on a context parameter

3.2 Core Model Elements

Restrictions, specializations and mappings that apply to the general elements of the Core UML meta-model are given in this subclause. They apply if nothing else is specified for more specific model elements.

3.2.1 Abstraction

Not applicable.

3.2.2 Association

In this version of the Recommendation, associations are used for representing the following SDL concepts:

- Composite aggregation for containment, i.e. entity sets contained in entities;
- Possible connections of agent sets (by channels connecting gates with interfaces) as part of the internal structure of an agent;
- Gate with endpoint constraint by a stereotyped association with gate.

A subset of associations other than these are valid (and described below), but are considered as comments and mapped to the corresponding associations in SDL.

Associations do *not* cover sets of types being connected by channels:

1) Associations are used to define properties of types, while channels are used (in SDL structure diagrams) to define links between sets of instances. Different channels can connect different typed-based sets based on the same type. Therefore, the connection of agent sets by means of channels is represented by collaborations in UML.
2) As basis for the use of collaborations, all agent types that have interfaces and gates that may be connected by channels will, in the SDL UML mapping, have implicit associations. These associations do not imply properties of the associated classes.

3.2.2.1 Composite aggregation

Composite aggregation is used to represent that instances of an agent type contain type-based sets of agents. For example, the instances of a system type contain sets of blocks. The sets are represented by the endpoints of the composite aggregation, with the role names being the names of the sets and the multiplicity being the number of instances.

Composition aggregation has not exactly the same meaning in UML as containment in SDL. This Recommendation imposes the more restricted meaning of SDL containment to UML composition. An element of a contained agent set *cannot* change set membership, as is the case in UML. A contained agent set and its members cannot exist without the container, in the same way as in UML.

Note that composite aggregation only implies properties for the type at the composite end. The type at the other end is not affected.

3.2.2.2 Gate endpoint constraint

An association stereotyped with gate represents that the agent type corresponding to the source end class has a gate with an endpoint constraint corresponding to the target end class. The name of the association is the name of the gate. Only interfaces, signals and remote procedures associated with the outgoing direction of the gate are represented, not signal lists.

3.2.2.3 General associations

Although general associations in *SDL UML* are comments in the corresponding SDL, this version of the Recommendation supports only a subset of UML associations.

UML	SDL
Association	• containment: agent sets as part of entities • gate with endpoint constraint • general associations: association as comments, but restricted by the following association elements *not* being supported: – n-ary associations (n>2) – Qualified association ends – Association Class – Aggregation – Changeability – Target scope
->connection	
^GeneralizableElement	not applicable
^ModelElement	
.name	• containment: not applicable • gate with endpoint constraint: *Gate-name* • otherwise: <association name>

3.2.3 Association class

Not applicable.

3.2.4 Association end

In the following descriptions, when referring to an association end for a binary association, the *source* end is the other end; the *target* end is the one whose properties are being discussed.

An AssociationEnd may represent a partial specification of two SDL concepts:

– Containment:
  - Agents containing other agents: The association must be a composition and the classes involved must represent agent types.
  - Composite states containing type based states: The association must be a composition and the classes involved must represent state types.

– The endpoint constraint of a gate: The association must then be stereotyped with «gate».

An AssociationEnd may also just be the end of an ordinary association. It will then just represent the corresponding association end of the corresponding SDL association. This means that a composition that is not covered by the two cases above just represent the corresponding composition association in SDL.

UML	SDL
AssociationEnd	• containment • gate with endpoint constraint in the entities at the source end • otherwise: <association end>
.aggregation = composite	• containment, that is the end represents an agent (system/block/process) type that contains sets of blocks and/or processes or the end represents a composite state with contained type-based states • otherwise: <association kind> = – <composition not bound kind>, or – <composition part end bound kind>, or – <composition composite end bound kind>, or – <composition two ends bound kind>
= aggregate	• <association kind> = – <aggregation not bound kind>, or – <aggregation part end bound kind>, or – <aggregation aggregate end bound kind>, or – <aggregation two ends bound kind>
= none	• if source end of association has aggregation = composite: the target end represents a set of agents or a type-based composite state as part of a composite state type • if association stereotyped with gate: gate with endpoint constraint
.changeability = none = frozen = addOnly	not applicable
.ordering = unordered = ordered	not applicable for containment and for gate endpoint constraint, otherwise: • *default* • **ordering**
.isNavigable = true = false	not applicable for containment and for gate endpoint constraint, otherwise: • navigable • not applicable
.multiplicity	• for containment: represents *Number-of-instances* • for gate endpoint constraint: 1 • otherwise: <multiplicity> of <association end>
.targetScope = instance = classifier	• entity • not applicable

UML	SDL
.visibility = public = protected: default = private	• for containment: not applicable • for gate endpoint constraint: not applicable • otherwise: – **exported** – not applicable – local
->qualifier	not applicable
->specification	• for containment: not applicable • for gate endpoint constraint: interfaces, signals and remote procedures associated with the outgoing direction of the gate • otherwise: <specifier>
->type	• for containment: type of type-based agent set • for gate endpoint constraint: the endpoint constraint type • otherwise: <linked type>
^ModelElement	
.name	• for containment: *Agent-name* in *Agent-definition* • for gate endpoint constraint: not applicable • otherwise: <role name>

The *type* can only denote model elements that are *SDL UML* classifiers, except interfaces.

3.2.5 Attribute

Attributes represent variables of agents and procedures, or fields of data objects or values, or signal parameters.

