International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.109

(04/2012)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Specification and Description Language – Unified modeling language profile for SDL-2010

Recommendation ITU-T Z.109

International Telecommunication Union

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.109

Specification and Description Language – Unified modeling language profile for SDL-2010

Summary

Objective: Recommendation ITU-T Z.109 defines a unified modeling language (UML) profile that maps to SDL-2010 semantics so that UML is able to be used in combination with the ITU-T Specification and Description Language.

Coverage: This Recommendation presents a definition of the UML-to-SDL-2010 mapping for use in the combination of SDL-2010 and UML.

Application: The main area of application of this Recommendation is the specification of telecommunication systems. The combined use of SDL-2010 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-2010 mapping for use in the combination of SDL-2010 and UML. It replaces the previous Recommendation ITU-T Z.109 that concerned earlier versions of UML and Specification and Description Language.

Associated work: Recommendations ITU-T Z.100, ITU-T Z.101, ITU-T Z.102, ITU-T Z.103, ITU-T Z.104 and ITU-T Z.107 concerning the ITU-T Specification and Description Language 2010 (SDL-2010).

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T Z.109	1999-11-19	10
2.0	ITU-T Z.109	2007-06-13	17
3.0	ITU-T Z.109	2012-04-29	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/.

Table of Contents

Introduction

The UML profile presented in this Recommendation is intended to support the usage of UML (version 2 or later) as a front-end for tools supporting specification and implementation of reactive systems, in particular for telecommunication applications. The intention is to enable tool vendors to create tools that benefit from the closure of semantic variations in UML with SDL-2010 semantics and benefit from Specification and Description Language tool technology that supports this particular application area.

The intention is that when the profile is applied to a model, the set of stereotypes and metaclasses defined in this Recommendation extends the elements in the model and has several consequences:

– additional properties are available as specified by the stereotype attributes;

– constraints defined for the stereotypes apply to the model elements introducing more semantic checks that need to be fulfilled for the model;

– semantics, in particular dynamic semantics, are defined for the model elements as specified by the mapping of the stereotyped UML concepts to the SDL-2010 abstract grammar.

The details of the profile mechanism in this Recommendation follow: The Recommendation is structured in a number of clauses. Each clause defines one stereotype or metaclass (see below). Each stereotype usually captures the semantics of one SDL-2010 concept based on a UML concept. A stereotype in most cases constrains a UML element with a multiplicity of [1..1] (that is, the stereotype is required), but in some cases extends rather than constrains the basic UML language. The UML user never manually has to apply the stereotype to a UML element: instead stereotypes are applied automatically when applying the profile to the model itself, or if the user has not kept within the language defined by this profile a suitable message given to the user. As a consequence, applying this profile results in extra properties, extra semantic checks, and a well understood semantics that are usable in tools to provide features like static model analysis, simulation and application generation as the model is sufficiently well defined to be executable.

Apart from the set of stereotypes, the Recommendation defines a set of metaclasses as extensions to the UML metamodel in order to represent SDL-2010 expressions and value specifications. That is because the UML concepts for value specification are not appropriate for this purpose.

This Recommendation introduces no particular textual notation for stereotypes defined by this UML profile. Instead, a textual notation and its mapping to corresponding model elements has to be defined by an additional description (possibly a Recommendation or provided by a tool supplier). So that the application of transformation models of SDL-2010 referenced in this profile are understandable, the syntax for an appropriate textual notation should be a subset of the concrete syntax of SDL-2010 or an SDL-like syntax, which is modified to the particular requirements of a UML-based domain specific language.

The idea is that when a user enters the described syntax, a tool should automatically create the corresponding model element with the correct stereotype applied.

Recommendation ITU-T Z.109

Specification and Description Language – Unified modeling language profile for SDL-2010

1 Scope and objectives

This Recommendation defines a unified modeling language (UML) profile for SDL-2010. It ensures a well-defined mapping between parts of a UML model and the SDL-2010 semantics. The profile is based upon the UML metamodel and upon the abstract grammar of SDL-2010, and in the following text is referred to as SDL-UML.

The specializations and restrictions are defined in terms of stereotypes for metaclasses of the UML metamodel and the abstract grammar of SDL-2010 and are in principle independent of any notation. However, to generate particular model elements, especially those that are instances of UML actions or activities, it is assumed that an appropriate notation is specified.

A software tool that claims to support this Recommendation (in the following called a tool) should be capable of creating, editing, presenting and analysing descriptions compliant with this Recommendation.

1.1 Conformance

A model that claims to be compliant to this Recommendation shall meet the metamodel constraints of UML and this Recommendation and, when mapped to the abstract grammar of SDL-2010, shall conform to the abstract grammar of the ITU-T Z.100 series of Recommendations included by reference. A model is non-compliant if it does not meet the constraints or if it includes an abstract grammar that is not allowed by the ITU-T Z.100 series of Recommendations, or has analysable semantics that are shown to differ from these Recommendations.

The abstract grammar of this Recommendation is a profile of UML and a set of additional metaclasses, which are specializations of the UML ValueSpecification metaclass. Therefore, any model that conforms to this Recommendation also conforms to the requirements of UML.

A tool that supports the profile shall support the specializations and restrictions of UML defined in the profile to conform to the Recommendation and should be capable of exporting such models to other tools and importing such models from other tools.

A conformance statement clearly identifying the profile features and requirements not supported should accompany any tool that handles a subset of this Recommendation. If no conformance statement is provided, it shall be assumed that the tool is fully compliant. It is therefore preferable to supply a conformance statement; otherwise, any unsupported feature allows the tool to be rejected as not valid. While it is suggested that tools provide a suitable notation, conformance to any particular notation is not a requirement.

A **compliant tool** is a tool that is able to detect non-conformance of a model. If the tool handles a superset of SDL-UML, it is allowed to categorize non-conformance as a warning rather than a failure. It is required that for those 'Language Units' (see the UML specification [OMG UML] clause 2, Conformance) handled by the tool, a compliant tool conforms to the metamodel defined by this profile combined with the UML specification [OMG UML] and the mapping of those 'Language Units' to the SDL-2010 abstract grammar as defined by this Recommendation.

A **fully compliant tool** is a compliant tool that supports the complete profile defined by this Recommendation.

A **valid tool** is a compliant tool that supports a subset of the profile. A valid tool should include enough of the profile for useful modelling to be done. The subset shall enable the implementation of structured applications with communicating extended finite state machines.

1.2 Restrictions on SDL-2010 and UML

There are no restrictions on SDL-2010. However, SDL-2010 is only partially covered by SDL-UML.

A general restriction on SDL-UML is that only the metamodel elements defined in this profile ensure a one-to-one mapping. In a combined use of UML and SDL-2010, more parts of UML are usable, but the mapping of these cannot be guaranteed to work the same with different tools.

This profile focuses on the following clauses of the UML Superstructure specification:

– Classes;
– Composite structures;
– Common behaviours;
– Actions;
– Activities;
– State machines.

Metamodel elements defined in these clauses are included in this profile if they are specifically mentioned in this Recommendation. Any metamodel element of the UML Superstructure specification that is not mentioned in this Recommendation is not included in this profile. A metamodel element that is a generalization of one of the included metamodel elements (that is, it is inherited) is included as part of the definition of the included metamodel element. Other specializations of such a generalization are only included if they are also specifically mentioned. If an included metamodel element has a property that is allowed to be non-empty, the metamodel element for the property is included. However, if the property is constrained so that it is always empty, such a property is effectively deleted from the model and therefore does not imply the metamodel element for the property is included.

Metamodel elements introduced in the following clauses of the UML Superstructure specification are not included in this profile:

– Components;
– Deployments;
– Use cases;
– Interactions;
– Auxiliary constructs;
– Profiles.

1.3 Mapping

UML classes generally represent entity types of SDL-2010. In most cases, the entity kind is represented by a stereotype. Where predefined model-elements or stereotypes or notation exist in UML that have a similar meaning as in SDL-2010, they have been used.

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.103] Recommendation ITU-T Z.103 (2011), *Specification and Description Language – Shorthand notation and annotation in 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.107] Recommendation ITU-T Z.107 (2012), *Specification and Description Language – Object-oriented data in SDL-2010*.

[ITU-T Z.119] Recommendation ITU-T Z.119 (2007), *Guidelines for UML profile design*.

[OMG UML] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure*. Version 2.4.1, document no. formal/2011-08-06.
<http://www.omg.org/spec/UML/2.4.1/Superstructure>

NOTE – This Recommendation references specific paragraphs of [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104], [ITU-T Z.107] and [OMG UML]. The specific paragraph references are only valid for the editions specifically referenced above. If a more recent edition of [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] and [ITU-T Z.107] or [OMG UML] is used, it is possible that the corresponding paragraph number or paragraph heading is different.

3 Definitions

3.1 Terms defined elsewhere

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

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms, which apply if they are also defined elsewhere:

3.2.1 compliant tool: A tool that is able to detect non-conformance of a model to the profile defined by this Recommendation.

3.2.2 direct container: A is the direct container of B (B is directly contained in A; A directly contains B), if A contains B and there is no intermediate C that contains B such that C is contained in A.

3.2.3 fully compliant tool: A compliant tool that supports the complete profile defined by this Recommendation.

3.2.4 type conformance: The UML type conformance (applied by "conforms to") is as defined in clause 7.3.8 Classifier of [UML-SS], and corresponds to SDL-2010 sort compatibility as defined in clause 12.1.9 of [ITU-T Z.104].

3.2.5 valid tool: A compliant tool that supports a subset of the profile defined by this Recommendation where the subset enables the definition of models containing structured applications with communicating extended finite state machines.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

SDL-2010 Specification and Description Language 2010, particularly as it relates to the relevant ITU-T Z.100 series Recommendations for the term.

SDL-UML The language defined by the UML profile in this Recommendation.

UML Unified Modeling Language 2.0 (see [OMG UML]).

UML-SS OMG UML-2.4 Superstructure Specification (see [OMG UML]).

5 Conventions and names

This clause defines conventions used throughout the rest of this Recommendation and the general handling of name resolution and template expansion that apply for the whole metamodel.

5.1 Conventions

The conventions defined in [ITU-T Z.119] apply. For convenience, these conventions are repeated below.

A term in this Recommendation is a sequence of printing characters usually being either an English word or a concatenation of English words that indicate the meaning of the term.

A term preceded by the word "stereotype" names a UML stereotype used for this profile, according to the stereotype concept defined in the UML Superstructure specification documentation (usually in a phrase "The stereotype X extends the metaclass X" where X is a term). If the multiplicity of the stereotype is [1..1], the stereotype is required (that is, the derived attribute isRequired of the Extension association between the extended metaclass and the stereotype is true). If the multiplicity of the stereotype is [0..1], the stereotype is not required.

New metaclasses are also introduced in this Recommendation (usually by a phrase such as "The abstract metaclass SdlExpression is a specialization of the UML metaclass ValueSpecification").

Some stereotypes and metaclasses are introduced only to define common elements shared between different metaclasses based on them and an instance of the base stereotype or metaclass is not allowed: in UML terminology the stereotype or metaclass is abstract and this is stated in the definition of the stereotype or metaclass.

An underlined term refers to a UML term or a term defined by a stereotype of this profile. A term starting with a capital letter by convention is the name of a metaclass.

A term is not underlined at the point at which it is introduced (for example, "X" in "The stereotype X extends the metaclass X", or SdlExpression in the phrase given above). Also in an attribute definition, neither the name or kind of the attribute is underlined, because the name is a defining occurrence and use of the kind as a term is obvious from context.

If a stereotype is required and has the same name as the metaclass it extends, the underlined term refers to both the metaclass and the stereotype. For example, "The visibility of the NamedElement shall not be **package**" means the same as the constraint: "The visibility of the <<NamedElement>> NamedElement shall not be **package**".

A term in italic in a stereotype or metaclass description refers to an SDL-2010 abstract syntax item.

A term in Courier font refers to some text that appears in the model either as written by a user or to represent some text created from the expansion of a shorthand notation (as outlined in clause 5.3, Transformation, below and in detail for the relevant construct).

The terms "supertype" and "subtype" are widely used in this Recommendation, SDL-2010 Recommendations and UML-SS and it is assumed that they are well understood. When the term

"supertype" is used in relation to the metamodel in this Recommendation, for Classifier (and metaclasses or stereotypes derived from Classifier) supertype corresponds to the general property of the Classifier. For a Class (and metaclasses or stereotypes derived from Class) supertype corresponds to the superClass property of the Class (which redefines general from Classifier). The term "subtype" is the inverse of "supertype": if A is a supertype of B, B is a subtype of A.

The metamodel diagrams in this Recommendation are informative overviews rather than normative.

5.1.1 References

UML-SS [**OMG UML**]:

6.3 The UML Metamodel

18.3.9 Stereotype (from Profiles)

5.2 Names and name resolution: NamedElement

The stereotype NamedElement extends the metaclass NamedElement with multiplicity [1..1].

NOTE – Names are resolved according to the UML name binding rules. However, there are constraints applied to names that are mapped to the SDL-2010 abstract syntax.

5.2.1 Attributes

No additional attributes are defined.

5.2.2 Constraints

[1] Any item that inherits from NamedElement and maps to SDL-2010 abstract syntax requiring a *Name* shall have a name. Any such name shall have a non-empty `String` value of characters derived from the syntax as defined in the Notation clause below.

[2] When a complete SDL-UML model maps to the SDL-2010 abstract syntax, no item shall have the same *Name* as another item of the same entity kind in the same defining context.

NOTE – It is always possible to modify a UML model to meet the above naming requirement by renaming elements that generate name clashes so that the UML model is a valid SDL-UML model for this profile.

[3] A NamedElement shall have a visibility and qualifiedName.

[4] The visibility of the NamedElement shall not be **package**.

[5] The visibility of the NamedElement (or of any item derived from it) shall be **protected** or **private** only if the NamedElement is an operation (including a literal) of a data type.

5.2.3 Semantics

The characters of the `String` for a name are each of the characters of the <name> taken in order.

Whenever a *Name* is required in the SDL-2010 abstract syntax (usually for the definition of an item), the *Name* is mapped from the name of the appropriate item derived from NamedElement. Whenever an *Identifier* is required in the SDL-2010 abstract syntax (usually to identify to a defined item), the *Identifier* is mapped from the name of the appropriate item derived from NamedElement. The detail of these mappings is described in the following paragraphs.

When a name maps to a *Name*, the `String` value of the name maps to the *Token* and if two items have a distinct `String` value each item maps a different *Token*. If two items have the same *Token* for their *Name*, they have the same `String` value for their name. If two items have the same `String` value for their name, they have the same *Token* for their *Name*, except if two UML elements are distinguishable by some additional means (such as distinct signatures of operations with the same name and same type in the same namespace). In such exceptional cases, each name maps to a different unique *Token*.

When the SDL-2010 abstract syntax requires an *Identifier*, the `String` value of the qualifiedName is used. A qualifiedName is a derived attribute that allows the NamedElement to be identified in a hierarchy. The *Qualifier* of the *Identifier* is a *Path-item* list that specifies uniquely the defining context of the identified entity and is derived from the qualifiedName. Starting at the root of the hierarchy, each name and class pair of the containing namespaces maps to the corresponding qualifier (*Package-qualifier*, *Agent-qualifier*, etc.) and name (*Package-name*, *Agent-name*, etc. respectively) pair. This mapping excludes the name of the NamedElement itself, which maps to the *Name* of the *Identifier*.

NOTE 1 – In SDL-2010 the *Qualifier* is usually derived by name resolution and context, and *Identifier* is usually represented in the concrete syntax by an SDL-2010 <name> and the SDL-2010 qualifier part of an SDL-2010 <identifier> is omitted. Even in cases where an SDL-2010 qualifier needs to be given, usually some parts of the SDL-2010 qualifier are optional, so that a full context does not have to be given. Similarly in UML, qualifiedName is usually derived, and is not given explicitly in the concrete syntax. Thus in both UML and SDL-2010 an item is usually identified in the concrete syntax simply by a name, whereas in the metamodel and SDL-2010 abstract syntax the item will be identifed by a qualifiedName and *Identifier* respectively.

NOTE 2 – The visibility of a Package contained in another Package or a Class or other entity contained in a Package is handled by name resolution.

5.2.4 Notation

<name> ::=
<underline>+ <word> {<underline>+ <word>}* <underline>*
| <word> <underline>+ [<word>{<underline>+ <word>}* <underline>*]
| <decimal digit>* <letter> <alphanumeric>*

NOTE – The syntax given for <name> assumes a one-to-one mapping between a <name> and an SDL-2010 <name> that has the same *Token*. The characters normally allowed in an SDL-2010 <name> are defined by Recommendation [b-ITU-T T.50]: uppercase letters A (Latin capital letter A) to Z (Latin capital letter Z); lowercase letters a (Latin small letter a) to z (Latin small letter z); decimal digits 0 (Digit zero) to 9 (Digit nine) and underline. The above syntax for <name> requires a name to include at least one underline (first 2 alternatives of <name>) or at least one <letter>. The ITU-T T.50 characters do not occur in the abstract grammar, therefore for alphabets and characters other than the Latin alphabet in Recommendation [b-ITU-T T.50] there just has to be a consistent mapping of name in an extended alphabet to a *Name*. Because the notation is a guideline and not mandatory, it is permitted to extend the syntax of <name> for this case.

<word> ::=
<alphanumeric>+

<alphanumeric> ::=
<letter>
| <decimal digit>

<letter> ::=
<uppercase letter> | <lowercase letter>

<uppercase letter> ::=
A | B | C | D | E | F | G | H | I | J | K | L | M
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<lowercase letter> ::=
a | b | c | d | e | f | g | h | i | j | k | l | m
| n | o | p | q | r | s | t | u | v | w | x | y | z

<decimal digit> ::=
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

When a <name> occurs in syntax that defines a name, the qualifiedName is derived from the defining context. Otherwise, a name shall be bound according to the UML name binding rules and if necessary the name is qualified by containing namespaces.

It is suggested to use the SDL-2010 syntax for <identifier> in [ITU-T Z.101] for specifying optionally qualified names.

An alternative suggestion is to use the following UML-like syntax for <identifier> for specifying optionally qualified names.

<identifier> ::=
[<containing namespaces>] <name>

<containing namespaces> ::=
[<name separator>] { <name> <name separator> }+

<name separator> ::=
<colon> <colon>

<colon> ::= :

In this case, if the <name> of an <identifier> is not unique and is ambiguous in the context where the <identifier> occurs, it is disambiguated by adding a <containing namespaces> item that contains one or more <name> elements. In the absence of an initial <name separator>, the right-most <name> elements in the <containing namespaces> have to unambiguously identify a context where the <name> of the <identifier> is defined. If the context is not identified unambiguously by the right-most <name> elements in the <containing namespaces>, further <name> elements are added until the context is unambiguous. If the initial <name separator> is given, the left-most name is a name defined at the top level of the model.