UML	SDL
Attribute	variable, field or signal parameter
.changeability = changeable = frozen = addOnly	 • mandatory • not applicable • not applicable
.initialValue	<default initialization> [:= <ground expression>] in <variables of sort> in *Variable-definition*
.multiplicity	Note – May be used for ASN.1 types.
.targetScope	
->type	*Sort-reference-identifier* in *Variable-definition*, as part of *Signal-definition*, or as part of field definition
->associationEnd	not applicable
^Feature	
.ownerScope = instance = classifier	 • mandatory • not applicable

UML	SDL
.visibility = public = protected = private	 • **exported** variable or public field • local variables of agents, and protected field • not applicable to variables of agents, but for private fields
-> owner	the entity defining the variable or field
^ModelElement	
.name	*Variable-name*/<field name>

The *type* of an attribute is restricted to be a class defined by this Recommendation.

The following rules apply:

- *type* = class with stereotype «block» or «process»: typed Pid
- *type* = class with stereotype «object»: object reference variable
- *type* = class with stereotype «value»: value type variable

Predefined types are represented by corresponding predefined classes of stereotype «process», «object» or «value».

3.2.6 BehaviouralFeature

UML	SDL
BehaviouralFeature	Operator or procedure
.isquery *= true* *= false*	 • *Static-operator*/*Dynamic-operator* (operator) • *Static-operator*/*Dynamic-operator* (method) or *Procedure-definition*
->Parameter	• *Argument-list* in *Signature* • *Procedure-formal-parameter**

3.2.7 Binding

A *binding* is a subclass of Dependency that represents the provision of actual context parameters in SDL. The binding as an explicit relationship does not exist in SDL, but it may be described in this way in UML.

For a *binding*, the arguments represent model elements within the scope (namespace) of the client that are actual parameters for the templateParameters of the source. The actual parameters must map to valid context parameters for the SDL type that is represented by the source.

Partially instantiated templates correspond to parameterized types where not all context parameters are provided.

UML	SDL
Binding	<type expression> with <actual context parameters>
->argument	<actual context parameter>
^Dependency	
->client	the type being defined using the <type expression>
->supplier	the <base type> (in <type expression>)
^ModelElement	
.name	not applicable

The *client* and *supplier* must be classes with the same stereotype.

3.2.8 Class

Classes represent SDL types and procedures. The different kinds of entity types in SDL are represented by stereotypes.

A type in SDL is completely defined by a type definition/diagram and possibly by a type reference (in case the type definition/diagram is referenced). A class defines parts of the type *and* possibly a type reference. The parts it defines depend upon the content of the class compartments and its composition. The choice of properties to be defined in *SDL UML* is left to the specifier, but specified properties shall be consistent with the corresponding properties defined in the SDL type definition/diagram.

The attributes and associations of a class are restricted/specialized and mapped to SDL as described in the following table. For each of the different SDL entity kinds, more details are given in succeeding tables in 3.2.8.1 to 3.2.8.4. Attributes defined in Foundation Package:Auxiliary Elements are also included here.

UML	SDL
Class	types and procedures
.isActive = true = false	 • *Agent-type-definition* • *Composite-state-type-definition*, *Data-type-definition*, *Procedure-definition*, *Signal-definition*
^Classifier	
->feature	• depends on entity kind
->participant – aggregation of association end: = composite = aggregate = none	 • contained agent sets • not applicable • if other end of association has aggregation = composite: a set of entities
^GeneralizableElement	<specialization>
.isAbstract	**abstract**
.isLeaf	not applicable
.isRoot	can be derived from the type definition
->generalization	Supertype – the type identified as part of the <type expression> of <specialization> for the type represented by the classifier
->specialization	can be derived from the *generalization*
^Namespace	scope unit defined by the type
->ownedElement	entities in a subset of the set of definitions allowed in the scope unit – depends on kind of type
^ModelElement	

UML	SDL
->taggedValue • (virtuality) • virtual • redefined • finalized	 • <virtuality> • **virtual** • **redefined** • **finalized**
->constraint	<virtuality constraint>
->supplierDependency • Realizes dependency to interfaces implemented by the class • Usage («create») dependency to classes from which objects are created by objects of this class • Permission («import») dependency from other packages using the class • Annotated generalization relation	Dependencies where this type is a supplier • <interface gate definition> **in with** <interface identifier> • <create line area> • <package use clause>/<definition selection list> • <type expression>
->clientDependency • Usage («create») dependency to classes from which objects create objects of this class • Permission «import» dependency indicating which package this class uses • Annotated generalization relation	Dependencies where this type is a client • <create line area> • <package use clause> on this type definition • <type expression>
->namespace	the owning (enclosing) scope unit of the entity type or procedure corresponding to the class

3.2.8.1 Agent

The following applies to agent types.

The signature operation feature is introduced in order to specify the formal parameters, as UML does not provide parameters for classes. It has the format of an operation with the process name as the operation name and with the sterotype «signature».

UML	SDL
{ «system» | «block» | «process» } Class	*Agent-type-definition* • if a *templateParameter* is defined, it is a parameterized agent type, • if a *generalization* is defined, it is a subtype
.isActive = true = false	 • mandatory • not applicable
^Classifier	
->feature • attributes • operations	 • *Variable-definition* • *Procedure-definition, or* • *Procedure-definition* (signature) that represents <formal parameters> of the agent type
->participant	Containment or other AssociationEnd where agent may be participant
^GeneralizableElement	
->generalization	the *Agent-type-definition* identified by *Parent-type-identifier* as part of this agent type
^Namespace	scope unit defined by the agent type
->ownedElement	entities in the subset of <entity in agent> defined below
^ModelElement	
->taggedValue • virtual • redefined • finalized	<virtuality> of agent type except for system type
->constraint	<virtuality constraint> on agent type except for system type
->namespace	• *Package-definition* with the *Agent-type-definition* • *Agent-type-definition* defining the agent type

The *generalization* can only be one class, and it must be of the same stereotype as this class.

The *ownedElement* are restricted to model elements that represent entities in the following subset of <entity in agent>:

- *Signal-definition*;
- *Variable-definition*;
- *Procedure-definition*;
- <remote procedure definition>;
- <remote variable definition>;
- *Data-type-definition*;
- *Composite-state-type-definition*;

- *Interface-definition*;
- *Agent-type-definition*; or
- *Agent-definition*.

The *namespace* must be either a package or a class stereotyped with «system», «block», or «process». The *namespace* of a «system» or «block» class can not be a «process» class, and the namespace of a «system» class can not be a «block» or «process» class.

3.2.8.2 Procedure

Procedure is a stereotype of class that corresponds to an SDL procedure. This is done in order to cover specialization of procedures, which is not part of UML.

UML	SDL
«procedure» Class	*Procedure-definition* or <remote procedure definition> • if a *templateParameter* is defined it is a parameterized procedure • if a *generalization* is defined it is a sub-procedure
.isActive = false = true	 • mandatory • not applicable
->taggedValue remote	 <remote procedure definition>
^Classifier	
->feature • attributes • operations	 • *Variable-definition* • *Procedure-definition*
->participant	not applicable
^GeneralizableElement	
->generalization	the *Procedure-definition* identified by the optional *Procedure-identifier* as part of the current *Procedure-definition*
^Namespace	scope unit defined by the process type
->ownedElement	entities in the subset of <entity in procedure> defined below
^ModelElement	
->taggedValue • virtual • redefined • finalized	<virtuality> of procedure • **virtual** • **redefined** • **finalized**
-> namespace	• *Package-definition* with the *Procedure-definition* • *Agent-type-definition* with the *Procedure-definition* • *Procedure-definition* with *Procedure-definition*

The *ownedElement* are model elements that represent entities in the following subset of <entity in procedure>:

- *Data-type-definition*;
- *Variable-definition*; or
- *Procedure-definition*.

The *namespace* must be a package or a class with stereotype «system», «block», «process», or «procedure».

3.2.8.3 Data types

Classes stereotyped with *value* and *object* represent data types: *value* is a class that corresponds to an SDL value-type, and *object* is a class that corresponds to an SDL object-type.

UML	SDL
• «value» Class • «object» Class	• *Value-type* • *Object-type* For both: • if a *templateParameter* is defined, it is a parameterized type • if a *generalization* is defined, it is a subtype
.isActive = false = true	 • mandatory • not applicable
^Classifier	
->feature • attributes • operations	 • <field> in <structure definition> • *Static-operator/Dynamic-operator* (operators and methods)
->participant	containment of the data type or any association endpoint where the type may be participant
^GeneralizableElement	
->generalization	the *Data-type-definition* identified by *Data-type-identifier* as part of the current *Data-type-definition*
^Namespace	scope unit defined by the *Data-type-definition*
->ownedElement	*Data-type-definition*s in the scope unit of the entity type represented by this class
^ModelElement	
->namespace	• *Package-definition*, *Agent-type-definition*, *Procedure-definition*, *Data-type-definition* with the *Data-type-definition*

The *generalization* must be «value» or «object» classes.

The *ownedElement* must be classes that map to *Data-type-definition*s.

The *namespace* must be a package, or a class with stereotype «system», «block», «process», «procedure», «state», «object», or «value».

3.2.8.4 State

State is a class stereotype that represents an SDL composite state type. SDL has the notion of composite state type (in addition to composite state), and as UML state machines do not have that, it is in *SDL UML* represented by a class with a stereotype state.

The state (entry and exit) connection points are represented as operations, as these are the visible properties of a composite state type.

UML	SDL
«state» Class	*Composite-state-type-definition* • if a *templateParameter* is defined, it is a parameterized composite state type • if a *generalization* is defined, it is a subtype
.isActive = false = true	 • mandatory • not applicable
^Classifier	
->feature • attributes • operations	 • *Variable-definition*s • *Procedure-definition*s • state connection points as defined by *State-entry-point-definition*
->participant	not applicable
^GeneralizableElement	
->generalization	the *Composite-state-type-definition* identified by [*Parent-type-identifier*] as part of this *Composite-state-type-definition*
^Namespace	scope unit defined by this *Composite-state-type-definition*
->ownedElement	entities in the subset of <entity in composite state type> defined below
^ModelElement	
.name	the *State-type-name*
->taggedValue • virtual • redefined • finalized	<virtuality> of the composite state type
->constraint	<virtuality constraint>
->namespace	*Package-definition*, *Agent-type-definition*, *Procedure-definition*, *Composite-state-type-definition* with the *Composite-state-type-definition*

The *generalization* must be one "state" Class.

The *ownedElement* must be model elements that represent entities in the subset of <entity in composite state>, that is:

- *Variable-definition*;
- *Data-type-definition*;
- *Procedure-definition*;
- <textual procedure definition>; or
- *Composite-state-type-definition*.

The *namespace* must be a package or a class with stereotype «system», «block», «process», «procedure», or «state».

3.2.9 Classifier

A classifier represents the common features of entity types, interface types, signals and procedures. If not specifically specified for the various kinds, the following applies.