5.2.5 References

SDL-2010 [ITU-T Z.101]:

6.1 Lexical rules

6.6 Names and identifiers, name resolution and visibility

UML-SS [OMG UML]:

7.3.34 NamedElement (from Kernel, Dependencies)

7.3.44 PrimitiveType (from Kernel)

5.3 Transformation

The SDL-2010 abstract syntax of a model is generated from a concrete grammar (as defined outside the scope of this Recommendation) of an SDL-UML model by the following process.

The model is parsed according to the concrete grammar defined for SDL-UML. Where the concrete grammar defines shorthand notations, these are expanded during the parsing process before the corresponding metamodel items are generated.

NOTE – The transformations that are applied to expand shorthand notations of the concrete grammar are intended to be the same as the models defined for the corresponding shorthand notation in SDL-2010. For example, an SDL-2010 remote procedure call is expanded into an exchange of implicit signals, and an SDL-UML remote operation call is similarly expanded into an exchange of signals.

To determine whether a model written in a concrete grammar is valid requires all uses of names to be resolved, but names are resolved according to the SDL-UML metamodel. It is, therefore, not possible to parse the model as represented in the concrete grammar independently of generating the metamodel.

Apart from name resolution, instances of metamodel elements are generated from the concrete grammar of an SDL-UML model according to the relationship between the concrete grammar and the metamodel. If the resultant model (expressed in terms of instances of metamodel elements) does not conform to the abstract grammar of SDL-UML, that model is not valid.

Conformance to the rules of the abstract grammar of SDL-UML is a necessary (but not sufficient) condition for an SDL-UML model to be a valid model.

The model expressed in terms of instances of SDL-UML metamodel elements maps to a model expressed in the abstract grammar of SDL-2010. The behaviour of this resultant model is determined by the semantics of SDL-2010. Any static semantic constraints of SDL-2010 are reflected in constraints of the SDL-UML metamodel. To obtain the dynamic behaviour of the resultant model, this model is interpreted according to the dynamic semantics of SDL-2010. The model is not valid if violation of a dynamic constraint of SDL-2010 is possible during interpretation of the model expressed in the abstract grammar of SDL-2010.

6 Summary of stereotypes and metaclasses

6.1 Stereotype summary

The following table gives a summary of the stereotypes defined in this profile with the UML metaclass each stereotype extends and if the stereotype is abstract.

Stereotype	Stereotyped metaclass	Stereotype abstract
ActiveClass	Class	
Activity	Activity	
ActivityFinalNode	ActivityFinalNode	abstract
AddStructuralFeatureValueAction	AddStructuralFeatureValueAction	
AddVariableValueAction	AddVariableValueAction	
Break	OpaqueAction	
CallOperationAction	CallOperationAction	
ChoiceType	Class	
Classifier	Classifier	
ConditionalNode	ConditionalNode	
Connector	Connector	
Continue	OpaqueAction	
CreateObjectAction	CreateObjectAction	
DataTypeDefinition	Class	
ExpressionAction	ValueSpecificationAction	
FinalState	FinalState	
Interface	Interface	
LiteralType	Class	
LoopNode	LoopNode	
OpaqueAction	OpaqueAction	abstract
Operation	Operation	
Package	Package	
Parameter	Parameter	
Port	Port	
Property	Property	
Pseudostate	Pseudostate	

Stereotype	Stereotyped metaclass	Stereotype abstract
Region	Region	
ResetAction	SendSignalAction	
Return	ActivityFinalNode	
SendSignalAction	SendSignalAction	
SequenceNode	SequenceNode	
SetAction	SendSignalAction	
Signal	Signal	
Specification	Model	
State	State	
StateMachine	StateMachine	
Stop	ActivityFinalNode	
StructureType	Class	
Syntype	Class	
Timer	Signal	
Transition	Transition	
TimerConstraint	OpaqueExpression	
Variable	Variable	

6.2 Metaclass summary

The following tables give a summary of metaclasses defined in this profile for representing SDL-2010 expressions and context parameters. In general, the introduced metaclasses are specializations of the UML metaclass ValueSpecification (see clause 7.3.55 of [OMG UML]) or of the metaclass Element (see clause 7.3.14 of [OMG UML]). For the metamodel diagrams, semantics and associated constraints of metaclasses to represent SDL-2010 expressions see clause 10. For the metamodel diagrams, semantics and associated constraints of metaclasses to represent context parameters see clause 11.

If an introduced metaclass is a direct subtype of the metaclass ValueSpecification or the metaclass Element, this is indicated in the second column of the table. The third column indicates if the metaclass is for SDL-2010 expressions or if it is for context parameters. The fourth column indicates if the metaclass is abstract.

Metaclass	Specialized UML metaclass	Represents	Abstract metaclass
ActiveAgentsExpression	–	SDL-2010 expressions	
ActualContextParameter	Element	Context parameters	
AgentContextParameter	–	Context parameters	
AgentTypeContextParameter	–	Context parameters	
AnyExpression	–	SDL-2010 expressions	
ClosedRange	–	SDL-2010 expressions	

Metaclass	Specialized UML metaclass	Represents	Abstract metaclass
CompositeStateTypeContextParameter	_	Context parameters	
ConditionalExpression	_	SDL-2010 expressions	
ConditionItem	ValueSpecification	SDL-2010 expressions	abstract
EqualityExpression	–	SDL-2010 expressions	
FormalContextParameter	Element	Context parameters	abstract
GateConstraint	Element	Context parameters	
GateContextParameter	–	Context parameters	
ImperativeExpression	_	SDL-2010 expressions	abstract
InterfaceContextParameter	_	Context parameters	
LiteralValue	_	SDL-2010 expressions	
NowExpression	_	SDL-2010 expressions	
OpenRange	_	SDL-2010 expressions	
OperationApplication	_	SDL-2010 expressions	
PidExpression	_	SDL-2010 expressions	
ProcedureContextParameter	_	Context parameters	
RangeCheckExpression	_	SDL-2010 expressions	
RangeCondition	ValueSpecification	SDL-2010 expressions	
SdlExpression	ValueSpecification	SDL-2010 expressions	abstract
SignalContextParameter	_	Context parameters	
SizeConstraint	_	SDL-2010 expressions	
SortContextParameter	_	Context parameters	
StateExpression	_	SDL-2010 expressions	
SynonymContextParameter	_	Context parameters	
TimerActiveExpression	_	SDL-2010 expressions	
TimerContextParameter	_	Context parameters	
TimerRemainingDuration	_	SDL-2010 expressions	

Metaclass	Specialized UML metaclass	Represents	Abstract metaclass
TypeCheckExpression	–	SDL-2010 expressions	
TypeCoercion	–	SDL-2010 expressions	
ValueReturningCallNode	–	SDL-2010 expressions	
VariableAccess	–	SDL-2010 expressions	
VariableContextParameter	–	Context parameters	

7 Structure

The stereotypes below define static structural aspects of an SDL-UML model.

The following packages from UML are included:

– Communications
– Constructs (from Infrastructure library)
– Dependencies
– Interfaces
– InternalStructures
– Models
– Kernel
– Ports.

The following metaclasses from UML are included:

– Class
– Connector
– Interface
– Model
– Operation
– Package
– Parameter
– Port
– Property
– Signal.

The metaclass ValueSpecification is included in clause 10.

7.1 Structure metamodel diagrams

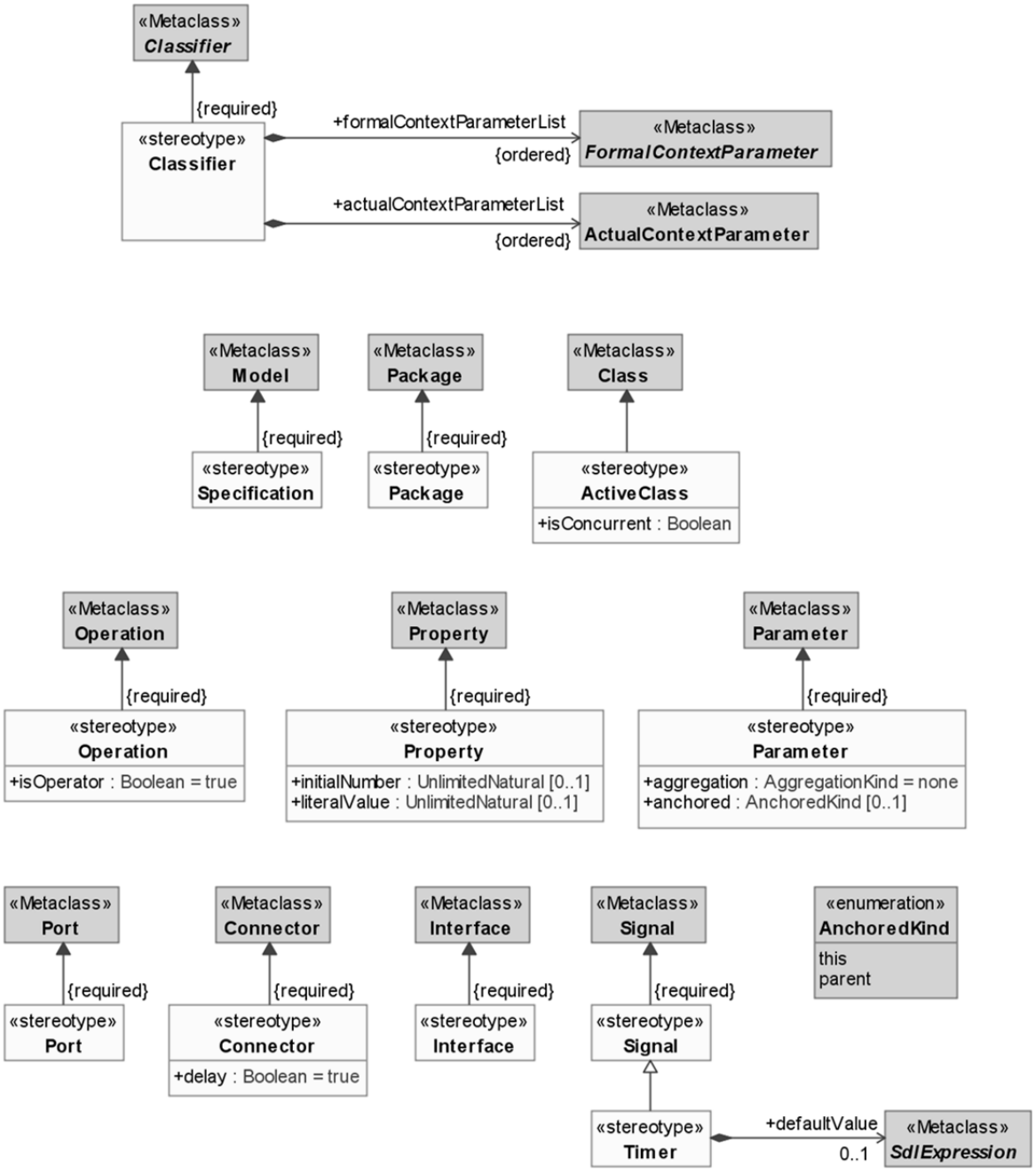

Figure 7-1 – Structure stereotypes

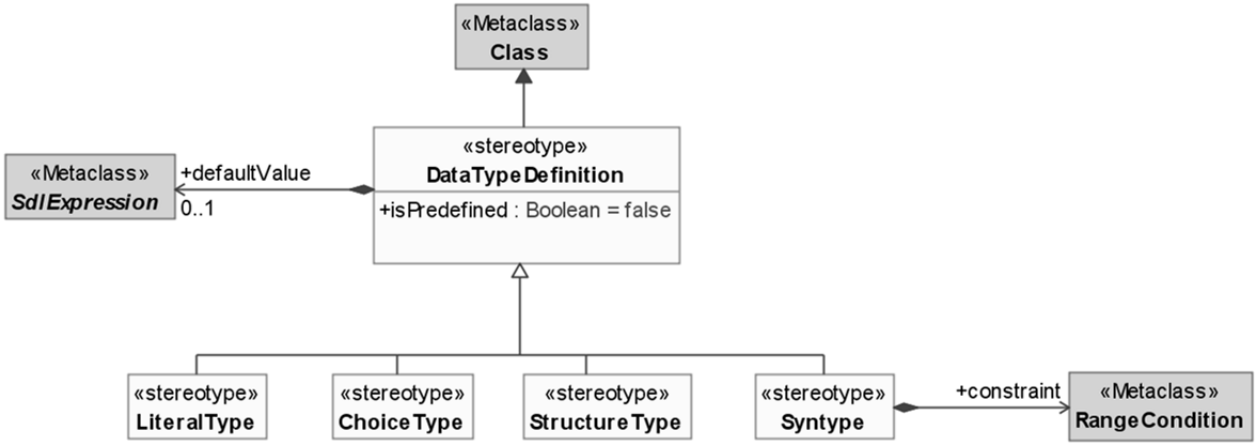

Figure 7-2 – Data type stereotypes

7.2 ActiveClass

The stereotype ActiveClass extends the metaclass Class with multiplicity [0..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

The concept of an active class (a class with isActive true) is separated from a data type definition (a Class with isActive false) to distinguish the classes for executable agents that map onto SDL-2010 agent types.

Specialization: A specializing active class `A2` is able to add attributes, port, operations, behaviour specifications and nested classifiers to those inherited from its supertype `A1` (see clause 8.4.1 in [ITU-T Z.102]).

Redefinition: If a classifier `C2` specializes a more general supertype `C1`, an enclosed active class EA of `C2` is able to redefine an active class EA that is specified in `C1`.

NOTE – The features of specialization and redefinition are introduced by the metaclass Classifier. For the common constraints and semantics see clause 7.4.

7.2.1 Attributes

- isConcurrent: Boolean

  defines the concurrency semantics of an active class. If isConcurrent is false, all contained instances execute interleaved. If isConcurrent is true, contained instances execute concurrently, provided they are not also contained in an instance for which isConcurrent is false.

7.2.2 Constraints

[1] An <<ActiveClass>> Class shall have isActive true.

[2] The clientDependency shall not include an InterfaceRealization, because interfaces are not realized directly but only via ports.

[3] If isConcurrent is false, the isConcurrent property of any contained instance shall be false.

[4] If the <<ActiveClass>> Class has a classifierBehavior, it shall be a StateMachine.

[5] If an <<ActiveClass>> Class has a classifierBehavior and it has a superClass that also has a classifierBehavior, the StateMachine of the subclass shall redefine the StateMachine of the superClass.

NOTE 1 – The reason is that in SDL-2010 the state machines of agents automatically extend each other, whereas they do not in UML.

[6] An ownedAttribute that has a type that is an <<ActiveClass>> Class and where aggregationKind composite shall not have public visibility.

NOTE 2 – An agent instance set cannot be made visible outside the enclosing agent type.

[7] A nestedClassifier shall not have public visibility.

NOTE 3 – An agent type, data type, interface type or signal definition cannot be made visible outside the enclosing agent type.

[8] An ownedConnector shall not have public visibility.

NOTE 4 – A channel cannot be made visible outside the enclosing agent type that owns the channel.

[9] An ownedPort shall have public visibility.

NOTE 5 – Gates are visible outside the enclosing agent type.

[10] An ownedBehavior shall not have public visibility.

NOTE 6 – A procedure or composite state type cannot be made visible outside the enclosing agent type.

[11] An ownedBehavior shall be a set of StateMachine items (one or more).

The following constraints shall apply, when the isConcurrent property of an <<ActiveClass>> Class is true and the owner is a <<Specification>> Model (indicating a system agent type):

[12] There shall be at least one ownedAttribute that has a type that is an <<ActiveClass>> Class or the classifierBehavior shall not be empty.

[13] The superClass property shall be empty.

[14] The redefinedClassifier shall be empty.

[15] If present, the formalContextParameterList shall not contain items that are of kind AgentContextParameter, VariableContextParameter or TimerContextParameter.

7.2.3 Semantics

An <<ActiveClass>> Class maps to an *Agent-type-definition*.

The name of the <<ActiveClass>> Class maps to the *Agent-type-name* of the *Agent-type-definition*.

The isConcurrent attribute maps to the *Agent-kind* of the *Agent-type-definition*. If isConcurrent is true and the owner is a <<Specification>> Model, the <<ActiveClass>> Class maps to an *Agent-type-definition* with an *Agent-kind* **SYSTEM**. If isConcurrent is true and the owner is not a <<Specification>> Model, the *Agent-kind* is a *BLOCK*; otherwise (isConcurrent false) the *Agent-kind* is a *PROCESS*.

NOTE 1 – The concurrency behaviour is that state machines within a *PROCESS* instance (for the instance itself and contained *PROCESS* instances) are interleaved, and agent instances directly contained within a *BLOCK* (even multiple instances of the same *PROCESS*) are logically concurrent. Actual concurrency depends on implementation constraints such as the number of execution engines.

If the isAbstract property is true, the optional *Abstract* node in the abstract syntax of an *Agent-type-definition* is present.

The qualifiedName of the optional general property maps to the *Agent-type-identifier* of the *Agent-type-definition* that represents inheritance in the SDL-2010 abstract syntax.

If the redefinedClassifier property is not empty, this is an implicit generalization of another <<ActiveClass>> Class. In this case, the qualifiedName of the redefinedClassifier maps to the *Agent-type-identifier* of the *Agent-type-definition*.

The nestedClassifier, ownedAttribute, ownedConnector, ownedPort and ownedBehavior associations map to the rest of the contents of the *Agent-type-definition* as described below.

Mappings of nested classifiers

A nestedClassifier that is an <<ActiveClass>> Class maps to an element of the *Agent-type-definition-**set*** of the *Agent-type-definition*.

A nestedClassifier that is a <<DataTypeDefinition>> Class maps to a *Value-data-type-definition* that is an element of the *Data-type-definition-**set*** of the *Agent-type-definition*.

A nestedClassifier that is an Interface maps to an *Interface-type-definition* that is an element of the *Data-type-definition-**set*** of the *Agent-type-definition*.

A nestedClassifier that is a Signal maps to a *Signal-definition* that is an element of the *Signal-definition-**set*** of the *Agent-type-definition*.

Mappings of owned attributes

An ownedAttribute is a Property. The mapping defined in clause 7.13, applies.

An ownedAttribute that maps to a *Variable-definition* (see clause 7.13) is an element of the *Variable-definition-**set*** of the *Agent-type-definition*. An ownedAttribute that is visible outside the <<ActiveClass>> Class (public visibility) and that has a type that is a <<DataTypeDefinition>> Class or <<Interface>> Interface is the *Variable-definition* for an exported variable and also maps to an implicit *Signal-definition* pair for accessing this exported variable in the defining context of the *Agent-type-definition*.

An ownedAttribute that maps to an *Agent-definition* (see clause 7.13) is an element of the *Agent-definition-**set*** of the *Agent-type-definition*.

Mappings of connectors and ports

Each Connector of the ownedConnector maps to an element of the *Channel-definition-**set*** of the *Agent-type-definition*.

Each Port of the ownedPort maps to an element of the *Gate-definition-**set*** of the *Agent-type-definition*.

Mappings of ownedBehavior

Each Behavior of the ownedBehavior maps to an element of either the *Composite-state-type-definition-**set*** or the *Procedure-definition-**set***. If the owned Behavior is the method of an Operation, it is an element of the *Procedure-definition-**set***; otherwise it is an element of the *Composite-state-type-definition-**set***.