UML	SDL
Classifier	entity types, interfaces, signals and procedures
->feature	variables, procedures, methods, operators, signal parameters – depending on entity kind
->participant	depend on kind of classifier
->powertypeRange	not applicable
^GeneralizableElement	depends on kind
^Namespace	scope unit
->ownedElement	entities defined in the scope unit of this entity
^ModelElement	
.template	the parameterized type if this type is a <formal context parameter>
.templateParameter	<formal context parameters> for a parameterized type
->constraint	<virtuality constraint>, if present
->supplierDependency	depends on kind
->clientDependency	depends on kind
->taggedValue • virtual • redefined • finalized	<virtuality>, if present • **virtual** • **redefined** • **finalized**

3.2.10 Comment

A comment is an annotation attached to a model element that represents a note or a comment attached to the SDL entity corresponding to the model element.

3.2.11 Component

Not applicable.

3.2.12 Constraint

General constraints are not supported, as there are no corresponding concepts in SDL. Virtuality constraints are supported. The UML text is the <virtuality constraint> as text.

UML	SDL
Constraint	
.body = atleast – Class name – Parameter name	 • <virtuality constraint> on a virtual type or virtual procedure/method • <virtuality constraint> on a <formal context parameter>
->constrainedElement	• a virtual type if the constraint is a virtuality constraint on a virtual type • a <formal context parameter> if the constraint is a virtuality constraint on a context parameter
^ModelElement	
.name	not applicable

3.2.13 Data type

Not applicable.

3.2.14 Dependency

Dependencies are used to represent:

- «create» dependency for create lines;
- «import» dependency to represent that a package uses the definitions of other packages; and
- «realizes» dependency to interfaces supported by the class.

3.2.15 Element

Not applicable in itself, but attributes are used; see the mapping of the more specific meta-model elements.

3.2.16 ElementOwnership

Not applicable. All contained elements are parts of the specification of the containing scope unit, and the visibility of contained elements is provided by properties of these.

3.2.17 ElementResidence

Not applicable.

3.2.18 Feature

Abstract class. Attributes and associations are detailed in connection with behaviouralFeature and structuralFeature.

3.2.19 Flow

Not applicable.

3.2.20 GeneralizableElement

Abstract class. Attributes and associations are detailed in connection with its subclasses.

3.2.21 Generalization

SDL specialization is represented by generalization in UML. This is also done for procedures. The UML meta-model has the notion of Generalization, while SDL has the opposite notion of Specialization.

Generalization has the opposite <specialization> counterpart in SDL, but it reflects the corresponding inheritance property of the subtype.

UML	SDL
Generalization	<specialization>
.discriminator	not applicable
->child	a subtype
->parent	*Parent-type-identifier* (<base type> (in <type expression>))of subtype,
->powertype	not applicable
^ModelElement	
.name	not applicable

Generalization applies to the stereotyped classes for the following kinds of types: systems, blocks, processes, states, objects, values, for procedures, and for interface and signals (Classifiers stereotyped with «interface» or «signal»).

The *parent* must be one class with the same stereotype as the subtype, except for interfaces that may have multiple parents.

The Standard Constraints do not apply.

3.2.22 Interface

Interfaces are Classifiers that map to interfaces in SDL. UML interfaces can only have operations (corresponding to exported procedures in SDL), while SDL interfaces also can have variables and signals. In *SDL UML*, interfaces therefore have a list of operations that represent signals, and exported variables. The stereotype «signal» is used to indicate signals, while variables and procedures are represented as operations.

Interfaces can only be applied to classes that represent agent types, not to classes that represent procedures, state type or data types.

UML	SDL
Interface	*Interface-definition*
^Classifier	
->feature • *attribute (not supported)* • *operation*	 • not applicable • *Signal-definition*, <remote variable definition> and <remote procedure definition>
->participant	association ends that are interfaces, in which case aggregation = none
^GeneralizableElement	
.isAbstract	**Abstract**
.isLeaf	not applicable
.isRoot	can be derived from the *Interface-definition*
->generalization	super interfaces, which are the interfaces identified by <interface specialization> for this *Interface-definition*
->specialization	can be derived from the *generalization*
^Namespace	scope unit of this *Interface-definition*
->ownedElement	<entity in interface>: the entities defined in the scope unit of this *Interface-definition*
^ModelElement	
->supplierDependency	realizes dependency representing **implements**
->clientDependency	the opposite of *supplierDependency*

The *ownedElement* must be model elements that represent entities in <entity in interface>, that is:

- *Signal-definition*;
- <remote variable definition>; or
- <remote procedure definition>.

3.2.23 Method

UML	SDL
Method	body of *Procedure-definition*, *Static-operator*/*Dynamic-operator* (operator or method)
.body	not applicable
->specification	*Procedure-formal-parameter*, Signature*
^ModelElement	
->constraint	<virtuality constraint>

The *constraint* describes that the body of a virtual procedure or operator inherits the constraint behaviour specification. This is a specialized use of the constraint association of UML Method.

3.2.24 ModelElement

A *ModelElement* represents common properties of SDL entity (type) definitions. If not specifically specified for the various specializations, the following applies.

UML	SDL
ModelElement	entity definition
.name	<name> of entity
->clientDependency	depends on kind of entity
->constraint	depends on kind of entity
->implementationLocation	not applicable
->namespace	the owning (enclosing) scope unit where the entity is defined
->presentation	not applicable
->supplierDependency	depends on kind of entity
->templateParameter	depends on kind of entity

3.2.25 Namespace

A namespace represents an SDL scope unit.

UML	SDL
Namespace	scope unit
-> ownedElement	entity types defined in the scope unit represented by this model element

The *ownedElement* must be model elements that represent entities in SDL according to *SDL UML*.

3.2.26 Node

Not applicable, as deployment specification is outside the scope of SDL and thereby of *SDL UML*.