The StateMachine that is the Behavior of the optional classifierBehavior maps to the *State-machine-definition* of the *Agent-type-definition* (see clause 8.6). The name of the optional classifierBehavior maps to the *State-name* of the *State-machine-definition*. The *Composite-state-type-identifier* of this *State-machine-definition* identifies the *Composite-state-type* derived from the StateMachine that is the classifierBehavior.

NOTE 2 – The UML StateMachine maps to the behaviour of an SDL-2010 composite state type, and the *State-machine-definition* references this behaviour.

The ownedParameter set of the <<StateMachine>> StateMachine that is the classifierBehavior maps to the *Agent-formal-parameter* list of the *Agent-type-definition*. The specific mappings are defined in clause 7.11.

NOTE 3 – It is a semantic variation in UML-SS whether one or more behaviours are triggered when an event satisfies multiple outstanding triggers.

NOTE 4 – It is currently not allowed to give actual parameter value to a formal parameter of an agent (see clause 9.10).

An event satisfies only one trigger (a signal initiates only one input transition).

NOTE 5 – In UML-SS, ordering of the events in the input pool and therefore the selection of the next event to be considered is a semantic variation.

At any specific wait point (that is, in a specific state), events for a trigger of higher priority are considered before those of triggers of lower priority. Within a given trigger priority, the events in the input pool are considered in the order of arrival in the input pool; therefore if all triggers have the same priority, the events are considered in order of arrival. If an event in the input pool of events satisfies no triggers at a wait point, it is left in the input pool if it is deferred at that wait point, or (if it is not deferred) it is consumed triggering an empty transition leading to the same wait point.

7.2.4 References

SDL-2010 [ITU-T Z.102]:

8.1.1 Structural type definitions

8.1.3 Abstract type

8.2 Type references and operation references

8.4 Specialization

UML-SS [OMG UML]:

7.3.6 BehavioredClassifier (from Interfaces)

7.3.7 Class (from Kernel)

9.3.1 Class (from StructuredClasses)

9.3.8 EncapsulatedClassifier (from Ports)

13.3.2 Behavior (from BasicBehaviors)

13.3.4 BehavioredClassifier (from BasicBehaviors, Communications)

13.3.8 Class (from Communications)

7.3 ChoiceType

The ChoiceType stereotype is a subtype of the <<DataTypeDefinition>> Class. The metamodel diagram for the stereotype is defined in Figure 7-2.

The <<ChoiceType>> Class corresponds to an SDL-2010 choice data type and it maps to a *Value-data-type-definition*. A choice data type comprises a set of different data types, but only one of those types is used as the actual type for a value for any given assignment or subsequent access. In SDL-UML, the ownedAttribute items of a <<ChoiceType>> Class represent the different variants of a choice type.

Specialization and redefinition for choice types is not supported in SDL-2010.

7.3.1 Attributes

No additional attributes.

7.3.2 Constraints

[1] The ownedAttribute shall not be empty.

[2] An ownedAttribute shall have a type that is a <<DataTypeDefinition>> Class (or one of its subtypes) or <<Interface>> Interface.

[3] A Property that is an ownedAttribute item shall have a multiplicity of [0..1].

[4] The general and redefinedClassifer properties shall be empty.

7.3.3 Semantics

A <<ChoiceType>> Class represents a choice data type of the concrete grammar of SDL-2010 and it maps to a *Value-data-type-definition*. Before the mapping is carried out, the transformation as specified in clause 12.1.6.3 of [ITU-T Z.101] shall be applied.

In addition, the mappings specified in the context of the <<DataTypeDefinition>> Class (clause 7.6) apply.

7.3.4 References

SDL-2010 [ITU-T Z.101]:

12.1 Data definitions

12.1.1 Data type definition

12.1.6.3 Choice data types

UML-SS [OMG UML]:

13.3.8 Class (from Communications)

7.4 Classifier

The stereotype Classifier extends the metaclass Classifier with multiplicity [1..1]. The metamodel diagram is defined in Figure 7-1.

A <<Classifier>> Classifier represents the SDL-2010 concepts for specialization and redefinition of type definitions. In addition, this stereotype introduces support for SDL-2010 context parameters, which are used instead of UML templates in order to specify generic type definitions (see clause 11).

Hence, the <<Classifier>> Classifier defines a common set of constraints, which also apply to metaclasses that inherit from the Classifier metaclass. In particular, the following metaclasses, which are relevant for SDL-UML, directly or indirectly inherit from Classifier:

- Class
- Signal
- Interface
- StateMachine
- Activity.

In general, each stereotype that extends one of the metaclasses listed above defines the specific semantics for specialization and redefinition. The common mechanisms of both concepts are described in the following paragraphs.

Specialization: A Classifier `C2` that specializes another Classifier `C1` is able to add particular kinds of features to those inherited from its superClass `C1` (see clause 8.4.1 in [ITU-T Z.102]). The kinds of features that it is possible to add to a Classifier depend on the stereotype applied to a specific Classifier instance. Hence, the semantics is defined in the scope of the relevant stereotypes.

NOTE 1 – The SDL-2010 concept of renaming is not supported in SDL-UML.

Redefinition: If a Classifier `C2` specializes a more general superClass `C1`, an enclosed classifier `EC` of `C2` is able to redefine (see clause 7.3.47 [OMG UML]) the Classifier `EC`, which is specified in `C1`. In SDL-2010, this corresponds to the redefinition of virtual types (see clause 8.4.2 in [ITU-T Z.102]). The redefined Classifier `EC` of `C1` corresponds to an SDL-2010 type that is denoted as '**virtual**'. The redefining Classifier `EC` of `C2` represents a '**redefined**' type of SDL-2010. When the isLeaf property of a Classifier is true, this corresponds to an SDL-2010 type denoted as '**finalized**' and therefore this Classifier is no longer redefinable.

NOTE 2 – The redefinition of a classifier EC of C1 by a classifier EC of C2 implies that EC of C2 is an implicit specialization of EC of C1.

NOTE 3 – The SDL-2010 concept of virtuality constraints is not supported in SDL-UML.

NOTE 4 – Parameterized types: Each actual context parameter in the actualContextParameterList corresponds, by position, to a formal context parameter in the formalContextParameterList of the supertype.

7.4.1 Attributes

- formalContextParameterList: FormalContextParameter [0..*] {ordered}

  specifies the formal context parameters of a data type definition (see clause 11.6).

- actualContextParameterList: ActualContextParameter [0..*] {ordered}

  specifies the actual context parameters of a data type definition (see clause 11.2).

NOTE – An SdlExpression represents an actual synonym context parameter. A variable access expression has to be used in order to access a synonym context parameter.

7.4.2 Constraints

[1] Except for <<Interface>> Interface, the general property of a Classifier shall contain at most one element.

NOTE 1 – Except for an interface definition, multiple inheritances are not allowed for SDL-2010 type definitions.

[2] A Classifier that is a subtype and its more general supertype shall have the same kind of stereotype applied.

[3] Multiple redefinitions are not allowed, so there shall be at most one element in the redefinedClassifier property of a Classifier.

[4] A Classifier and its redefinedClassifier shall have the same name.

NOTE 2 – In SDL-2010, redefined types have the same name as the original type.

[5] A Classifier and its redefinedClassifier shall have the same kind of stereotype applied.

[6] If the redefinedClassifier property is not empty, the general property shall be absent.

NOTE 3 – When a Classifier A in context AB is a redefinition of another Classifier A in context AA, this implies an implicit generalization so that A in context AB is a subtype of A in context AA.

[7] The actualContextParameterList and formalContextParameterList shall be empty except in the stereotypes <<DataTypeDefinition>> Class, <<Interface>> Interface, <<ActiveClass>> Class, <<StateMachine>> StateMachine or <<Signal>> Signal.

NOTE 4 – The SDL-2010 concept of context parameters is applicable only for agent type definitions, state type definitions, procedure definitions, signal definitions or data type definitions.

[8] The number of actualContextParameterList items shall be less than or equal to the number of formalContextParameterList items in the supertype.

7.4.3 Semantics

Specialization and redefinition

The stereotypes for metaclasses that inherit from Classifier define the semantics of redefinition and specialization.

NOTE – The set of features inherited by a subtype are derived from the inheritedMember property of that subtype.

Parameterized types

A parameterized type is a type that has at least one formalContextParameterList item or has a supertype with at least one formalContextParameterList item and less actualContextParameterList items than the number of formalContextParameterList items in the supertype. A parameterized type has isAbstract true.

NOTE – An SDL-2010 type with unbound formal parameters is abstract (see clause 8.1.3 of [ITU-T Z.102]).

A Classifier with an actualContextParameterList is an anonymous type (it has an anonymous unique name) that is defined by applying the actual context parameters to the parameterized supertype as specified in clause 8.1.2 of [ITU-T Z.102]. This anonymous type is then used as type in the context where the actual context parameters are given to the parameterized supertype, for example, as the supertype for inheritance in a type definition. If the Classifier is a parameterized type, it does not have a mapping to the SDL-2010 abstract grammar. Otherwise (that is, all the formal context parameters are bound) the resulting Classifier is mapped to the SDL-2010 abstract grammar in the same manner as any other non-parameterized type.

7.4.4 References

SDL-2010 [ITU-T Z.102]:

8.1.2 Type expression

8.1.3 Abstract type

8.2 Type references and operation references

8.3 Context parameters

8.4 Specialization

8.4.1 Adding properties

8.4.2 Virtuality and virtual type

UML-SS [OMG UML]:

7.3.8 Classifier (from Kernel, Dependencies, PowerTypes, Interfaces)

7.3.47 RedefinableElement (from Kernel)

7.5 Connector

The stereotype Connector extends the metaclass Connector with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

In UML-SS, Connector is a general concept for a communication link between two instances and the mechanism for communication could be by parameter passing in variables or slots, via pointers or some other means. In this profile Connector items only provide communication by signals, which are identified by the information flows associated with the Connector and the Connector maps to a *Channel-definition*.

7.5.1 Attributes

- delay: Boolean

  If true, the signals transported on the connector are potentially delayed. The default value is true.

7.5.2 Constraints

[1] In the case of an InformationItem associated with an InformationFlow associated with a Connector, the represented property of the InformationItem shall be a Signal or an Operation or an Interface.

[2] There shall always be exactly 2 end properties.

[3] A ConnectorEnd that is part of the end property shall have empty lowerValue and upperValue properties.

[4] The role property of a ConnectorEnd that is part of the end property of the Connector shall be a Port.

[5] The type property shall be empty.

[6] The redefinedConnector property shall be empty.

[7] The isStatic property shall be false.

[8] There shall be at least one InformationFlow associated with a Connector.

7.5.3 Semantics

A <<Connector>> Connector maps to a *Channel-definition*.

The name attribute defines the *Channel-name*.

If the delay attribute of a <<Connector>> Connector is false, this maps to *NODELAY*. Otherwise the *NODELAY* is omitted.

An InformationFlow associated with a <<Connector>> Connector maps to an item in the *Channel-path-**set*** of a *Channel-definition* as follows:

- The conveyed property of an InformationFlow defines the *Signal-identifier-**set*** of the *Channel-path*.
- If the conveyed property is omitted, the *Signal-identifier-**set*** of a *Channel-path* is computed based on the realizedInterface and requiredInterface of the Port items attached to the Connector that is associated with the InformationFlow.
- If the conveyed property refers to an Interface, the *Signal-identifier-**set*** of a *Channel-path* is computed according to the transformation rules of SDL-2010 (see clause 7.7).
- The informationSource and informationTarget properties of an InformationFlow map to the *Originating-gate* and *Destination-gate* of a *Channel-path*. The *Gate-identifier* is derived from the name of the Port given by the informationSource or the informationTarget property.

NOTE 1 – InformationFlow in one direction only (with or without any InformationItem) implies that the channel is unidirectional. InformationFlow in both directions (with or without any InformationItem) implies that the channel is bidirectional.

NOTE 2 – If the partWithPort property of a ConnectorEnd is non-empty, *Gate-identifier* contains as its last path-name (before the name of the gate) the name of the part identified with partWithPort.

7.5.4 References

SDL-2010 [ITU-T Z.101]:

10.1 Channel

UML-SS [OMG UML]:

9.3.6 Connector (from InternalStructures)

9.3.7 ConnectorEnd (from InternalStructures, Ports)

17.2 InformationFlows (from InformationFlows)

7.6 DataTypeDefinition

The stereotype DataTypeDefinition extends the metaclass Class with multiplicity [0..1]. The metamodel diagram for the stereotype is defined in Figure 7-2. The concept of date type definition (a class with isActive false) is separated from active class (a class with isActive true).

The <<DataTypeDefinition>> Class represents a *Value-data-type-definition* in the SDL-2010 abstract syntax. In particular, this stereotype introduces the features of redefinition and specialization that are inherited by the subtypes of the <<DataTypeDefinition>> Class.

Specialization: A specializing data type `D2` is able to add literals, fields, choice variants, context parameters, and operations; and add default initializations or default assignments to those features inherited from its supertype `D1` (see clauses 8.4 in [ITU-T Z.102] and 12.1.9 in [ITU-T Z.104]). In the case of parameterized data types (a data type definition with context parameters), a subtype is allowed to add additional formal context parameters or to bind inherited formal context parameters of its supertype to actual context parameters.

Redefinition: If a Classifier `C2` specializes a more general superClass `C1`, an enclosed data type `ED` of `C2` is able to redefine a data type `ED` that is specified in `C1`.

NOTE – The features of specialization and redefinition are introduced by the metaclass Classifier. For the common constraints and semantics see clause 7.4.

Subtypes: The following subtypes are specified for the <<DataTypeDefinition>> Class:

- <<LiteralType>> Class that corresponds to types defined by a set of literal names.
- <<ChoiceType>> Class that corresponds to an SDL-2010 choice data type.
- <<StructureType>> Class that represents an SDL-2010 structure data type.
- <<Syntype>> Class that represents an SDL-2010 syntype definition.

7.6.1 Attributes

- isPredefined: Boolean

  if true, a data type definition represents one of the predefined data types. The default value of the property is false.

- defaultValue: SdlExpression [0..1]

  a constant expression that defines the optional default initialization of a data type definition.

NOTE 1 – The defaultValue maps to the *Default-initialization* of a *Data-type-definition* or *Syntype-definition* of any otherwise un-initialized property of an active class or local variable definition within an activity (see clause 12.3.3.2 of [ITU-T Z.101]).

NOTE 2 – Redefinition of a defaultValue occurs if both a subtype and an associated supertype have defined a defaultValue. In this case, it is the defaultValue of the subtype that specifies the default initialization of the subtype (see clause 12.3.3.2 of both [ITU-T Z.104] and [ITU-T Z.107]).

7.6.2 Constraints

[1] A <<DataTypeDefinition>> Class shall have isActive false.

[2] A <<DataTypeDefinition>> Class shall have no classifierBehavior.

[3] A nestedClassifier shall be a <<DataTypeDefinition>> Class (including its subtypes, e.g., <<LiteralType>>).

[4] An ownedAttribute where aggregation is composite shall have a type that is a <<DataTypeDefinition>> Class (including its subtypes, e.g., <<LiteralType>>) or <<Interface>> Interface.

[5] The ownedConnector, the ownedPort and the ownedTrigger properties shall be empty.

[6] Each ownedBehavior shall be an <<Activity>> Activity.

[7] The ownedReception shall be empty.

[8] If only the stereotype <<DataTypeDefinition>> is applied, the ownedAttribute property of a Class shall be empty.

[9] The isPredefined property shall only be true, when the <<DataTypeDefinition>> Class is contained in the package Predefined.

NOTE 1 – The predefined data types of SDL-UML are specified in clause 12.

[10] The defaultValue shall be an SdlExpression with isConstant true.

[11] If present, the formalContextParameterList shall only contain items that are of kind SynonymContextParameter or SortContextParameter.

7.6.3 Semantics

A <<DataTypeDefinition>> Class that is not parameterized (or has all the formal context parameters of its parameterized supertype bound – see below) maps to a *Value-data-type-definition*. The name of the <<DataTypeDefinition>> Class maps to the *Sort*.

A nestedClassifier that is a <<DataTypeDefinition>> Class (except of <<Syntype>> Class) maps to a *Value-data-type-definition* that is an element of the *Data-type-definition-**set***.

A nestedClassifier that is a <<Syntype>> Class maps to a *Syntype-definition* and is an element of the *Syntype-definition-**set***.

An ownedBehavior maps to a *Procedure-definition* in the *Procedure-definition-**set*** of the *Value-data-type-definition*.

The ownedOperation items are mapped to items in the *Static-operation-signature-**set*** of the *Value-data-type-definition*.

The optional defaultValue maps to the *Default-initialization* of a *Value-data-type-definition*.

If the isAbstract property is true, the optional *Abstract* node in the abstract syntax of a *Value-data-type-definition* is present.

The qualifiedName of the optional general property maps to the *Data-type-identifier* of the *Value-data-type-definition* that represents inheritance in the SDL-2010 abstract syntax.

If the redefinedClassifier property is not empty, this is an implicit generalization of another <<DataTypeDefinition>> Class. In this case, the qualifiedName of the redefinedClassifier maps to the *Data-type-identifier* of the *Value-data-type-definition*.

Model for inheritance of operations

For the inheritance of Operation items specified in a <<DataTypeDefinition>> Class that is a supertype of a subtype, the rules specified in clause 12.1.9 of [ITU-T Z.104] apply.

NOTE – The set of operations or attributes inherited by a subtype is derived from the inheritedMember property of that subtype.

7.6.4 References

SDL-2010 [ITU-T Z.101]:

12.1 Data definitions

12.1.1 Data type definition

12.3.3.2 Default initialization

SDL-2010 [ITU-T Z.102]:

8.1.2 Type expression

8.1.3 Abstract type

8.2 Type references and operation references

8.4 Specialization

8.4.2 Virtuality and virtual type

SDL-2010 [ITU-T Z.104]:

12.1.9 Specialization of data types

12.3.3.2 Default initialization

14 Package Predefined

SDL-2010 [ITU-T Z.107]:

12.3.3.2 Default initialization

UML-SS [OMG UML]:

7.3.6 BehavioredClassifier (from Interfaces)

7.3.7 Class (from Kernel)

7.3.47 RedefinableElement (from Kernel)

9.3.1 Class (from StructuredClasses)

9.3.8 EncapsulatedClassifier (from Ports)

13.3.2 Behavior (from BasicBehaviors)

13.3.4 BehavioredClassifier (from BasicBehaviors, Communications)

13.3.8 Class (from Communications)

7.7 Interface

The stereotype Interface extends the metaclass Interface with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

An interface defines public features that are used to communicate with an object. In SDL-UML, these are signals, remote variables and remote procedures. Accesses to remote variables and calls of remote procedures are signal exchanges in the SDL-2010 abstract grammar, so the components of an SDL-UML interface map to signals in the corresponding *Interface-definition*.