3.2.27 Operation

An operation represents a procedure or an operator/method. Operation is just used to represent the fact that a type defines a procedure or operator/method, together with its signature. A nested procedure-stereotyped class represents the procedure itself with the same name as the operation. Operators and methods are not represented by stereotyped classes, but by UML Methods. Operations sterotyped with «signature» represent the formal parameters of the enclosing agent type.

UML	SDL
Operation	*Procedure-definition*, *Static-operator*/*Dynamic-operator* (operator or method) or formal parameters of agent type
.concurrency = sequential = guarded = concurrent	applicable only for operator and method • depends on semantics for data • not applicable • depends on semantics for data
.isAbstract = true = false	 • **abstract** • not **abstract**
.isLeaf = true = false	 • **finalized** • **virtual** or **redefined**
.isRoot = true = false	not applicable
^Feature	
.ownerScope = instance = classifier	 • mandatory • not applicable
.visibility = public = protected: default = private	 • **exported procedure**, **public** (<visibility>) for operator/method • local **procedures**, **protected** (<visibility>) for operator/method • not applicable for procedures, **private** (<visibility>) for operator/method

Note that the formal parameters of agents are given by an operation sterotyped with «signature» and with the same name as the agent being defined by the actual class. Other operations of agents and procedures represent locally defined procedures.

The virtuality of a procedure is also specified as part of the class representing the procedure. The virtuality of operators and methods is specified as part of the UML Methods.

Note that the type of a value returning procedure or the result type of an operator can not be represented directly by a property of the corresponding *SDL UML* operation, but by a parameter of kind return.

3.2.28 Parameter

A parameter represents a formal parameter for a procedure, operator or method. Agent type parameters are represented by parameters (with *kind* = in) to a procedure with the same as the agent type and stereotyped with «signature».

UML	SDL
Parameter	*Procedure-formal-parameter*
.default value	no concept
.kind = in = inout = out = return	• *In-parameter* (**in** keyword in <parameter kind>) • *Inout-parameter* (**in/out** keyword in <parameter kind>) • not applicable • <sort> in *Result* or <operation result>
.name	*Variable-name*
-> type	*Data-type-reference-identifier* or <sort> (in *Result* as part of *Procedure-definition*)

3.2.29 Permission

The "import" permission is used to represent SDL package reference clauses, indicating which packages this class uses.

UML	SDL
Permission «import»	<package use clause>
^Dependency	
->client	the client is a *Package-definition* or type definition that uses types defined in the package represented by the supplier
->supplier	the supplier is a package used by the client
^ModelElement	
.name	not applicable

3.2.30 PresentationElement

Not applicable.

3.2.31 Relationship

This is an abstract concept and is not applicable directly.

3.2.32 StructuralFeature

This is an abstract concept, used only for Attribute. See Attribute for details.

3.2.33 TemplateParameter

This represents a context parameter in SDL.

3.2.34 Usage

Usage of the kind *create* represents create line.

3.3 Extension mechanisms

The extension mechanisms of UML can also be used in SDL UML, but as for UML it is left to the user of these to provide their meaning.

3.4 Data types

Not used. The predefined types of SDL are assumed being predefined as classes with appropriate stereotypes and properties.

3.5 Behavioural elements

Although it would be possible, this version of the Recommendation does not provide a detailed mapping of behavioural elements of UML to SDL, except for signals.

3.5.1 Common Behaviour

It is not foreseen that *SDL UML* will be used for specifying actions in details, as SDL provides complete action specification mechanisms. No mapping is thus specified for the Common Behaviour package. However, an overview of a possible mapping is given in Appendix I.

3.5.2 Signal

Signal is defined in UML as a subclass of Classifier. It corresponds to an SDL signal with the following restrictions.

UML	SDL
«signal»	*Signal-definition* • if a *templateParameter* is defined, it is a parameterized signal • if a *generalization* is defined, it is a subtype
->context	may be derived: the transitions that send signals of this type
->reception	may be derived: the agent types that have this signal in its valid input signal set
^Classifier	
->feature	The list of *Data-type-reference-identifier*s as part of *Signal-definition*, i.e. signal parameters
->participant	not applicable
^GeneralizableElement	
->generalization	the *Signal-definition* identified as part of the <type expression> of <specialization> for this *Signal-definition*
^Namespace	scope unit defined by the *Signal-definition*
->ownedElement	not applicable
^ModelElement	
->taggedValue	no concept for virtual signals
->constraint	not applicable
->supplierDependency	not applicable
->clientDependency	not applicable
->namespace	*Package-definition* or <agent type definition> defining the *Signal-definition*

The *generalization* must be one «signal» classifier.

The *namespace* must be a package or a class of agents («system», «block», or «process»).

3.6 Collaborations

A subset of Collaborations is used to represent the internal agent/state machine structure of an agent and their connections. The contained agent sets are represented by the ownedElements of a Collaboration:

- AssociationEnds for the agent sets, with *collaborationMultiplicity* representing the <number of instances> and the ClassifierRoles the types of the agent sets.
- AssociationRoles for the channels connecting the agent sets.

The notion of Interaction is not used. Interactions as parts of Collaborations are only meant as behaviour property models, and they correspond to the models described by MSC [3].

UML is somewhat unclear on the implications of Collaborations for the instances of the Classifier that is represented by the Collaboration (*representedClassifier*). This Recommendation defines that a Collaboration *defines* the content of instances of the represented Classifier.

3.6.1 AssociationEndRole

An AssociationEndRole may represent one of the following elements of an {<interaction area> | <agent body area> } ***set***:

- a contained agent set;
- the state machine of the containing agent;
- a gate of the containing agent.

UML	SDL
AssociationEndRole	The interface classifier associated with a gate of an agent set to which a channel is connected
.collaborationMultiplicity	• If a contained agent set: *Number-of-instances* • If the state machine of the agent: not applicable • If a gate of the containing agent: not applicable
->availableQualifier	The set of signals and remote procedures on the channel
->base	The interface qualifier represented by the interface of the gate to which the channel is connected
^AssociationEnd	
^GeneralizableElement	not applicable
^ModelElement	
.name	*Gate-name*

3.6.2 AssociationRole

AssociationRoles represent channels connecting agent sets, the state machine of the containing agent, and gates of the containing agent.

UML	SDL
AssociationRole	A *Channel-definition*
.multiplicity	The number of connections implied by the multiplicity of the AssociationEndRoles at both ends
->base	• the implicit association between the types of the agent sets, • the composite associations between the type of the enclosing agent and the types of the agent sets, or • the implicit associations between the composite state type of the state machine and the containing agent type and the types of the agent sets.
^Association	
^GeneralizableElement	The channels follows the generalization of the containing agent type, i.e. the connections of an agent type are inherited
^ModelElement	
.name	*Channel-names*

3.6.3 ClassifierRole

A ClassifierRole represents an interface of a gate of an agent set.

UML	SDL
ClassifierRole	the type of an agent set contained in an agent
.multiplicity	the same as the multiplicity of the corresponding AssociationEndRole, i.e. *Number-of-instances*
->availableContents	the signals and exported procedures that are available through the interface gate
->availableFeature	the signals and exported procedures that are available through the interface gate
->base	the type of the agent set
^Classifier	
^GeneralizableElement	the interface follows the generalization of the containing agent type, i.e. they are inherited
^ModelElement	
.name	agent set name

3.6.4 Collaboration

A Collaboration represents the internal agent/state machine structure of an agent and their connections. The properties defined by a Collaboration are properties of all instances of the class represented by *representedClassifier*.

The association *representedOperation* is not used, i.e. SDL entities that would be represented by Operations do not have their internal structure represented by Collaboration.

UML	SDL
Collaboration	{ <interaction area> | <agent body area> } }***set*** as part of <agent diagram content>
->constrainingElement	not applicable
->interaction	not applicable
->ownedElement	the elements of { <interaction area> | <agent body area> } }***set***
->representedClassifier	the agent type with the { <interaction area> | <agent body area> } }***set*** represented by this collaboration
->representedOperation	not applicable
^GeneralizableElement	follows the corresponding property of the container agent type, i.e. all the properties of an agent type specified by a Collaboration are inherited
^Namespace	the scope unit of the agent type with the { { <interaction area> | <agent body area> } }***set*** represented by this collaboration. While the ownedElement of a collaboration only represent the { { <interaction area> | <agent body area> } }***set***, the ownedElement of the Namespace of the agent type represent all entities in the agent type
^ModelElement	the attributes and associations from ModelElement are the same as for the class representing the agent type whose {<interaction area> | <agent body area>}}***set*** is represented by this Collaboration

3.6.5 Interaction

Not applicable.

3.6.6 Message

Not applicable.

3.7 Use Cases

Use Cases are not used in this Recommendation. They describe a system in a form, which is outside the scope of SDL. They may, however, still be used in combination with *SDL UML*.

3.8 State machines

3.8.1 CallEvent

A CallEvent represents the input of a remote procedure in SDL.

UML	SDL
CallEvent	input of remote procedure call
->operation	<remote procedure identifier>
^Event	
->parameters	<procedure formal parameters>

Note that in SDL an input of a remote procedure cannot be discarded, only deferred if not executed.

3.8.2 ChangeEvent

A ChangeEvent represents the input of a continuous signal in SDL.

UML	SDL
ChangeEvent	*Continuous-signal*
.changeExpression	*Continuous-expression*
^Event	
->parameters	not applicable

In SDL the *Continuous-signal* is only evaluated while no other stimulus are found in the input port.

3.8.3 CompositeState

A CompositeState represents an SDL composite state.

The fact that a composite state type can be a virtual type and a subtype is represented in the class with stereotype «state».

UML	SDL
CompositeState	*Composite-state-part*
->subvertex	*State-start-nodes*, *State-nodes* and *Return-nodes* of the *Composite-state-graph*
.isConcurrent = false = true	 Composite state that is not a *Multi-state* *Multi-state*
.isRegion = false = true	 Composite state as part of a *State-machine-definition* *State-partition*

3.8.4 Event

An Event has no counterpart in SDL; however, subclasses of Event have a mapping to SDL.

3.8.5 FinalState

A FinalState represents a return from an SDL composite state.

UML	SDL
FinalState	Unlabeled *Return-node* from composite state

3.8.6 Guard

A Guard represents an enabling condition in SDL.

UML	SDL
Guard	*Provided-expression*
.expression	*Boolean-expression*

Note that in UML, if a guard evaluates to false, the input event is discarded, whereas in SDL the input event is deferred (saved).

3.8.7 PseudoState

A PseudoState represents an abstraction that encompasses different types of transient nodes in a state machine graph.

UML	SDL
PseudoState	Nodes in a *State-machine-definition*
.kind = initial = deepHistory = shallowHistory = join = fork = junction = choice	 Unlabeled *State-start-node* of a composite state no concept no concept no concept no concept *Join-node* for merging, *Decision-node* for branching no concept

3.8.8 SignalEvent

A SignalEvent represents the input of a signal in SDL.