Specialization: A specializing interface is able to add signals, remote procedures and remote variables to those inherited from its supertypes. In contrast to value data type definitions, an interface multiple-inheritance is allowed (see clauses 12.1.2 and 12.1.9 in [ITU-T Z.104]).

Redefinition: If an enclosing agent `A2` (an active class) specializes a more general agent `A1`, an enclosed interface `EI` of `A2` is able to redefine an interface `EI` that is specified in `A1`.

NOTE – The features of specialization and redefinition are introduced by the metaclass Classifier. For the common constraints and semantics see clause 7.4.

7.7.1 Attributes

No additional attributes.

7.7.2 Constraints

[1] Each nestedClassifier shall be a Signal.

[2] The ownedReception property shall be empty.

[3] If the general property is not empty, each referenced element shall be an Interface.

[4] If the redefinedInterface property is not empty, each referenced element shall be an Interface.

[5] If present, the formalContextParameterList shall only contain items that are of kind SignalContextParameter or SortContextParameter.

7.7.3 Semantics

An <<Interface>> Interface maps to an *Interface-definition*.

The name defines the *Sort* of the *Interface-definition*.

The general property defines the optional *Data-type-identifier* list that represents inheritance in the SDL-2010 abstract syntax.

If the redefinedClassifier property is not empty, this is an implicit generalization of another <<Interface>> Interface. In this case, the qualifiedName of the redefinedClassifier maps to the *Data-type-identifier* of the *Interface-definition*.

The nestedClassifier, ownedAttribute, and ownedOperation properties define the rest of the contents of the interface.

The ownedAttribute and ownedOperation properties are transformed to signals according to the SDL-2010 rules for remote variables (see clause 10.6 of [ITU-T Z.102]) and remote procedures (see clause 10.5 of [ITU-T Z.102]) and are thus mapped to *Signal* items in the *Signal-definition-**set*** of the *Interface-definition*.

Each nestedClassifier property (each of which is a Signal, see constraints above) maps to an element of the *Signal-definition-**set*** of the *Interface-definition*.

7.7.4 References

SDL-2010 [ITU-T Z.102]:

8.4.2 Virtuality and virtual type

10.5 Remote procedures

10.6 Remote variables

SDL-2010 [ITU-T Z.104]:

12.1.2 Interface definition

12.1.9 Specialization of data types

UML-SS [OMG UML]:

7.3.24 Interface (from Interfaces)

13.3.15 Interface (from Communications)

7.8 LiteralType

The stereotype LiteralType is a subtype of the <<DataTypeDefinition>> Class. The metamodel diagram for the stereotype is defined in Figure 7-2.

A <<LiteralType>> Class corresponds to an SDL-2010 literal data type and its owned attributes represent the set of user-defined literals. A <<LiteralType>> Class maps to a *Value-data-type-definition* in the SDL-2010 abstract syntax.

Specialization: When a literal type is specialized, the subtype is able to add additional literals (in terms of ownedAttribute items) and operations.

7.8.1 Attributes

No additional attributes.

7.8.2 Constraints

[1] The ownedAttribute property shall not be empty.

[2] The owner and the type property of each ownedAttribute shall be equal.

NOTE – In contrast to a choice type, which consists of different kinds of data types, each literal of a literal type shall be of the same type.

[3] The literalValue property of an ownedAttribute, which is a <<Property>> Property, shall be distinct from the literalValue property of every other ownedAttribute.

7.8.3 Semantics

For the mapping of a <<LiteralType>> Class to a *Value-data-type-definition*, the mappings defined in clause 7.6 apply.

Each item of the ownedAttribute property of a <<LiteralType>> Class maps to a *Literal-signature* in the *Literal-signature-**set*** of a *Value-data-type-definition*. The unique *Literal-name* is derived from the name of the ownedAttribute plus the name of the enclosing <<LiteralType>> Class. The literalValue maps to the *Result* of the *Literal-signature*.

NOTE – A <<LiteralType>> Class implies a set of *Static-operation-signature* items as specified in clause 12.1.6.1 of [ITU-T Z.101].

7.8.4 References

SDL-2010 [ITU-T Z.101]:

12.1 Data definitions

12.1.1 Data type definition

12.1.6.1 Literals constructor

UML-SS [OMG UML]:

13.3.8 Class (from Communications)

7.9 Operation

The stereotype Operation extends the metaclass Operation with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

An operation is a feature that determines how an object behaves. If the operation is contained in an agent (that is, an <<ActiveClass>> Class), its method has to be a state machine (see clause 8.6) and maps to a procedure. An operation contained in an interface is treated as a remote procedure. Otherwise, the operation has to be an activity (see clause 9.2) and maps to an operation of the SDL-2010 data type for the <<DataTypeDefinition>> Class that contains the operation.

7.9.1 Attributes

- isOperator: Boolean

  if true, the Operation of a data type definition represents an SDL-2010 operator; otherwise it is an SDL-2010 method. The default value of the property is true.

7.9.2 Constraints

[1] If the owner of an <<Operation>> Operation is a <<DataTypeDefinition>> Class, the method associated with the <<Operation>> Operation shall be an Activity.

[2] If the owner of an <<Operation>> Operation is an <<ActiveClass>> Class, the method associated with the <<Operation>> Operation shall be a StateMachine.

[3] Both the <<Operation>> Operation and the corresponding method shall be defined in the scope of the same owner.

[4] The ownedParameter set of the <<Operation>> Operation shall be the same as the ownedParameter set of the method implementing the operation.

[5] The raisedException shall be empty.

[6] If the isOperator property is true, the redefinedOperation property shall be empty.

NOTE 1 – Redefinition is only allowed for an Operation that represents an SDL-2010 method of a data type definition (see clause 12.1.3 of [ITU-T Z.107]).

NOTE 2 – The generalization and redefinition of a procedure definition is determined by the Behavior specifying the method of an operation.

[7] If the redefinedOperation property is not empty, an Operation and its redefinedOperation shall have the same parameters except for the result parameter.

[8] If the redefinedOperation property is not empty, the result parameter of an Operation shall be the same type or a subtype as the result parameter of the redefinedOperation.

7.9.3 Semantics

Operation in an active class

An <<Operation>> Operation directly contained in an <<ActiveClass>> Class maps to a *Procedure-definition*. The name defines the *Procedure-name*. The rest of the mapping to a *Procedure-definition* is defined in clause "Mapping to a procedure definition" below.

Operation in a data type definition representing an operator

An <<Operation>> Operation directly contained in a <<DataTypeDefinition>> Class and with an isOperator property of true maps to a *Static-operation-signature* and an anonymous *Procedure-definition* identified by the *Procedure-identifier* in the abstract syntax for the *Operation-signature*.

The *Procedure-definition* is placed in the same context as the data type corresponding to the <<DataTypeDefinition>> Class. The rest of the mapping to a *Procedure-definition* is defined in clause "Mapping to a procedure definition" below.

The name of an <<Operation>> Operation defines the *Operation-name* of the *Operation-signature*.

An ownedParameter defines a *Formal-argument* or the *Operation-result* of the *Operation-signature*. The detailed mappings are specified in clause 7.11.

NOTE 1 – When an <<Operation>> Operation of a <<DataTypeDefinition>> Class is inherited from a supertype, the transformation specified in clause 7.11.3 has to be applied before the operation is mapped.

Operation in a data type definition representing a method

An <<Operation>> Operation directly contained in a <<DataTypeDefinition>> Class and with an isOperator property of false represents an SDL-2010 method. Before any mappings, the transformation specified in clause 12.1.3 of [ITU-T Z.104] has to be applied.

An Operation with an isLeaf property of false maps to a *Dynamic-operation-signature*; otherwise it maps to a *Static-operation-signature*. Furthermore, the Operation maps to an anonymous *Procedure-definition* identified by the *Procedure-identifier* in the abstract syntax for the *Operation-signature*.

The *Procedure-definition* is placed in the same context as the data type corresponding to the <<DataTypeDefinition>> Class. The rest of the mapping to a *Procedure-definition* is defined in "Mapping to a procedure definition" below.

The name of an <<Operation>> Operation defines the *Operation-name* of the *Operation-signature*.

An ownedParameter defines a *Formal-argument* or the *Operation-result* of the *Operation-signature.* The detailed mappings are specified in clause 7.11.

Operation in an interface

An <<Operation>> Operation contained in an Interface maps to signals according to the rules described in clause 7.7.3.

Mapping to a procedure definition

If the <<Operation>> Operation maps to a *Procedure-definition* (named or anonymous), each ownedParameter defines a *Procedure-formal-parameter* or the *Result* of the *Procedure-definition.* The detailed mappings are specified in clause 7.11.

The Behavior identified by the method property defines the *Procedure-graph*, *Data-type-definition-**set***, and *Variable-definition-**set*** of the *Procedure-definition.*

NOTE 2 – The Operation metaclass does not inherit from the Classifier metaclass that introduces the feature of generalization. Therefore, while it is not allowed to specialize an <<Operation>> Operation directly, it is possible to specialize the Behavior specifying the method of an Operation.

NOTE 3 – In UML-SS, an operation is not allowed to directly contain an operation itself, so therefore when the model is mapped to the SDL-2010 abstract syntax, there will never be a procedure contained within a procedure (that is, a local procedure).

7.9.4 References

SDL-2010 [ITU-T Z.101]:

9.4 Procedure

12.1.3 Operation signature

SDL-2010 [ITU-T Z.102]:

10.5 Remote procedures

10.6 Remote variables

SDL-2010 [ITU-T Z.104]:

12.1.3 Operation signature

SDL-2010 [ITU-T Z.107]:

12.1.3 Operation signature

UML-SS [OMG UML]:

7.3.5 BehavioralFeature (from Kernel)

7.3.37 Operation (from Kernel, Interfaces)

13.3.3 BehavioralFeature (from BasicBehaviors, Communications)

13.3.22 Operation (from Communications)

7.10 Package

The stereotype Package extends the metaclass Package with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

The concept of a package in UML is simply mapped to a package in SDL-2010.

7.10.1 Attributes

No additional attributes are defined.

7.10.2 Constraints

[1] All ownedMember elements of the Package shall belong to items for which mappings or transformations are described in this profile.

[2] The packageMerge composition shall be empty.

[3] The name of the Package shall not be empty.

7.10.3 Semantics

A <<Package>> Package maps to a *Package-definition*.

The name of the package maps to the *Package-name* of the *Package-definition*.

The elements of the ownedMember composition define the contents of the package, that is, the *Package-definition-**set***, *Data-type-definition-**set***, *Syntype-definition-**set***, *Signal-definition-**set***, *Agent-type-definition-**set***, *Composite-state-type-definition-**set*** and *Procedure-definition-**set***. Each ownedMember that is a nestedPackage maps to an element of the *Package-definition-**set*** of the *Package-definition*. An ownedMember that is not a nestedPackage is mapped as defined in other clauses to a *Data-type-definition*, *Syntype-definition*, *Signal-definition*, *Agent-type-definition*, *Composite-state-type-definition* or *Procedure-definition* element of the corresponding set of the *Package-definition*.

NOTE – The UML ElementImport and PackageImport (which are not stereotyped in this profile) define the import and visibility of elements of the package and define the name resolution of imported package elements. The resolved items map to *Name* and *Identifier* items in the SDL-2010 abstract syntax as described in clause 5.2.

7.10.4 References

SDL-2010 [ITU-T Z.101]:

7.2 Package

UML-SS [OMG UML]:

7.3.38 Package (from Kernel)

7.11 Parameter

The stereotype Parameter extends the metaclass Parameter with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

Depending on the context in which a Parameter is used, it represents a formal parameter of a procedure or an agent, or it represents a formal argument of an operation signature. In SDL-UML, a Parameter has an aggregation kind, which is in contrast to UML. Furthermore, the Parameter stereotype implements the SDL-2010 concept of anchored sorts.

7.11.1 Attributes

- anchored: AnchoredKind [0..1]

  This optional parameter represents an SDL-2010 anchored sort for parameters used for operations of data type definitions.

- aggregation: AggregationKind

  The aggregation kind of a parameter. The default value is none.

7.11.2 Constraints

[1] The anchored property shall only be present for an <<Operation>> Operation that is owned by a <<DataTypeDefinition>> Class.

NOTE 1 – In SDL-2010, an anchored sort is legal concrete syntax only if it occurs within a data type definition.

[2] If the anchored property is present, the type property of a Parameter shall refer to the next enclosing <<DataTypeDefinition>> Class.

NOTE 2 – An SDL-2010 anchored sort shall name the sort introduced by the enclosing data type definition.

[3] The aggregation shall not be of AggregationKind shared.

7.11.3 Semantics

Parameters of a procedure definition

If the <<Operation>> Operation maps to a *Procedure-definition* (named or anonymous), each Parameter that does not have a return direction defines (in order) a *Procedure-formal-parameter*. The direction (in, inout, or out) of an ownedParameter determines (respectively) if the corresponding *Procedure-formal-parameter* is an *In-parameter* or *Inout-parameter* or *Out-parameter*. Each of these formal parameters is a *Parameter* and detailed mappings are defined below.

A Parameter that does have a direction of return defines the *Result* of the *Procedure-definition*. The *Sort-reference-identifier* of the *Result* is determined in the same way as for a <<Property>> Property (see clause 7.13.3). The aggregation property maps to the *Result-aggregation* of a *Result*. If the aggregation is of AggregationKind none, the *Aggregation-kind* is **REF**. Otherwise, if the aggregation is of AggregationKind composite, the *Aggregation-kind* is **PART**.

NOTE 1 – The aggregation kind '**PART**' is a feature of Basic SDL-2010 (see clause 12.3.1 of [ITU-T Z.101]), whereas the aggregation kind '**REF**' is introduced in [ITU-T Z.107] in order to support object-oriented data.

Agent formal parameters of an agent type definition

The ownedParameter set of a <<StateMachine>> StateMachine that specifies the classifierBehavior of an <<ActiveClass>> Class maps to the *Agent-formal-parameter* list of the *Agent-type-definition*. Each of these formal parameters is a *Parameter* and detailed mappings are defined below.

Mapping to a parameter

An ownedParameter of an <<Operation>> Operation or <<ActiveClass>> Class representing a *Parameter* is mapped as described below.

The name and type (including the multiplicity) of the ownedParameter define, respectively, the *Variable-name* and the *Sort-reference-identifier* of the *Parameter*. The *Sort-reference-identifier* is determined in the same way as for a <<Property>> Property (see clause 7.13.3). The aggregation property of an ownedParameter maps to the *Parameter-aggregation* of a *Procedure-formal-parameter*. If the aggregation is of AggregationKind none, the *Aggregation-kind* is **REF**. Otherwise, if the aggregation is of AggregationKind composite, the *Aggregation-kind* is **PART**.

Formal arguments and result of an operation signature

The ownedParameter set of an <<Operation>> Operation that defines an *Operation-signature* is mapped as follows:

For each ownedParameter that does not have a return direction, the type and multiplicity together define (in order of the parameters) a *Formal-argument* of the *Operation-signature* with a type

determined in the same way as for a <<Property>> Property (see clause 7.13.3). The type of the <<Operation>> Operation defines the *Operation-result* of the *Operation-signature*.

NOTE 2 – For the mapping to an Operation-signature, the aggregation property is ignored.

Transformation of anchored parameters

Before an inherited Operation of a <<DataTypeDefinition>> Class that is a subtype of a supertype (its general property is not empty) is mapped to an *Operation-signature*, the transformation as specified in clause 12.1.9 of [ITU-T Z.104] has to be applied on each ownedParameter that has an anchored property.

7.11.4 References

UML-SS [OMG UML]:

7.3.42 Parameter (from Kernel)

SDL-2010 [ITU-T Z.101]:

8.1.1.1 Agent types

9.4 Procedure

12.1.3 Operation signature

SDL-2010 [ITU-T Z.104]:

12.1.3 Operation signature

12.1.9 Specialization of data types

SDL-2010 [ITU-T Z.107]:

12.3.1 Variable definition

7.12 Port

The stereotype Port extends the metaclass Port with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

An SDL-UML port defines an SDL-2010 *Gate*. The required interfaces characterize the requests from the classifier to its environment through the port and therefore define the outgoing signals for the *Gate*. The provided interfaces of a port characterize requests to the classifier that are permitted through the port and therefore define the incoming signals for the *Gate*.

7.12.1 Attributes

No additional attributes.

7.12.2 Constraints

[1] The redefinedPort property shall be empty.

[2] The aggregationKind shall be composite.

[3] The isDerived and isDerivedUnion properties shall be false.

[4] The isReadOnly property shall be true.

[5] The defaultValue property shall be empty.

[6] The subsettedProperty property shall be empty.

[7] The qualifier property shall be empty.

[8] The isStatic property shall be false.

[9] The lowerValue and upperValue properties shall be ValueSpecification items that evaluate to 1.

[10] The isService property shall be false.

7.12.3 Semantics

A <<Port>> Port maps to a *Gate-definition*.

The name defines the *Gate-name*.

The list of required interfaces maps to the *Out-signal-identifier-**set***. The set is computed according to the rules given in clause 12.1.2 of [ITU-T Z.101].

The list of provided interfaces defines the *In-signal-identifier-**set***. The set is computed according to the rules given in clause 12.1.2 of [ITU-T Z.101].

If isBehavior is true, a channel is constructed in the SDL-2010 abstract syntax that connects the gate and the state machine of the containing agent.

7.12.4 References

SDL-2010 [ITU-T Z.101]:

8.1.4 Gate

12.1.2 Interface definition

UML-SS [OMG UML]:

9.3.12 Port (from Ports)

7.13 Property

The Property extends the metaclass Property with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

A property is an attribute that corresponds to a variable definition, an agent instance, or a field of a structure type, or a literal signature of a literal type or a variant of a choice type in SDL-2010.

NOTE – The mappings of properties (ownedAttribute) owned by data type definition are specified in the context of the specific stereotypes.

7.13.1 Attributes

- initialNumber: UnlimitedNatural [0..1]

  defines the initial number of instances created when an instance of the containing classifier is created.

- literalValue: UnlimitedNatural [0..1]

  defines the literal number of an attribute owned by a literal type.

7.13.2 Constraints

[1] The type shall not be omitted.

[2] If the upperValue is omitted, the lowerValue shall also be omitted.

[3] If the upperValue is included, the lowerValue shall also be included.

NOTE 1 – The upper and lower bounds of multiplicity are optional in UML-SS.

[4] If the upperValue value is greater than 1 and isOrdered is true, isUnique shall be false,

NOTE 2 – That is because there is not a predefined SDL-2010 data type that is ordered and requires each of its elements to have unique values.

[5] The initialNumber shall be included only if the type is an <<ActiveClass>> Class.

[6] The value of the initialNumber shall not be less than the lowerValue.

[7] The value of the initialNumber shall not be greater than the upperValue.

[8] The literalValue shall be included only if the type is a <<LiteralType>> Class.

[9] The isDerived shall be false.

[10] The isDerivedUnion shall be false.

[11] If isReadOnly is true, the type shall be a <<DataTypeDefinition>> Class.

NOTE 3 – A Property with isReadOnly is true corresponds to a synonym definition in the concrete grammar of SDL-2010.

[12] The defaultValue shall be an SdlExpression with isConstant true.

[13] The redefinedProperty shall be empty.

NOTE 4 – Since <<Property>> Property maps to variable definition or an identifier of an agent or data type in SDL-2010, the feature of redefinition is not applicable. That is because these kinds of SDL-2010 elements cannot be redefined.

[14] The aggregation shall not be of AggregationKind shared.

7.13.3 Semantics

Mapping to Variable-definition

A <<Property>> Property owned by an <<ActiveClass>> Class or <<StateMachine>> StateMachine maps to *Variable-definition*, if its type is a <<DataTypeDefinition>> Class (or an <<Interface>> Interface) and its isReadOnly property is false.

The aggregation property maps to the *Aggregation-kind* of a *Variable-definition*. If the aggregation is of AggregationKind none, the *Aggregation-kind* is **REF**. Otherwise, if the aggregation is of AggregationKind composite, the *Aggregation-kind* is **PART**.

NOTE 1 – The aggregation kind '**PART**' is a feature of Basic SDL-2010 (see clause 12.3.1 of [ITU-T Z.101]), whereas the aggregation kind '**REF**' is introduced in [ITU-T Z.107] in order to support object-oriented data.

The name defines the *Variable-name*. The defaultValue defines the *Constant-expression*. The *Sort-reference-identifier* is the *Sort-identifier* of the sort derived from the type property. The *Sort-identifier* is determined as follows:

- If there is no upperValue and no lowerValue, the name of the type maps to the *Sort-identifier*.
- Otherwise, the *Sort-identifier* identifies an anonymous sort formed from the SDL-2010 predefined `Bag` (if isOrdered is false and isUnique is false) or `Powerset` (if isOrdered is false and isUnique is true) or `String` (if isOrdered is true) datatype instantiated with the sort given by the type as the `ItemSort`. The anonymous sort is a *Value-data-type-definition* or *Syntype-definition* in the same context as the *Variable-definition*. If the upperValue value is omitted or the lowerValue value is zero and the upperValue value is unlimited (* value in UML), there are no size constraints and the anonymous sort is a *Value-data-type-definition* with its components derived from the instantiated predefined data type. Otherwise, the lowerValue value and upperValue value map (as described below) to a *Range-condition* of the anonymous sort, which is a *Syntype-definition*. The *Parent-sort-identifier* of this *Syntype-definition* is a reference to another anonymous sort that is the *Value-data-type-definition* derived in the same way as the case with no size constraints.
- The mapping of lowerValue value and upperValue value to a *Range-condition* (see above) is to a *Condition-item-**set*** consisting of one *Condition-item*. If the upperValue value is unlimited, the *Condition-item* is an *Open-range* where the *Operator-identifier* identifies the ">=" (greater than or equal to) operator for the parent sort, and the lowerValue value maps to the *Constant-expression* of this *Open-range*. Otherwise (when upperValue value is not unlimited), the *Condition-item* is a *Closed-range*, and the lowerValue value maps to the

first *Constant-expression* of the *Closed–range* and the upperValue value maps to the second *Constant-expression* of the *Closed–range*.

NOTE 2 – In UML the multiplicity of a property is separate from the type of the property; whereas in SDL-2010, the bounds, uniqueness of values and ordering of elements are considered to be part of a data type and, if these differ, two types are considered to be different and incompatible. If two properties have the same type but have different bounds and both map to `Bags`, `Powersets` or `Strings`, the bounds are treated as size constraints, so in these special cases two types could be compatible if they both had the same kind and item sort. The mappings defined above result in anonymous data types for each property, which has multiple values, with the consequence that such properties cannot be compatible even for the special cases. In SDL-2010 it is possible to define a type that has a specific name and item sort (and in the case of a Vector the upper bound) and to use this for different variable definitions so that the value of one variable is assignable to another using the same type.

Mapping to Constant-expression

If isReadOnly is true, the type is required to be a <<DataTypeDefinition>> Class. In this case, the <<Property>> Property maps to a *Variable-definition* as described above. The <<Property>> Property maps to a *Variable-access* each time the <<Property>> Property is used in an expression.

Mapping to Agent-definition

If the type is an <<ActiveClass>> Class, the <<Property>> Property maps to an *Agent-definition*. The name defines the *Agent-name*. The type property defines the *Agent-type-identifier* that represents the type in the SDL-2010 abstract syntax. The initialNumber defines the *Initial-number*. The upperValue defines the *Maximum-number*. If the initialNumber is omitted, the lowerValue defines the *Initial-number*. If both the initialNumber and lowerValue are omitted, the *Initial-number* is 1. The lowerValue defines the *Lower-bound*.

NOTE 3 – It is possible for the number of agent instances to go below the *Initial-number*.

7.13.4 References

SDL-2010 [ITU-T Z.101]:

9 Agents

12.3.1 Variable definition

SDL-2010 [ITU-T Z.104]:

14.3 String sort

14.9 Vector sort

14.10 Powerset sort

14.13 Bag sort

SDL-2010 [ITU-T Z.107]:

12.3.1 Variable definition

UML-SS [OMG UML]:

7.3.33 MultiplicityElement (from Kernel)

7.3.45 Property (from Kernel, Association Classes, Interfaces)

7.3.50 StructuralFeature (from Kernel)

7.3.53 TypedElement (from Kernel)

7.14 Signal

The stereotype Signal extends the metaclass Signal with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

A signal represents the type for communication message instances and maps to a *Signal-definition*.

Specialization: A specializing signal is allowed to append additional attributes to those inherited from its supertype (see clause 8.4.1 in [ITU-T Z.102]).

Redefinition: If an enclosing agent `A2` (an active class) specializes a more general agent `A1`, an enclosed signal `ES` of `A2` is able to redefine a signal `ES` that is specified in `A1`. This corresponds to the redefinition of virtual signal types (see clause 8.4.2 in [ITU-T Z.102]).

NOTE – The features of specialization and redefinition are introduced by the metaclass Classifier. For the common constraints and semantics see clause 7.4.

7.14.1 Attributes

No additional attributes.

7.14.2 Constraints

[1] The aggregation of an ownedAttribute shall not be of AggregationKind shared.

[2] If present, the formalContextParameterList shall only contain items that are of kind SortContextParameter.

7.14.3 Semantics

A <<Signal>> Signal maps to a *Signal-definition*. The name defines the *Signal-name* and the general property maps to the optional *Signal-identifier*.

If the redefinedClassifier property is not empty, this is an implicit generalization of another <<Signal>> Signal. In this case, the qualifiedName of the redefinedClassifier maps to the *Signal-identifier*.

Each ownedAttribute maps to an item in the *Signal-parameter* list. The type of an ownedAttribute defines the *Sort-reference-identifier* and its aggregation property defines the *Aggregation-kind*. If the aggregation is of AggregationKind none, the *Aggregation-kind* is **REF**. Otherwise, if the aggregation is of AggregationKind composite, the *Aggregation-kind* is **PART**.

NOTE 1 – The aggregation kind '**PART**' is a feature of Basic SDL-2010 (see clause 12.3.1 of [ITU-T Z.101]), whereas the aggregation kind '**REF**' is introduced in [ITU-T Z.107] in order to support object-oriented data.

If the isAbstract property is true, the optional *Abstract* node in the abstract syntax of a *Signal-definition* is present.

7.14.4 References

SDL-2010 [ITU-T Z.102]:

8.1.3 Abstract type

8.4.1 Adding properties

10.3 Signal

SDL-2010 [ITU-T Z.107]:

12.3.1 Variable definition

UML-SS [OMG UML]:

13.3.24 Signal (from Communications)

7.15 Specification

The stereotype Specification extends the metaclass Model with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 7-1.

7.15.1 Attributes

No additional attributes are defined.

7.15.2 Constraints

[1] All ownedMember elements of the Model shall be a <<Package>> Package or an <<ActiveClass>> Class.

[2] At least one ownedMember shall be of type <<ActiveClass>> Class.

[3] The packageMerge composition shall be empty.

[4] The name of the Package shall not be empty.

7.15.3 Semantics

A <<Specification>> Model maps to an *SDL-specification*.

Each nestedPackage maps to the *Package-definition-**set***. An ownedMember that is an <<ActiveClass>> Class maps to the optional *Agent-definition* and the name maps to the *Agent-name*. The *Initial-number* of the *Agent-definition* is 1. The qualifiedName of <<ActiveClass>> forms the *Agent-type-identifier* of the *Agent-definition*.

7.15.4 References

SDL-2010 [ITU-T Z.101]:

7.1 Framework

UML-SS [OMG UML]:

17.3.1 Model (from Models)

7.16 StructureType

The stereotype StructureType is a subtype of the <<DataTypeDefinition>> Class. The metamodel diagram for the stereotype is defined in Figure 7-2.

The <<StructureType>> Class represents an SDL-2010 structure data type and it maps to a *Value-data-type-definition*. A structure data type consists of a set of mandatory or optional fields that are allowed to have different types. It is allowed to omit optional fields in a value for a structure type, whereas mandatory fields always have to be present.

Specialization: When a structure type is specialized, its subtypes are able to add additional fields (in terms of ownedAttribute items) and operations.

7.16.1 Attributes

No additional attributes.

7.16.2 Constraints

[1] The ownedAttribute property shall not be empty.

[2] An ownedAttribute shall have a type that is a <<DataTypeDefinition>> Class (or one of its subtypes) or <<Interface>> Interface.

7.16.3 Semantics

A <<StructureType>> Class maps to a *Value-data-type-definition*. Depending on the multiplicity of the Property that is an ownedAttribute item, a distinction is made between the following three cases:

- A multiplicity of [0..1] is an optional data field.
- A multiplicity of [1..1] is a mandatory data field.
- In all other cases, an anonymous data type has to be derived from the multiplicity and the type of an ownedAttribute as specified in clause 7.13.3.

Each ownedAttribute implies a set of implicit defined operations as specified in clause 12.1.6.2 of [ITU-T Z.104]. During the computation of these operations, also the defaultValue of an ownedAttribute is evaluated.

NOTE – The defaultValue of an ownedProperty corresponds to the default initialization of a data field.

In addition, the mappings specified in the context of the <<DataTypeDefinition>> Class (see clause 7.6) apply.

7.16.4 Notation

UML standard syntax is used.

7.16.5 References

SDL-2010 [ITU-T Z.104]:

12.1 Data definitions

12.1.1 Data type definition

12.1.6.2 Structure data types

UML-SS [OMG UML]:

13.3.8 Class (from Communications)

7.17 Syntype

The Syntype stereotype is a subtype of the <<DataTypeDefinition>> Class. The metamodel diagram for the stereotype is defined in Figure 7-2.

The Syntype stereotype represents an SDL-2010 syntype and it maps to a *Syntype-definition*. The Syntype stereotype constrains a predefined or user-defined data type in order to restrict the usable set of valid values. The association between a <<Syntype>> Class and the constrained <<DataTypeDefinition>> Class shall be established by a Dependency.

7.17.1 Attributes

- constraint: RangeCondition

  The range condition that defines the constraint.

7.17.2 Constraints

[1] The type of the constraint property of a <<Syntype>> Class and the supplier of a Dependency (between the constrained data type and its associated syntype) shall refer to the same <<DataTypeDefinition>> Class.

[2] The ownedAttribute and ownedOperation properties shall be empty.

[3] The general and redefinedClassifier properties shall be empty.

NOTE – An SDL-2010 syntype cannot be generalized or redefined.

7.17.3 Semantics

The name property of a <<Syntype>> Class maps to the *Name* and the constraint maps to the *Range-condition* of a *Syntype-definition*. In addition, the qualifiedName of the constraint maps to the *Parent-sort-identifier*.

The optional defaultValue maps to the *Default-initialization* of a *Syntype-definition*.

7.17.4 References

SDL-2010 [ITU-T Z.101]:

12.1.8.1 Syntypes

12.1.8.2 Constraint

UML-SS [OMG UML]:

13.3.8 Class (from Communications)

7.3.12 Dependency (from Dependencies)

7.18 Timer

The Timer stereotype is a subtype of the stereotype Signal (see clause 7.14). The metamodel diagram for the stereotype is defined in Figure 7-1.

7.18.1 Attributes

- defaultValue: SdlExpression [0..1]

  The optional default value for timer initialization.

7.18.2 Constraints

[1] The defaultValue shall be an SdlExpression with isConstant true.

[2] The general and redefinedClassifier properties shall be empty.

NOTE – In contrast to a <<Signal>> Signal, neither redefinition nor generalization is allowed for a <<Timer>> Signal. That is because these features are not applicable to an SDL-2010 timer definition.

7.18.3 Semantics

A <<Timer>> Signal maps to a *Timer-definition*. The name attribute defines the *Timer-name*. The type of each ownedAttribute defines a corresponding item in the list of *Sort-reference-identifiers*. If present, the defaultValue maps to the optional *Timer-default-initialization*.

7.18.4 References

SDL-2010 [ITU-T Z.101]:

11.15 Timer

UML-SS [OMG UML]:

13.3.24 Signal (from Communications)

8 State machines

The finite state machine models of SDL-UML provide details of how a model behaves in terms of state transitions for the protocol part of a system.

The following metaclasses from the UML package BehaviorStateMachines are included:

– FinalState

– Pseudostate

– Region

– State

– StateMachine

– Transition.

8.1 State machine metamodel diagrams

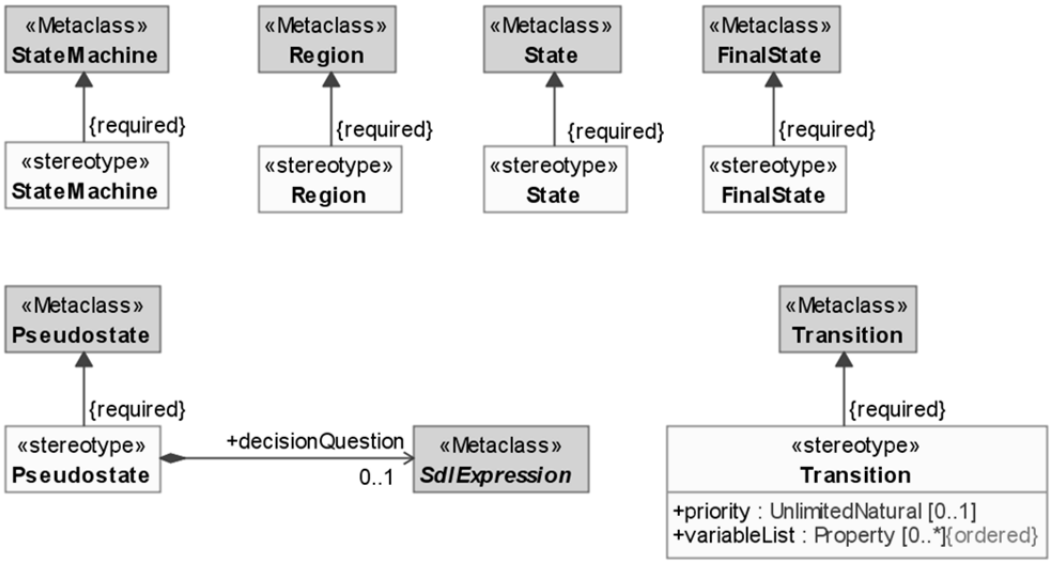

Figure 8-1 – State machine stereotypes

8.2 FinalState

The stereotype FinalState extends the metaclass FinalState with multiplicity [1..1]. The metamodel diagram of the stereotype is defined in Figure 8-1.

When a FinalState is reached the containing graph completes. In SDL-UML a graph for a procedure will complete with a <<Return>> ActivityFinalNode. In this case, there is no mapping to the SDL-2010 abstract syntax for FinalState because the return node terminates the graph. A FinalState that is not in a procedure graph maps to an *Action-return-node* or *Named-return-node* for the enclosing composite state.

8.2.1 Attributes

No additional attributes.

8.2.2 Constraints

[1] If the <<FinalState>> FinalState is part of the region of a <<StateMachine>> StateMachine that maps to a *Procedure-graph*, the name of the <<FinalState>> FinalState shall be empty and any Transition that has the <<FinalState>> FinalState as its target shall end in a <<Return>> ActivityFinalNode.

NOTE – The *Action-return-node* or *Value-return-node* of the procedure is defined by the <<Return>> ActivityFinalNode.

8.2.3 Semantics

Mapping to an Action-return-node or a Stop-node

If the <<FinalState>> FinalState has an empty name and it is not part of the region of a <<StateMachine>> StateMachine that maps to a *Procedure-graph*, the <<FinalState>> FinalState maps to a *Stop-node* or an *Action-return-node*. It maps to a *Stop-node* if (and only if) it is part of the region of a <<StateMachine>> StateMachine that is the classifierBehavior of an <<ActiveClass>> Class.

NOTE – In UML FinalState the context object of the state machine is terminated if all enclosed regions are terminated, whereas in SDL-2010 an explicit stop is required, but, on the other hand, in SDL-2010 it is not allowed to have a return node in the state machine of an agent.

Mapping to a Named-return-node

If the <<FinalState>> FinalState has a non-empty name, it maps to a *Named-return-node* where the name defines the *State-exit-point-name*.

8.2.4 References

SDL-2010 [ITU-T Z.101]:

11.12.2.4 Return

UML-SS [OMG UML]:

15.3.2 FinalState (from BehaviorStateMachines)

8.3 Pseudostate

The stereotype Pseudostate extends the metaclass Pseudostate with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 8-1.

A Pseudostate is used instead of a state before initial or state entry point transitions, when there is a junction of transitions, when there is a decision to make a choice of transitions, when the transition leads to a history nextstate, or after a transition to lead to a state exit point or terminate the state graph. They allow more complex transitions between states to be built from simpler, shorter transitions that end or start (or start and end) in a Pseudostate. They map to start, next state (with history), decision, join and free action, return and stop nodes in the SDL-2010 state transition graph.

8.3.1 Attributes

- decisionQuestion: SdlExpression [0..1]

  An optional expression that defines the question for a choice Pseudostate.

8.3.2 Constraints

[1] A Transition shall have an empty guard property if the Transition is an outgoing property of a <<Pseudostate>> Pseudostate with kind initial.

[2] A Transition shall have an empty trigger property if the Transition is an outgoing property of a <<Pseudostate>> Pseudostate with kind initial.

[3] The classifierBehavior of a <<ActiveClass>> Class with isAbstract false shall have a <<Pseudostate>> Pseudostate with kind initial.

[4] The kind property of <<Pseudostate>> Pseudostate shall not be join or fork.

[5] A <<Pseudostate>> Pseudostate with kind of deepHistory or shallowHistory or exitPoint or terminate shall not have an outgoing property.

[6] The optional decisionQuestion shall only be present for a <<Pseudostate>> Pseudostate with kind choice.

[7] A Transition shall have a non-empty guard property Constraint (a RangeCondition or the predefined "else" guard) and an empty trigger property if the Transition is an outgoing property of a <<Pseudostate>> Pseudostate with kind choice.

[8] Each Boolean guard of each Transition that is an outgoing property of a <<Pseudostate>> Pseudostate with kind choice, except the predefined "else" guard, shall be a RangeCondition.