UML	SDL
SignalEvent	input of signal
->signal	instance of «signal» Class
^Event	
->parameters	signal parameters
.kind = in = inout = return	 Mandatory not applicable not applicable

3.8.9 SimpleState

A SimpleState represents a basic state in SDL.

UML	SDL
SimpleState	*State-node* without *Composite-state-part*

3.8.10 State

A State represents a combination of an SDL state and the contents of an SDL composite state. The entry- and exit procedures and internal transitions are part of the contents of the SDL composite state, while *deferrableEvent* corresponds to save on the state as such.

UML	SDL
State	*State-node* and contents of *Composite-state-part*
->deferrableEvent	<save part> associated with the *State-node*
->entry	*Entry-procedure-definition* (of the *Composite-state-part*)
->exit	*Exit-procedure-definition* (of the *Composite-state-part*)
->doActivity	no concept
->internalTransition	internal *Transitions* (of the *Composite-state-part*)

3.8.11 StateMachine

A StateMachine represents a state machine of an agent or composite state.

UML	SDL
StateMachine	*State-machine-definition*
->context	*Agent-type-definition* with the *State-machine-definition* or *Procedure-definition* with *Procedure-graph*
->top	the enclosing composite state of the *State-machine-definition*
->transition	the *State-transition-graph* of the *State-machine-definition*

3.8.12 StateVertex

A StateVertex represents an abstraction of a node in a state machine graph. Possible nodes in SDL are (in this context): *State-node*, *Nextstate-node*, *Start-node*, *Return-node*, *Stop-node*, *Decision-node* and <join>.

UML	SDL
StateVertex	*Transition*
->outgoing	*Transition*s leading from a node to transition *Terminators*
->incoming	*Transition*s having the node as transition *Terminator*
->container	the enclosing *State-transition-graph*

3.8.13 StubState

A StubState has no direct counterpart in SDL. The similar concept in SDL is state entry/exit connection point. It is regarded as essential in SDL to define the possible entry/exit points as part of the interface of a composite state, and not to allow entry into any substate.

3.8.14 SubmachineState

It is not obvious from the UML Semantics if a SubmachineState is class-based or not, even though reuse is mentioned. The closest correspondence is to a type-based composite state in SDL. Alternatively it corresponds to a composite state reference in SDL, and the fact that the same submachine state can be referenced more than once in the containing state machine supports this, but the macro-like expansion described in UML indicates that is not a reference.

3.8.15 SynchState

A SynchState has no counterpart in SDL.

3.8.16 TimeEvent

A TimeEvent represents the input of a timer in SDL.

UML	SDL
TimeEvent	input of timer
.when	*Time-expression*
^Event	
->parameters *.kind* = in = inout = return	*Variable-identifier**s in *Input-node* mandatory not applicable not applicable

3.8.17 Transition

A Transition represents a transition in SDL. The only difference is that a transition in SDL is associated with the *source*, while this is an association of *transition*.

UML	SDL
Transition	*Transition*
->trigger	*Input-node* or *Continuous-signal*
->guard	*Provided-expression*
->effect	*Graph-node** of *Transition*
->source	*State-node* to which the transition is associated
->target	*Terminator*

When *source* and *target* are located in different state machines, the *effect* will be executed in the state machine owning the *source*. When mapping to SDL, an *effect* crossing a state boundary going from an inner to an outer state machine will introduce a <state exit point> in the inner state machine. Contrary, an *effect* crossing a state boundary going from an outer to an inner state machine will introduce a <state entry point> in the inner state machine.

3.9 Activity Graphs

Activity Graphs are not used in this Recommendation.

3.10 Model Management

From the Model Management Package, only the Package and Model meta-class are used, not Subsystem. Model and Package are basic meta-model classes in UML. No special stereotype is defined, but the use shall be restricted as described in this subclause.

3.10.1 ElementImport

Not applicable.

3.10.2 Model

A Model represents an *SDL-specification*. This implies that a Model may contain only a set of packages and one system class.

NOTE – Possibly, also classes in the environment, nested types, interfaces and associations.

UML	SDL
Model	*SDL-specification*
^Package	
->*importedElement*	no concept
^GeneralizableElement	not applicable
^Namespace	scope unit
->*ownedElement*	*Package-Definition*-**set** [*Agent-definition*]
^ModelElement	
.name	no concept (but for further study).
->*constraint*	no concept
->*namespace*	not applicable – an SDL-specification is not defined in a scope unit
.template	not applicable
.templateParameter	not applicable
->*supplierDependency*	not applicable
->*clientDependency*	not applicable

Only ModelElements that map to contained entities in *SDL-specification* can be *ownedElements* of a Model.

3.10.3 Package

A Package represents an SDL *Package-definition*.

UML	SDL
Package	*Package-definition*
->*importedElement*	Definitions made visible to this package through <package use clause> and <definition selection list>
^GeneralizableElement	not applicable
^Namespace	Package scope unit
->*ownedElement*	Entities in the subset of <entity in package> defined below
^ModelElement	
.name	
->*constraint*	no concept
->*namespace*	the enclosing scope unit that contains the package, e.g. *SDL-specification* or *Package-definition*
.template	not applicable
.templateParameter	not applicable
->*supplierDependency*	for a package in a <package use clause> the fact that this package is used by a number of packages
->*clientDependency*	for a package with a <package use clause> the fact that this package uses other packages

The *namespace* must be Model or Package.

The *ownedElement* must be model elements that represent the following elements from <entity in package>:

- *Package-definition*;
- *Agent-type-definition*;
- *Signal-definition*;
- <remote variable definition>;
- *Data-type-definition*;
- *Procedure-definition*;
- *Composite-state-type-definition*; or
- *Interface-definition*.