[9] A <<Pseudostate>> Pseudostate with kind choice shall have at most one outgoing Transition with an empty trigger property and an "else" guard. The Constraint representing this guard shall have a specification property that is a RangeCondition that always evaluates to true.

8.3.3 Semantics

Mapping of an initial node

A <<Pseudostate>> Pseudostate with kind initial is mapped to a *Procedure-start-node* in a region that defines a *Procedure-graph* and *State-start-node* in a region that defines a *Composite-state-graph*. The outgoing Transition maps to the *Graph-node* list of the *Transition* of the *Procedure-start-node* or *State-start-node*. The target property of this outgoing Transition maps to the last item of the *Transition* (a *Terminator* or *Decision-node*) of the *Procedure-start-node* or *State-start-node* in the same way as the target is mapped in clause 8.7 for a Transition.

If the outgoing Transition of a <<Pseudostate>> Pseudostate with kind initial is redefining (the redefinedTransition property is not empty) another transition, the redefining Transition specifies the *Transition* of a *Procedure-start-node* or *State-start-node*.

NOTE 1 – When the outgoing transition of an initial node is redefined this corresponds to a virtual procedure start (see clause 9.4 in [ITU-T Z.102]) or a virtual process start (see clause 11.1 in [ITU-T Z.102]).

NOTE 2 – A Pseudostate cannot be redefined, so that an outgoing Transition of the Pseudostate has to be used for the purpose of redefinition.

Mapping of a deep history node

A <<Pseudostate>> Pseudostate with kind deepHistory maps to a *Nextstate-node* that is a *Dash-nextstate* with **HISTORY**.

Mapping of a shallow history node

A <<Pseudostate>> Pseudostate with kind shallowHistory maps to a *Nextstate-node* that is a *Dash-nextstate* without **HISTORY**.

Mapping of a junction node

A <<Pseudostate>> Pseudostate with kind junction maps to a *Free-action* and one or more *Join-node* elements. The name property defines the *Connector-name* in the *Free-action* and each *Join-node*. The effect of the outgoing property maps to the *Graph-node* list of the *Transition* of the *Free-action*. The target property of this outgoing property Transition maps to the last item of the *Transition* (a *Terminator* or *Decision-node*) of the *Free-action* in the same way as the target is mapped in clause 8.7 for a Transition. There is a *Join-node* for each Transition that has a target property that is a <<Pseudostate>> Pseudostate with kind junction and the *Join-node* is the *Terminator* of the *Transition* with its *Graph-node* list derived from the effect of the Transition.

NOTE 3 – UML-SS has a constraint "a junction vertex must have at least one incoming and one outgoing transition". Pseudostate maps to both the *Join-node* elements and the *Free-action* labels, so the possibility (allowed in SDL-2010) to have a *Free-action* without a corresponding *Join-node* is not allowed.

Mapping of a choice node

A <<Pseudostate>> Pseudostate with kind choice maps to a *Decision-node*. The decisionQuestion maps to the *Decision-question* and the outgoing Transition items map to the *Decision-answer-**set***. The Boolean guard property of each outgoing Transition maps to the *Range-condition* of the corresponding *Decision-answer*. The effect of this outgoing Transition maps to the *Graph-node* list of the *Transition* of the same *Decision-answer*.

The target property of each outgoing Transition maps to the last item of the *Transition* (a *Terminator* or *Decision-node*) of the same *Decision-answer* in the same way as the target is

mapped in clause 8.7 for a Transition. An outgoing property with an "else" guard property maps to an *Else-answer* where the *Transition* is mapped in the same way as for a Boolean guard property.

Mapping of an entry point

A <<Pseudostate>> Pseudostate with kind entryPoint maps to a *State-start-node*. The name property defines the *State-entry-point-name*. The effect of the outgoing Transition defines the *Graph-node* list of the *Transition*. The target property of this outgoing Transition maps to the last item of the *Transition* (a *Terminator* or *Decision-node*) of the *State-start-node* in the same way as the target is mapped in clause 8.7 for a Transition.

If the outgoing Transition of a <<Pseudostate>> Pseudostate with kind entryPoint is redefining (the redefinedTransition property is not empty) another transition, the redefining Transition specifies the *Transition* of the *State-start-node*.

Mapping of an exit point

A <<Pseudostate>> Pseudostate with kind exitPoint maps to a *Named-return-node*. The name property defines the *State-exit-point-name*.

Mapping of a termination node

A <<Pseudostate>> Pseudostate with kind terminate maps to a *Stop-node*.

8.3.4 References

SDL-2010 [ITU-T Z.101]:

11.1 Start

11.10 Label (connector name)

11.12.2.2 Join

11.12.2.3 Stop

11.13.5 Decision

SDL-2010 [ITU-T Z.102]:

8.4.3 Virtual transition/save

9.4 Procedure

11.1 Start

UML-SS [OMG UML]:

15.3.8 Pseudostate (from BehaviorStateMachines)

15.3.9 PseudostateKind (from BehaviorStateMachines)

8.4 Region

The stereotype Region extends the metaclass Region with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 8-1.

A region contains states and transitions and maps to the definition of how a procedure or a composite state behaves. For the composite state mapping of a StateMachine, a single region maps to a *Composite-state-graph*, whereas two or more regions map to a *State-aggregation-node* (see clause 8.5). A region in SDL-UML is always part of a StateMachine and is never part of a State, because the region of a State is constrained to be empty.

8.4.1 Attributes

No additional attributes.

8.4.2 Constraints

[1] A Region that extends another Region (as specified by an extendedRegion property) shall have the same name as the extended Region.

[2] The triggers in the different orthogonal regions shall refer to disjoint sets of signals.

[3] The extendedRegion property of a Region shall be empty.

NOTE – The redefinition of a procedure or composite state type is determined by the StateMachine that contains a Region.

8.4.3 Semantics

Mapping to a Procedure-graph

A <<Region>> Region that is the region of StateMachine with a specification maps to a *Procedure-graph,* and the subvertex set of Vertex elements (State, Pseudostate or FinalState) of the region together with the transition elements of the region that reference these Vertex elements, define the *Procedure-graph*.

Mapping to a Composite-state-graph

A <<Region>> Region that is the only region of a StateMachine without a specification maps to a *Composite-state-graph,* and the subvertex set of Vertex elements (State, Pseudostate, or FinalState) of the region together with the transition elements of the region that reference these Vertex elements, define the *Composite-state-graph* of the StateMachine mapping. Each *State-node* or *Free-action* derived from these Vertex elements are elements of the *State-node-**set*** and *Free-node-**set***, respectively, of the *State-transition-graph* of the *Composite-state-graph*.

Mapping to a Composite-state-type-definition

Otherwise, each <<Region>> Region that is one of two or more region items of a StateMachine (the outer *Composite-state-type-definition*) without a specification maps to a *State-partition* and to an inner *Composite-state-type-definition* with a unique *State-type-name*. Each *State-partition* is an element of the *State-partition-**set*** of the *State-aggregation-node* of the outer *Composite-state-type-definition* of the StateMachine mapping. The mapping to a *State-partition* and the corresponding inner *Composite-state-type-definition* is described in more detail in the following paragraphs.

Each Pseudostate with kind entryPoint (in the connectionPoint property of the containing StateMachine) maps to a distinct *State-entry-point-definition* of the inner *Composite-state-type-definition*.

Because a Pseudostate of kind entryPoint is directly owned by a StateMachine and not by one of its region items, the association between a *State-entry-point-definition* and its containing *Composite-state-type-definition* has to be determined. For this purpose, the container property (which refers to the containing Region) of the outgoing Transition of a Pseudostate with kind entryPoint is used to determine the containing *Composite-state-type-definition*. The result of determination is mapped so that the *Connection-definition-**set*** of the *State-partition* contains an *Entry-connection-definition* that connects the *State-entry-point-definition* of the outer *Composite-state-type-definition* to the corresponding *State-entry-point-definition* of the inner *Composite-state-type-definition.*

NOTE 1 – The *State-entry-point-name*s of the *Outer-entry-point* and *Inner-entry-point* of an *Entry-connection-definition* are equal. That is because a Pseudostate with kind entryPoint maps to a *State-entry-point-definition* of the outer as well as of the inner *Composite-state-type-definition*.

Each Pseudostate with kind exitPoint in the connectionPoint property of the containing StateMachine maps to a distinct *State-exit-point-definition* of the *Composite-state-type-definition*.

Because a Pseudostate of kind exitPoint is directly owned by a StateMachine and not by one of its region items, the association between a *State-exit-point-definition* and its containing *Composite-state-type-definition* has to be determined. For this purpose, the container property (which refers to

the containing Region) of the incoming Transition of a Pseudostate with kind exitPoint is used to determine the containing *Composite-state-type-definition*. The result of determination is mapped so that the *Connection-definition-**set*** of the *State-partition* contains an *Exit-connection-definition* that connects the *State-exit-point-definition* of the outer *Composite-state-type-definition* to the corresponding *State-exit-point-definition* of the inner *Composite-state-type-definition*.

NOTE 2 – The *State-exit-point-name*s of the *Outer-exit-point* and *Inner-exit-point* of an *Exit-connection-definition* are equal. That is because a Pseudostate with kind exitPoint maps to a *State-entry-point-definition* of the outer as well as of the inner *Composite-state-type-definition*.

The name maps to the *Name* of the *State-partition*.

The *Composite-state-type-identifier* of the *State-partition* identifies the inner *Composite-state-type-definition*.

The subvertex and transition properties of the Region map to the *Composite-state-graph* of the inner *Composite-state-type-definition* in the same way that a *Composite-state-graph* is derived for only one region in a StateMachine. See clauses 8.5, 8.3 and 8.2 covering subclasses of Vertex (that is, State, Pseudostate, or FinalState, respectively) and clause 8.7 for more details.

8.4.4 References

SDL-2010 [ITU-T Z.101]:

8.1.1.5 Composite state type

SDL-2010 [ITU-T Z.102]:

11.11.2 State aggregation

UML-SS [OMG UML]:

13.3.2 Behavior (from BasicBehaviors)

15.3.10 Region (from BehaviorStateMachines)

8.5 State

The stereotype State extends the metaclass State with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 8-1.

A state represents a condition where an object is waiting for some condition to be fulfilled: usually for an event to occur. A state in SDL-UML maps to an SDL-2010 state.

8.5.1 Attributes

No additional attributes.

8.5.2 Constraints

[1] The doActivity property shall be empty.

[2] The entry and exit properties shall be empty, because entry/exit actions are not supported.

[3] The isComposite property shall be false, because only decomposition using submachine properties is allowed and a State shall have an empty region property.

[4] If a trigger of an outgoing Transition has an omitted port property, the associated signal shall only be used in another outgoing Transition or deferrableTrigger when the corresponding port property is not empty.

NOTE 1 – This constraint specifies that signal without **via** gate shall only be used for another input or save with a gate.

[5] If a trigger of an outgoing Transition has a port property, the associated signal shall not be used in another outgoing Transition or deferrableTrigger with a port that has the same name.

NOTE 2 – This constraint specifies the rule for a signal with **via** gate.

[6] The event property of the deferrableTrigger property of a State shall be a SignalEvent.

NOTE 3 – A SignalEvent is used to represents events for received signals or expired timers, which are declared in terms of <<Timer>> Signal items.

8.5.3 Semantics

A <<State>> State maps to a *State-node*. The name maps to the *State-name*.

A ConnectionPointReference that is part of the connection property and corresponds to an *Exit-Connection-Point* (a Pseudostate with kind exitPoint in the connectionPoint property of the containing StateMachine) maps to a member of the *Connect-node-**set***.

The submachine property maps to *Composite-state-type-identifier*.

Each item in the deferrableTrigger list maps to a *Save-item* in the *Save-item-**set*** of the *Save-signalset*. The qualifiedName of a Signal that is the event of a deferrableTrigger maps to the *Signal-identifier* of a *Save-item*. The qualifiedName of the port property of a deferrableTrigger maps to the optional *Gate-identifier* of a *Save-item*.

The outgoing property (inherited from Vertex) maps to the *Input-node-**set***, *Spontaneous-transition-**set*** and *Continuous-signal-**set***. See clause 8.7 on Transition for more details on the mapping to the *Input-node-**set***, *Spontaneous-transition-**set*** and *Continuous-signal-**set***.

NOTE – The semantics for parameters for state types is defined in clause 8.6.3, subclause "Mapping to a Composite-state-type-definition", and clause 8.7.3, subclause "Mappings of the target property".

8.5.4 References

SDL-2010 [ITU-T Z.102]:

8.4.3 Virtual transition/save

11.2 State

11.7 Save

UML-SS [OMG UML]:

15.3.11 State (from BehaviorStateMachines, ProtocolStateMachines)

15.3.16 Vertex (from BehaviorStateMachines)

8.6 StateMachine

The stereotype StateMachine extends the metaclass StateMachine with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 8-1.

An SDL-UML StateMachine either maps to the graph of an SDL-2010 procedure or an SDL-2010 composite state type. The two cases are distinguished by whether or not the StateMachine has a specification. If it does, then it is the procedure case; otherwise, it is a composite state type. Because there are two different mappings, some constraints on StateMachine are dependent on whether there is a specification or not.

Specialization: A state machine `S2` (subtype) is allowed to specialize from a more general state machine `S1` (supertype). In general, the subtype state machine is allowed to add additional states, transitions, regions and parameters to those elements inherited from the supertype state machine (see clause 8.4.1 in [ITU-T Z.102]). When a subtype StateMachine adds additional parameters, they have to be added after those parameters inherited from the supertype. When a state machine maps to

a procedure, a subtype is not allowed to add additional regions. That is because a state machine that is the specification of an operation shall consist of one region only.

Redefinition: The redefinition of a state machine `ES` is only possible if an enclosing classifier `C1` is specialized by a classifier `C2` that contains a state machine `ES` (the redefining state machine) (see clause 8.4.2 in [ITU-T Z.102]).

NOTE – The features of specialization and redefinition are introduced by the metaclass Classifier. For the common constraints and semantics see clause 7.4.

8.6.1 Attributes

No additional attributes.

8.6.2 Constraints

[1] The isReentrant property shall be false.

[2] The ownedConnector shall be empty.

[3] If the redefinedClassifier is not empty, the redefining and the redefined StateMachine shall each have the same ownedParameter list.

If a StateMachine maps to a *Composite-state-type* (the classifierBehavior property is not empty) the following constraints apply:

[4] No ownedParameter property shall have a direction=return (so that StateMachine does not return a result).

[5] The specification property shall be empty.

[6] The classifierBehavior property shall be an <<ActiveClass>> Class.

If a StateMachine maps to a *Procedure-graph* (the specification property is not empty) the following constraints apply:

[7] The specification property shall be an Operation.

NOTE – The other possibility, Reception, is not allowed.

[8] There shall only be one Region.

[9] The connectionPoint property shall be empty.

[10] The classifierBehavior shall be empty.

[11] The ownedPort shall be empty.

[12] The specification shall not be an Operation contained in an Interface.

[13] The ownedParameter list of the StateMachine shall be the same as the ownedParameter list of the Operation that is the specification property.

8.6.3 Semantics

A <<StateMachine>> StateMachine maps to a *Composite-state-type-definition* or a *Procedure-graph*. If the StateMachine has a specification, the StateMachine maps to the *Procedure-graph* (as defined by its contained Region) of the *Procedure-definition* from the mapping of the <<Operation>> Operation identified by the specification. If the StateMachine does not have a specification, the StateMachine maps to a *Composite-state-type-definition*.

Mapping to a Procedure-graph

Semantics for the *Procedure-graph* case (where the *Procedure-definition* is the mapping of <<Operation>> Operation identified by the specification):

The region property defines the *Procedure-graph* through the subvertex set of Vertex elements (State, Pseudostate, or FinalState) of the region together with the transition elements of the region that reference these Vertex elements. Each *State-node* or *Free-action* derived from these Vertex

elements are elements of the *State-node-**set*** and *Free-node-**set***, respectively, of the *Procedure-graph*.

NOTE 1 – A Pseudostate with kind initial defines the *Procedure-start-node*.

The nestedClassifier and ownedAttribute associations (both inherited from Class via Behavior) define the rest of the contents of the state machine according to the following paragraphs.

A nestedClassifier that is a <<DataTypeDefinition>> Class defines a *Value-data-type-definition* that is an element of the *Data-type-definition-**set*** of the *Procedure-definition*.

A nestedClassifier that is an Interface defines an *Interface-definition* that is an element of the *Data-type-definition-**set*** of the *Procedure-definition*.

A nestedClassifier that is a <<StateMachine>> StateMachine defines a *Composite-state-type-definition* that is an element of the *Composite-state-type-definition-**set*** of the *Procedure-definition*.

An ownedOperation defines a *Procedure-definition* that is an element of the *Procedure-definition-**set*** of the *Procedure-definition* mapping the Operation identified by the specification.

An ownedAttribute maps to a *Variable-definition* in the *Variable-definition-**set*** of the *Procedure-definition* (see clause 7.13).

If the isAbstract property of a <<StateMachine>> StateMachine is true, the optional *Abstract* node in the abstract syntax of a *Procedure-definition* is present.

If the general property is not empty, this refers to the specialization of a procedure definition. In this case, the qualifiedName of the generalized StateMachine maps to the *Procedure-identifier* of the *Procedure-definition*.

If the redefinedClassifier property is not empty, this is an implicit generalization of another StateMachine. In this case, the qualifiedName of the redefinedClassifier maps to the *Procedure-identifier* of the *Procedure-definition*.

Mapping to a Composite-state-type-definition

The name defines the *State-type-name*. If the region contains only one Region, the content of the region maps to a *Composite-state-graph* of the *Composite-state-type-definition*; otherwise the region maps to a *State-aggregation-node* of the *Composite-state-type-definition* with one *State-partition* for each contained Region.

Each connectionPoint with kind entryPoint defines an element of the *State-entry-point-definition-**set*** and each connectionPoint with kind exitPoint defines an element of *State-exit-point-definition-**set***. The name property of a connectionPoint with kind entryPoint or exitPoint maps to the *Name* of a *State-entry-point-definition* or *State-exit-point-definition*, respectively.

The ownedParameter property defines the *Composite-state-formal-parameter*s.

The nestedClassifier and ownedAttribute associations define the rest of the contents of the state machine according to the following paragraphs.

A nestedClassifier that is a <<DataTypeDefinition>> Class defines a *Value-data-type-definition* that is an element of the *Data-type-definition-**set***.

A nestedClassifier that is an Interface defines an *Interface-definition* that is an element of the *Data-type-definition-**set***.

A nestedClassifier that is a <<StateMachine>> StateMachine defines a *Composite-state-type-definition* that is an element of the *Composite-state-type-definition-**set***.

An ownedOperation defines a *Procedure-definition* that is an element of the *Procedure-definition-**set***.

An ownedAttribute maps to a *Variable-definition* in the *Variable-definition-**set*** (see clause 7.13).

If the isAbstract property of a <<StateMachine>> StateMachine is true, the optional *Abstract* node in the abstract syntax of a *Composite-state-type-definition* is present.

The general property (derived from generalization) maps to the optional *Composite-state-type-identifier*.

If the redefinedClassifier property is not empty, this is an implicit generalization of another StateMachine. In this case, the qualifiedName of the redefinedClassifier maps to the *Composite-state-type-identifier*.

NOTE 2 – If a StateMachine is a classifierBehavior and it has an ownedParameter set, these parameters are used as parameters when creating instances of the containing Class. See clause 7.2.3.

8.6.4 References

SDL-2010 [ITU-T Z.101]:

9.4 Procedure

SDL-2010 [ITU-T Z.102]:

8.1.1.5 Composite state type

8.1.3 Abstract type

UML-SS [OMG UML]:

13.3.2 Behavior (from BasicBehaviors)

13.3.4 BehavioredClassifier (from BasicBehaviors, Communications)

15.3.12 StateMachine (from BehaviorStateMachines)

8.7 Transition

The stereotype Transition extends the metaclass Transition with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 8-1.

A transition is the part of a state transition graph that defines what happens when the object goes from one vertex in the graph to another vertex. Each vertex is usually a state, but may be a pseudostate. Signals (including timer signals) timers are used to trigger transitions. Standard UML notation and semantics are used.

8.7.1 Attributes

- priority: UnlimitedNatural [0..1]

  The priority determines the order of interpretation of a received signal.

- variableList: Property [0..*] {ordered}

  Reference to owned attributes of a StateMachine used to store submitted values of a received Event.

8.7.2 Constraints

[1] The Transition shall have kind external or local.

NOTE 1 – The UML concept of internal transitions is not allowed.

[2] The port of a Trigger that is the trigger property of a Transition shall at most refer to one Port.

[3] The event property of the trigger property of a <<Transition>> Transition shall be an AnyReceiveEvent, SignalEvent, CallEvent or ChangeEvent.

NOTE 2 – A SignalEvent is used to represents events for received signals or expired timers, which are declared in terms of <<Timer>> Signal items.

[4] The effect property shall reference an Activity.

NOTE 3 – There is a constraint on states that signals for each transition have to be distinct, so that a given signal is not allowed to trigger more than one transition.

8.7.3 Semantics

A <<Transition>> Transition that is the outgoing property of a <<Pseudostate>> Pseudostate with kind choice is mapped as defined for Pseudostate in clause 8.3.3.

NOTE 1 – In this clause the term 'trigger event of a <<Transition>> Transition' means the Event that is the event property of the Trigger that is the trigger property of the Transition. The Event is a MessageEvent (an AnyReceiveEvent, a SignalEvent, or a CallEvent) or ChangeEvent.

Transformation of an asterisk state list

If the <<Transition>> Transition has a TransitionKind that is local, it is expanded as specified in clause 11.2 of [ITU-T Z.103] before applying any expansions or mappings below.

Transformation of an asterisk input list

If the trigger event of a <<Transition>> Transition is an AnyReceiveEvent, the transition is expanded as specified in clause 11.3 of [ITU-T Z.103] before applying any expansions or mappings below.

Transformation of a remote procedure call

If the trigger event of a <<Transition>> Transition is a CallEvent, the transition is expanded as specified in clause 10.5 of [ITU-T Z.102] before any expansions or mappings below.

Mapping to a Spontaneous-transition

If the trigger event of a <<Transition>> Transition is a SignalEvent and the name of the Signal is "none" or "NONE" (case sensitive therefore excludes "None", etc.), the Transition maps to a *Spontaneous-transition*. The effect property maps to the *Graph-node* list of the *Transition* of the *Spontaneous-transition*.

If a Transition that maps to a *Spontaneous-transition* is redefining another transition (the redefinedTransition property is not empty), the redefining Transition specifies the *Spontaneous-transition*.

NOTE 2 – A redefining transition that maps to a *Spontanous-transition* corresponds to a virtual continuous signal in SDL-2010 (see clause 11.5 in [ITU-T Z.102]).

Mapping to an Input-node

If the trigger event of a <<Transition>> Transition is a SignalEvent (a received signal or an expired timer) and the name of the Signal is neither "none" nor "NONE" (so it does not map to *Spontaneous-transition*), the Transition maps to an *Input-node*.

If present, the priority maps to the optional *Priority-name* of an *Input-node*.

The qualifiedName of the Signal maps to the *Signal-identifier* of the *Input-node*. The qualifiedName of the port property of the trigger of a Transition maps to the optional *Gate-identifier* of the *Input-node*.

The qualifiedName of each item in the variableList (by order) maps to a *Variable-identifier* of the *Input-node*. The effect property maps to the *Graph-node* list of the *Transition* of the *Input-node*.

NOTE 3 – Because UML provides no concrete mechanisms for storing submitted values of received events, the variableList property of a Transition is used for this purpose.

If a Transition that maps to an *Input-node* is redefining another transition (the redefinedTransition property is not empty), the redefining Transition specifies the *Input-node*.

NOTE 4 – A redefining transition that maps to an *Input-node* corresponds to a virtual input (see clause 11.3 in [ITU-T Z.102]) or virtual priority input (see clause 11.4 in [ITU-T Z.102]) in SDL-2010.

Mapping to a Continuous-signal

If the trigger event of a <<Transition>> Transition is a ChangeEvent, the transition maps to a *Continuous-signal*.

The changeExpression maps to the *Continuous-expression* of the *Continuous-signal*. The effect property maps to the *Graph-node* list of the *Transition* of the *Continuous-signal*. The priority maps to the *Priority-name*.

If the <<Transition>> Transition has an empty trigger property and a non-empty guard property, the Transition maps to a *Continuous-signal*. The guard maps to the *Continuous-expression* of the *Continuous-signal*. The effect property maps to the *Graph-node* list of the *Transition* of the *Continuous-signal*. The priority maps to the *Priority-name*.

NOTE 5 – It is a consequence of the SDL-2010 semantics that in the Transition set defined by the outgoing properties of a State, when evaluating the guard of each *Continuous-signal* (each Transition with only a guard and an empty trigger), an unevaluated guard of a Transition with a lowest priority attribute is evaluated before any unevaluated guard of a Transition with a higher priority attribute.

If a Transition that maps to a *Continuous-signal* is redefining another transition (the redefinedTransition property is not empty), the redefining Transition specifies the *Continuous-signal*.

NOTE 6 – A redefining transition that maps to a *Continuous-signal* corresponds to a virtual continuous signal in SDL-2010 (see clause 11.5 in [ITU-T Z.102]).

Mapping to a Connect-node

If the <<Transition>> Transition has an empty trigger property and an empty guard property, the Transition maps to a *Connect-node*. The effect property maps to the *Graph-node* list of the *Transition* of the *Connect-node*.

If the source of the Transition is a ConnectionPointReference, the qualifiedName of the exit property Pseudostate of the ConnectionPointReference maps to *State-exit-point-name*. If the source is a State, the *State-exit-point-name* is empty.

If a Transition that maps to a *Connect-node* is redefining another transition (the redefinedTransition property is not empty), the redefining Transition specifies the *Connect-node*.

NOTE 7 – A redefining transition that maps to a *Connect-node* corresponds to a virtual connect in SDL-2010 (see clause 11.11.4 in [ITU-T Z.102]).

Mapping to a Decision-node

If a <<Transition>> Transition has a non-empty trigger property and non-empty guard property and is not the outgoing property of a <<Pseudostate>> Pseudostate with kind choice, the guard maps to the *Transition* as follows:

- A *Decision-node* is inserted first in the *Transition* with a *Decision-answer* with a Boolean *Range-condition* that is the *Constant-expression* true and another *Decision-answer* for false.
- The specification property of the guard property of the Transition maps to *Decision-question* of the *Decision-node*.
- The false *Decision-answer* has a *Transition* that is a *Dash-nextstate* without **HISTORY**.
- The effect property of the Transition maps to the *Graph-node* list of the *Transition* of the true *Decision-answer*.

NOTE 8 – The mapping to a *Decision-node* instead of mapping to an enabling condition (a *Provided-expression*) makes it possible to access the signal parameters from the expression in the guard and also means that the signal is consumed even if guard is false, whereas if an enabling condition is false the signal is not consumed.

NOTE 9 – The mapping to a *Decision-node* works because entry/exit actions are not allowed on states. If such actions were allowed, the exit and entry actions of the states would be incorrectly invoked even when taking the false branch through the decision.

Mappings of the target property

A target property that is a State maps to a *Terminator* of the *Transition* (mapped from the effect) where this *Terminator* is a *Nextstate-node* that is a *Named-nextstate* without *Nextstate-parameters,* and where the qualifiedName of the State maps to the *State-name* of the *Named-nextstate*.

A target property that is a ConnectionPointReference maps to a *Terminator* of the *Transition* (mapped from the effect) where this *Terminator* is a *Nextstate-node* that is a *Named-nextstate* with *Nextstate-parameters*, and where the qualifiedName of the state property of the ConnectionPointReference maps to the *State-name* of the *Named-nextstate,* and the qualifiedName of the entry property Pseudostate of the ConnectionPointReference maps to *State-entry-point-name* of the *Nextstate-parameters*.

A target property that is a Pseudostate maps to the last item of the *Transition* (a *Terminator* or *Decision-node*) as defined in clause 8.3.3.

8.7.4 References

SDL-2010 [ITU-T Z.102]:

8.4.3 Virtual transition/save

10.5 Remote procedure

11.3 Input

11.4 Priority Input

11.5 Continuous signal

11.8 Spontaneous transition

11.11.4 Connect

SDL-2010 [ITU-T Z.103]:

11.2 State

11.3 Input

UML-SS [OMG UML]:

13.3.25 SignalEvent (from Communications)

13.3.31 Trigger (from Communications)

15.3.1 ConnectionPointReference (from BehaviorStateMachines)

15.3.14 Transition (from BehaviorStateMachines)

9 Actions and activities

An activity is used to describe how the model behaves, for example, the control flow of actions in an operation body or a transition. When invoked, each action takes zero or more inputs, usually modifies the state of the system in some way such as a change of the values of an instance, and

produces zero or more outputs. The values that are used by an action are described by value specifications (see clause 10), obtained from the output of actions or in ways specific to the action.

The following packages from UML are included either explicitly or because elements of the packages are generalizations that are specialized as the elements that are used:

– BasicActions
– BasicActivities
– BasicBehaviors
– CompleteActivities
– CompleteStructuredActivities
– FundamentalActivities
– IntermediateActivities
– IntermediateActions
– Kernel
– StructuredActions
– StructuredActivities.

The following metaclasses from UML are included:

– Activity
– ActivityFinalNode
– AddStructuralFeatureValueAction
– AddVariableValueAction
– CallOperationAction
– CreateObjectAction
– ConditionalNode
– LoopNode
– OpaqueAction
– OpaqueExpression
– SendSignalAction
– SequenceNode.

9.1 Action and activity metamodel diagrams

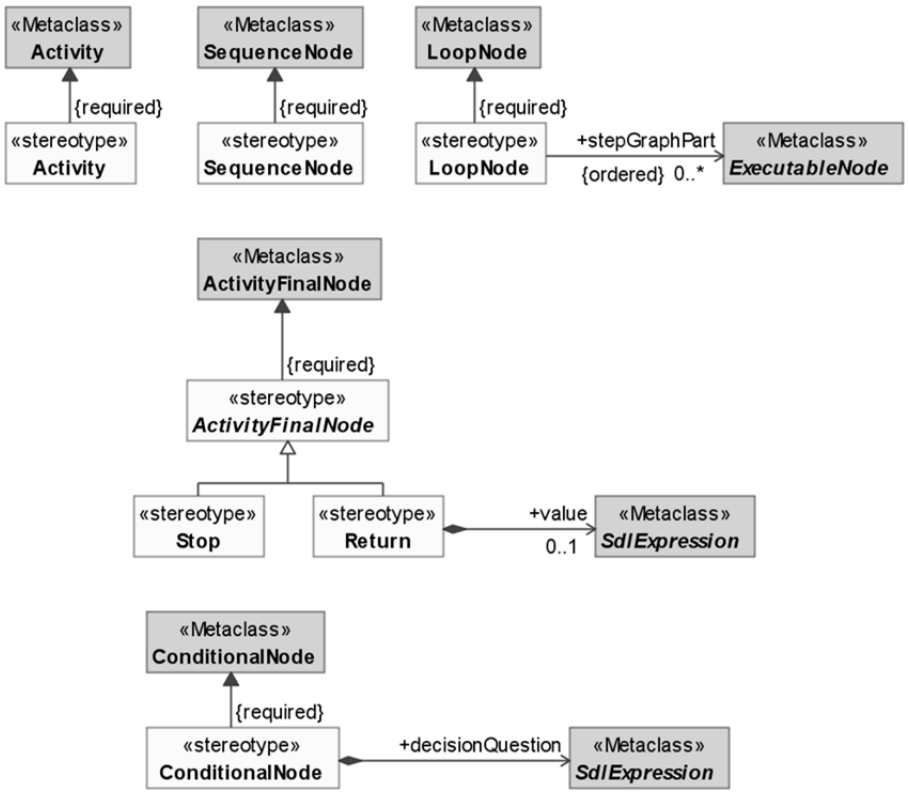

Figure 9-1 – Activity stereotypes

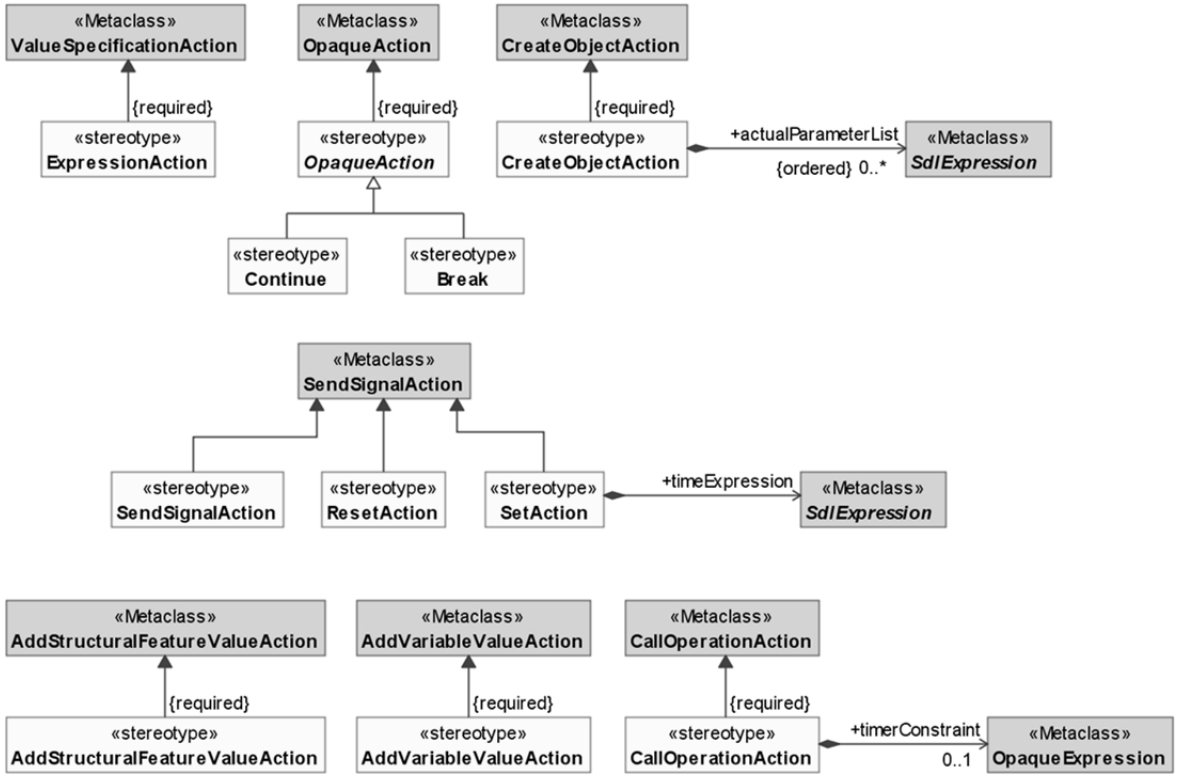

Figure 9-2 – Action stereotypes

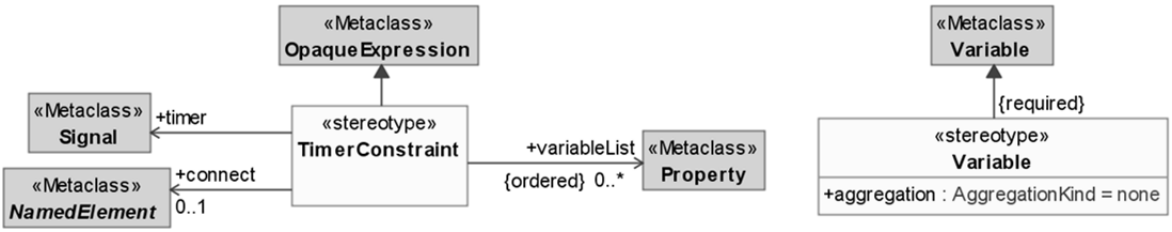

Figure 9-3 – Auxiliary stereotypes

9.2 Activity

The stereotype Activity extends the metaclass Activity with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-1.

An activity defines the effect of a transition or the body of an operation of a data type definition.

9.2.1 Attributes

No additional attributes.

9.2.2 Constraints

[1] Each node of an <<Activity>> Activity shall be an Action or a StructuredActivityNode that is defined in this profile.

[2] The variable property of an <<Activity>> Activity shall be empty.

[3] The redefinedClassifier and general properties shall be empty.

NOTE 1 – Neither a procedure graph nor the graph node list of a transition in SDL-2010 is redefinable.

[4] The redefinedElement property of each node and edge of an <<Activity>> Activity shall be empty.

[5] The isAbstract property shall be false.

[6] The ownedPort and ownedConnector properties shall be empty.

If an <<Activity>> Activity maps to the *Graph-node* list of a *Transition*:

[7] The ownedAttribute and variable properties shall be empty.

[8] The nestedClassifier property shall be empty.

If an <<Activity>> Activity maps to the *Procedure-graph* of a *Procedure-definition*:

[9] A nestedClassifer shall be a <<DataTypeDefinition>> Class (including its subtypes).

[10] The specification property shall be an <<Operation>> Operation.

9.2.3 Semantics

Mapping to a Graph-node

An <<Activity>> Activity that is the effect of a Transition maps to the *Graph-node* list of the *Transition* for the effect. Each node of the Activity maps to an item in the *Graph-node* list of the *Transition*.

Mapping to a Procedure-definition

An <<Activity>> Activity that has a specification (that is, the Activity is the method of an Operation) maps to a *Procedure-graph* containing only a *Procedure-start-node* consisting of a *Transition*. Each Action or ActivityNode of an Activity maps to the *Graph-node* list of the *Transition*.

An ownedAttribute maps to a *Variable-definition* in the *Variable-definition-**set*** of the *Procedure-definition* (see clause 7.13).