An SDL package is completely defined by the *Package-definition*. A Package defines parts of the SDL package. The parts it defines depend upon the content of the Package. The choice of this content is left to the user, but they shall be consistent with the corresponding properties defined in the SDL *Package-definition*.

The <interface> of a <package heading> is not covered directly by this extension. Indirectly it can be represented by the visibility attribute of each Feature in the *ownedElement*s of the Package.

Dependencies between Packages represent <package reference clause>s (in <package reference area>). The <definition selection list> is not represented. *supplierDependency* for a package in a <package use clause> denotes the Dependencies that represent that this package is used by a number of packages. *ClientDependency* for a package with a <package use clause> denote the Dependencies that represent the use of other packages.

3.10.4 Subsystem

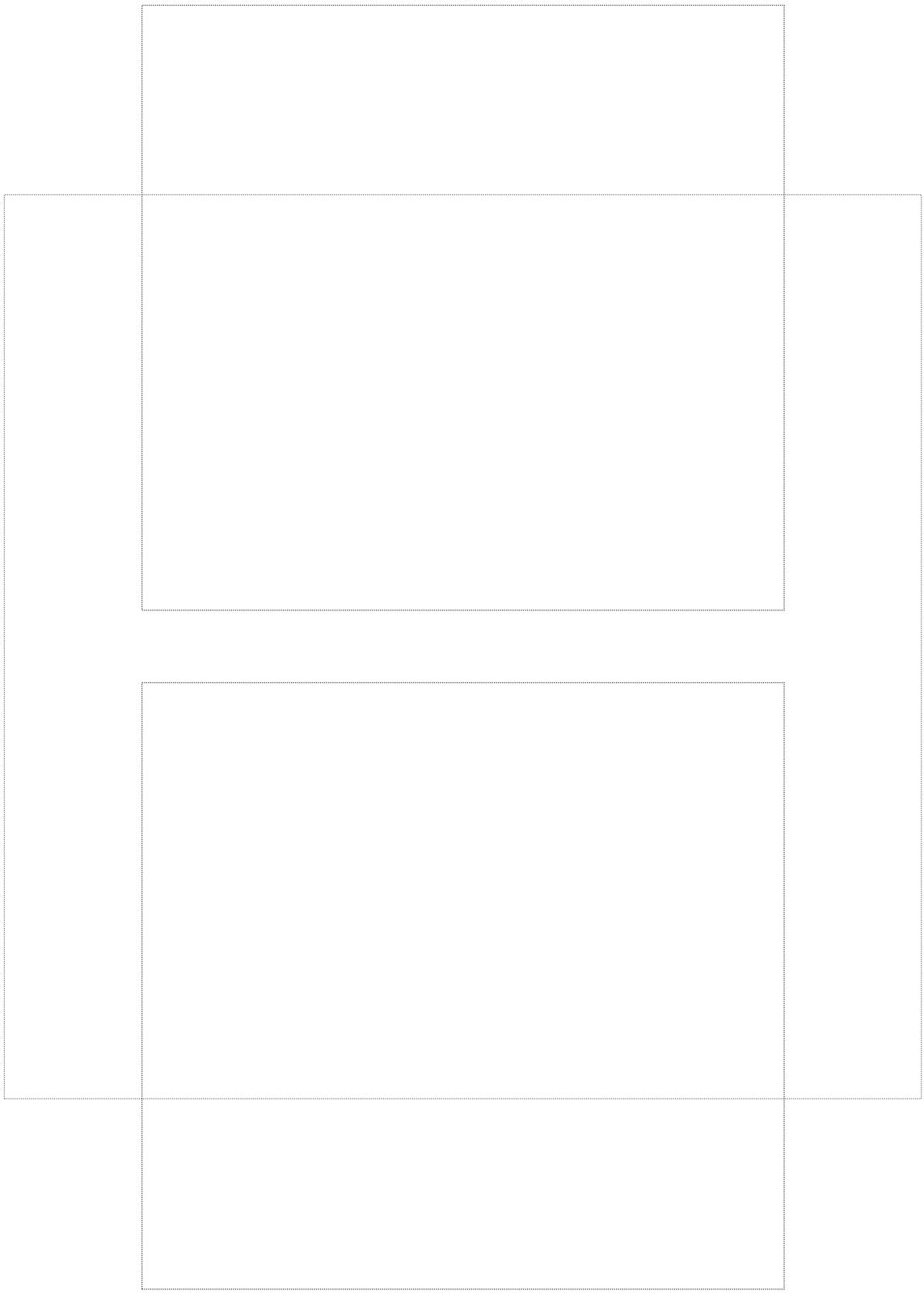
Not applicable.

APPENDIX I

Common Behaviour

The following is an overview of a possible mapping of the UML Common Behaviour package.

UML	SDL
Action	Action
ActionSequence	Transition string
Argument	Signal parameters of output
AssignmentAction	Assignment
AttributeLink	The binding of an identifier to the definition of the variable or field
CallAction	Remote procedure call or method invocation
ComponentInstance	Not applicable
CreateAction	Create request
DestroyAction	No concept
DataValue	Value
Exception	Exception instance
Instance	Instance/entity
Link	No concept
LinkEnd	No concept
LinkObject	No concept
NodeInstance	No concept
Object	Type-based entity
Reception	Input
ReturnAction	Return of a value returning procedure
SendAction	Output
Signal	Signal – See separate table in 3.5.2.
Stimulus	Consumption of a signal or execution of a remote procedure
TerminateAction	Stop
UninterpretedAction	Informal action



ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
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	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	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
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 and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2000