A nestedClassifier that is a <<DataTypeDefinition>> Class maps to a *Value-data-type-definition* that is an element of the *Data-type-definition-**set*** of the *Procedure-definition*.

A nestedClassifier that is an Interface maps to an *Interface-definition* that is an element of the *Data-type-definition-**set*** of the *Procedure-definition*.

In addition, the mapping rules specified in point [8] above and clause 7.11.3 apply.

9.2.4 References

SDL-2010 [ITU-T Z.101]:

11.12 Transition

UML-SS [OMG UML]:

12.3.4 Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities)

9.3 ActivityFinalNode

The stereotype ActivityFinalNode extends the metaclass ActivityFinalNode with multiplicity [1..1]. This stereotype is abstract and its metamodel diagram is defined in Figure 9-1.

This stereotype is introduced to ensure that every ActivityFinalNode is one of the subtypes: <<Return>> ActivityFinalNode or <<Stop>> ActivityFinalNode.

9.3.1 Attributes

No additional attributes.

9.3.2 Constraints

No additional constraints.

9.3.3 Semantics

The subtypes of <<ActivityFinalNode>> ActivityFinalNode give its semantics.

9.3.4 References

UML-SS [OMG UML]:

12.3.6 ActivityFinalNode (from BasicActivities, IntermediateActivities)

9.4 AddStructuralFeatureValueAction

The stereotype AddStructuralFeatureValueAction extends the metaclass AddStructuralFeatureValueAction with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

An <<AddStructuralFeatureValueAction>> AddStructuralFeatureValueAction is used to define an assignment to structural features of a Class or other Classifier.

9.4.1 Attributes

No additional attributes.

9.4.2 Constraints

[1] The value property shall be a ValuePin.

[2] The type of the value and of the structuralFeature property shall refer to a <<DataTypeDefinition>> Class (which includes its subtypes).

[3] The object property shall be an InputPin and its type property shall refer to an <<ActiveClass>> Class or <<StateMachine>> StateMachine.

9.4.3 Semantics

An <<AddStructuralFeatureValueAction>> AddStructuralFeatureValueAction maps to an *Assignment*.

The value property maps to the *Expression* of the *Assignment* and the qualifiedName of the structuralFeature property maps to the *Variable-identifier*.

NOTE – In a notation for SDL-UML that supports "extended variables" (notation for indexed elements and/or field elements of a data item), they are transformed as specified in clause 12.3.3.1 of [ITU-T Z.101] before the mapping to an AddStructuralFeatureValueAction.

9.4.4 References

SDL-2010 [ITU-T Z.101]:

12.3.3 Assignment

UML-SS [OMG UML]:

11.3.5 AddStructuralFeatureValueAction (from IntermediateActions)

11.3.48 StructuralFeatureAction (from IntermediateActions)

9.5 AddVariableValueAction

The stereotype AddVariableValueAction extends the metaclass AddVariableValueAction with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

An <<AddVariableValueAction>> AddVariableValueAction is used to specify a value assignment to local variables of compound statements.

9.5.1 Attributes

No additional attributes.

9.5.2 Constraints

[1] The value property shall be a ValuePin.

[2] The type of the value or the variable property shall refer to a <<DataTypeDefinition>> Class (which includes its subtypes) or <<Interface>> Interface.

9.5.3 Semantics

An <<AddVariableValueAction>> AddVariableValueAction maps to an *Assignment*.

The value property maps to the *Expression* of the *Assignment* and the qualifiedName of the variable property maps to the *Variable-identifier*.

NOTE – In a notation for SDL-UML that supports extended variables, they are transformed as specified in clause 12.3.3.1 of [ITU-T Z.101] before the mapping to an AddVariableValueAction.

9.5.4 References

SDL-2010 [ITU-T Z.101]:

12.3.3 Assignment

UML-SS [OMG UML]:

11.3.6 AddVariableValueAction (from StructuredActions)

11.3.53 VariableAction (from StructuredActions)

9.6 Break

The stereotype Break is a subtype of the stereotype OpaqueAction. The metamodel diagram for the stereotype is defined in Figure 9-2.

A <<Break>> OpaqueAction represents a break action that causes termination of an enclosing statement labelled by the name given. The enclosing statement is a loop node, a sequence node or a conditional node. A break action causes interpretation to be transferred to the point following the enclosing node with the matching connector name.

9.6.1 Attributes

No additional attributes.

9.6.2 Constraints

[1] A <<Break>> OpaqueAction shall have an empty input property.

[2] A <<Break>> OpaqueAction shall only be used inside of a LoopNode, ConditionalNode or SequenceNode that has a name that is the same as the name of the <<Break>> OpaqueAction.

NOTE – According to this constraint, the specification of a <<Break>> OpaqueAction is allowed within a nested StructuredActivityNode that is enclosed by a <<LoopNode>> LoopNode.

9.6.3 Semantics

A <<Break>> OpaqueAction maps to a *Break-node* and its name property maps to the *Connector-name*.

9.6.4 References

SDL-2010 [ITU-T Z.101]:

11.10 Label (connector name)

SDL-2010 [ITU-T Z.102]:

11.14.1 Compound and loop statements

9.7 CallOperationAction

The stereotype CallOperationAction extends the metaclass CallOperationAction with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

Depending on the context, a call operation action maps to the call of a procedure (*Call-node*) in the SDL-2010 abstract grammar or it is transformed to an implicit exchange of signals (remote procedure invocation). For the description in this clause, the following terminology is used:

- The operation-owner is the <<ActiveClass>> Class that has (as an ownedOperation property) the Operation identified by the operation property of the <<CallOperationAction>> CallOperationAction.
- The active-container is the closest containing <<ActiveClass>> Class of the CallOperationAction.

9.7.1 Attributes

- timerConstraint: OpaqueExpression [0..1]

  The optional timer communication constraint for remote procedure calls.

9.7.2 Constraints

[1] The timerConstraint shall be a <<TimerConstraint>> OpaqueExpression.

[2] The target property shall be a ValuePin.

[3] If the CallOperationAction maps to a *Call-node*, the target, the onPort and the timerConstraint properties shall be empty.

[4] If the CallOperationAction does not map to a *Call-node*, the value of the target property shall be an SdlExpression that conforms to the type of `Predefined::Pid`.

9.7.3 Semantics

Mapping to a Call-node

A <<CallOperationAction>> CallOperationAction maps to a *Call-node* if the active-container is the same as the operation-owner or is a generalization of the operation-owner.

For mapping to a *Call-node*, the qualifiedName of the operation property maps to the *Procedure-identifier* of the *Call-node*. The argument property list maps to the *Actual-parameters* list of the *Call-node*.

Mapping to a remote procedure call

If the criteria for mapping to a *Call-node* are not satisfied, the <<CallOperationAction>> CallOperationAction is transformed to a signal exchange as specified in clause 10.5 of [ITU-T Z.102] for a remote procedure call, including transformation of the optional timerConstraint property.

9.7.4 References

SDL-2010 [ITU-T Z.101]:

11.13.3 Procedure call

SDL-2010 [ITU-T Z.102]:

10.5 Remote procedure

11.13.3 Procedure call

UML-SS [OMG UML]:

11.3.10 CallOperationAction (from BasicActions)

9.8 ConditionalNode

The stereotype ConditionalNode extends the metaclass ConditionalNode with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-1.

A <<ConditionalNode>> ConditionalNode is used to define textual switch statements and maps to a *Decision-node* in SDL-2010. A Pseudostate with kind choice also maps to a *Decision-node*.

9.8.1 Attributes

- decisionQuestion: SdlExpression

  An expression that defines the question for a ConditionalNode.

9.8.2 Constraints

[1] Each item in the body of each Clause shall be an Action or a StructuredActivityNode that is defined in this profile.

[2] Each Clause of a <<ConditionalNode>> ConditionalNode shall have a test part that is an <<ExpressionAction>> ValueSpecificationAction representing an SDL-2010 range condition.

[3] A <<ConditionalNode>> ConditionalNode shall have at most one "else" Clause.

NOTE 1 – It is assumed that the <<ExpressionAction>> ValueSpecificationAction of the test part always returns true.

[4] For every Clause except the "else" Clause, the predecessorClause set shall be empty, so that there is no requirement that any Clause is evaluated before any other Clause (except the "else" Clause).

[5] The predecessorClause set for an "else" Clause shall include every other Clause, so that they all have to be evaluated before the "else" Clause.

[6] For every Clause except the "else" Clause, the successorClause set shall contain only the "else" Clause if there is one; otherwise the successorClause set shall be empty, because the order of evaluation is never enforced in SDL-2010.

[7] The successorClause set of the "else" Clause shall be empty.

NOTE 2 – The "else" Clause is a Clause that is a successor to all others and whose test part always returns true, so that it is only invoked if all others are false (see UML-SS 12.3.18 ConditionalNode).

[8] The isAssured property shall be true. Therefore either there shall be an "else" Clause, or there shall be at least one test that succeeds.

9.8.3 Semantics

A <<ConditionalNode>> ConditionalNode maps to a *Decision-node*. The decisionQuestion property maps to the common *Decision-question*. The Clause set (excluding the "else" Clause) defines the *Decision-answer-**set*** of the *Decision-body*. The test of each Clause is an <<ExpressionAction>> ValueSpecificationAction that maps to the *Range-condition* (see clause 9.11) in each *Decision-answer*. The body of the Clause maps to *Transition* in the corresponding *Decision-answer*. The "else" Clause (if present) defines the *Else-answer*; otherwise there is no *Else-answer*.

NOTE – The decider property of a Clause, owned by an <<ConditionalNode>> ConditionalNode, references the same OutputPin as the output property of the <<ExpressionAction>> ValueSpecificationAction used as the test of that Clause.

9.8.4 References

SDL-2010 [ITU-T Z.101]:

11.13.5 Decision

UML-SS [OMG UML]:

12.3.17 Clause (from CompleteStructuredActivities, StructuredActivities)

12.3.18 ConditionalNode (from CompleteStructuredActivities, StructuredActivities)

9.9 Continue

The stereotype Continue is a subtype of the stereotype OpaqueAction. The metamodel diagram for the stereotype is defined in Figure 9-2.

A <<Continue>> OpaqueAction represents a continue action within a loop that causes a jump to the next iteration of the loop or termination of the loop if already in the last iteration.

9.9.1 Attributes

No additional attributes.

9.9.2 Constraints

[1] A <<Continue>> OpaqueAction shall have an empty input property.

[2] A <<Continue>> OpaqueAction shall only be used inside of a LoopNode that has a name with a value equal to the name of the <<Continue>> OpaqueAction.

NOTE – According to this constraint, specification of a <<Continue>> OpaqueAction is allowed within a nested StructuredActivityNode that is enclosed by a <<LoopNode>> LoopNode.

9.9.3 Semantics

A <<Continue>> OpaqueAction maps to a *Continue-node* and its name property maps to the *Connector-name*.

9.9.4 References

SDL-2010 [ITU-T Z.102]:

11.14.1 Compound and loop statements

9.10 CreateObjectAction

The stereotype CreateObjectAction extends the metaclass CreateObjectAction with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

A create object action is used to create instances of agents.

9.10.1 Attributes

- actualParameterList: SdlExpression [0..*] {ordered}

  The list of expressions representing the actual parameters of the agent to be created.

9.10.2 Constraints

[1] The classifier property shall refer to an <<ActiveClass>> Class.

9.10.3 Semantics

The <<CreateObjectAction>> CreateObjectAction maps to a *Create-request-node* where the classifier maps to the *Agent-identifier*. Each SdlExpression in actualParameterList maps to an *Expression* of the *Actual-parameters* list.

NOTE – According to the semantics of SDL-2010 for a *Create-request-note* (see clause 11.13.2 of [ITU-T Z.101]), the Pid value for a created agent is stored in its '**self**' variable and in the '**offspring**' variable of the creating agent. A variable access expression is used to retrieve the Pid value of the '**offspring**' variable, for instance, in order to send a signal to a newly created agent.

9.10.4 References

SDL-2010 [ITU-T Z.101]:

11.13.2 Create

UML-SS [OMG UML]:

11.3.16 CreateObjectAction (from IntermediateActions)

9.11 ExpressionAction

The stereotype ExpressionAction extends the metaclass ValueSpecificationAction with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

An <<ExpressionAction>> ValueSpecificationAction represents an action that only contains an expression. This is a utility to simplify the modelling of a <<ConditionalNode>> ConditionalNode or <<LoopNode>> LoopNode.

9.11.1 Attributes

No additional attributes.

9.11.2 Constraints

[1] The value property of an <<ExpressionAction>> shall be a RangeCondition or an SdlExpression.

NOTE – The result property of an <<ExpressionAction>> ValueSpecificationAction is only used to be compliant with the UML-SS in the context of a <<ConditionalNode>> ConditionalNode.

9.11.3 Semantics

If an <<ExpressionAction>> ValueSpecificationAction is used in the context of an <<ConditionalNode>> ConditionalNode, it maps to a *Range-condition* (see 9.8). Otherwise, if an <<ExpressionAction>> ValueSpecificationAction is used in the context of a <<LoopNode>> LoopNode, it maps to an *Expression* (see 9.12).

9.11.4 References

SDL-2010 [ITU-T Z.101]:

12.2.1 Expressions and expressions as actual parameters

UML-SS [OMG UML]:

11.3.52 ValueSpecificationAction (from IntermediateActions)

9.12 LoopNode

The stereotype LoopNode extends the metaclass LoopNode with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-1.

A LoopNode represents a *Compound-node* in the SDL-2010 abstract syntax. It is equivalent to loop constructs (such as "for" or "while") of traditional programming languages.

9.12.1 Attributes

- stepGraphPart: ExecutableNode [0..*] {ordered}

  The ExecutableNode items to be execute after the body of the loop, normally to carry out such actions as stepping the loop variables.

9.12.2 Constraints

[1] A LoopNode shall have a name.

[2] The isTestedFirst attribute shall be true.

[3] Each item in the bodyPart shall be an Action or a StructuredActivityNode that is defined in this profile.

[4] The test part shall only consist of <<ExpressionAction>> ValueSpecificationAction items that represent an expression of the `Predefined::Boolean type`.

[5] The result property shall be empty.

[6] The bodyOutput property shall be empty.

[7] The loopVariableInput property shall be empty.

[8] Each item in the setupPart shall be an AddVariableValueAction node (to initialize variables including loop variables).

[9] The stepGraphPart shall only contain AddVariableValueAction actions or CallOperationAction actions.

9.12.3 Semantics

A LoopNode maps to a *Compound-node*. The name of the LoopNode maps to the *Connector-name* of the *Compound-node*.

The variable property maps to the *Variable-definition-**set*** of the *Compound-node* (see clause 9.21).

The setupPart maps to the *Init-graph-node* list of the *Compound-node*, defining the initialization of the loop.

Each <<ExpressionAction>> ValueSpecificationAction contained in the test part maps to an *Expression* of the *While-graph-node*.

The bodyPart maps to the *Transition* of the *Compound-node*.

If present, the ExecutableNode items of the stepGraphPart map to the *Step-graph-node* list.

9.12.4 References

SDL-2010 [ITU-T Z.102]:

11.14.1 Compound and loop statements

UML-SS [OMG UML]:

12.3.35 LoopNode (from CompleteStructuredActivities, StructuredActivities)

9.13 OpaqueAction

The stereotype OpaqueAction extends the metaclass OpaqueAction with multiplicity [1..1]. This stereotype is abstract and its metamodel diagram is defined in Figure 9-2.

This stereotype is introduced to ensure that every OpaqueAction is one of the subtypes: <<Break>> OpaqueAction or <<Continue>> OpaqueAction.

9.13.1 Attributes

No additional attributes.

9.13.2 Constraints

No additional constraints.

9.13.3 Semantics

The subtypes of the stereotype OpaqueAction give its semantics.

9.13.4 References

UML-SS [OMG UML]:

11.3.26 OpaqueAction (from BasicActions)

9.14 ResetAction

The stereotype ResetAction extends the metaclass SendSignalAction with multiplicity [0..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

A timer is cancelled with a reset action represented by a ResetAction stereotype. The reset action cancels a timer and removes any corresponding timer signals that are queued for the agent instance executing the timer.

9.14.1 Attributes

No additional attributes.

9.14.2 Constraints

[1] The signal property shall refer to a <<Timer>> Signal.

[2] The onPort property shall be empty.

[3] The number of ownedAttribute items of the referenced signal shall be equal to the number of argument items of the <<ResetAction>> SendSignalAction.

[4] The type of each argument shall be the same as the type of the corresponding ownedAttribute of the referenced signal.

9.14.3 Semantics

A <<ResetAction>> SendSignalAction maps to a *Reset-node*. The signal maps to the *Timer-Identifier* and the argument list maps to the *Expression* list.

9.14.4 References

SDL-2010 [ITU-T Z.101]:

11.15 Timer

UML-SS [OMG UML]:

11.3.45 SendSignalAction (from BasicActions)

9.15 Return

The stereotype Return is a subtype of the stereotype ActivityFinalNode. The metamodel diagram for the stereotype is defined in Figure 9-1.

A <<Return>> ActivityFinalNode represents the action to return from a procedure (in the SDL-2010 abstract grammar) to the point where the procedure was called.

9.15.1 Attributes

- value: SdlExpression [0..1]

  An expression that represents the return value of the operation.

9.15.2 Constraints

[1] The <<Return>> ActivityFinalNode shall be part of an <<Activity>> Activity that is used to define the behaviour associated with an <<Operation>> Operation.

[2] The value shall be empty if the <<Operation>> Operation does not return a value. Otherwise, the value shall match the return type of the <<Operation>> Operation.

9.15.3 Semantics

A <<Return>> ActivityFinalNode maps to an *Action-return-node* if the value property is empty, otherwise to a *Value-return-node*. If it maps to a *Value-return-node*, the value property defines the *Expression* in the *Value-return-node*.

9.15.4 References

SDL-2010 [ITU-T Z.101]:

11.12.2.4 Return

9.16 SequenceNode

The stereotype SequenceNode extends the metaclass SequenceNode with multiplicity [1..1]. The metamodel diagram for the stereotype is defined in Figure 9-1.

A sequence node is a sequence of actions and describes the body of a compound node.

9.16.1 Attributes

No additional attributes.

9.16.2 Constraints

[1] Each executableNode of a SequenceNode shall be an Action or a StructuredActivityNode that is defined in this profile.

9.16.3 Semantics

Mapping to a Compound-node

A <<SequenceNode>> SequenceNode maps to a *Compound-node*.

The name of the <<SequenceNode>> SequenceNode defines the *Connector-name* of the *Compound-node*.

The variable definitions contained in the variable property of the SequenceNode map to the *Variable-definition-**set*** of the *Compound-node* (see clause 9.21).

The actions contained in the executableNode property of the SequenceNode map to the various *Graph-node*s in the *Transition* that are contained in the *Compound-node*.

9.16.4 References

SDL-2010 [ITU-T Z.102]:

11.14.1 Compound and loop statements

UML-SS [OMG UML]:

12.3.47 SequenceNode (from StructuredActivities)

9.17 SendSignalAction

The stereotype SendSignalAction extends the metaclass SendSignalAction with multiplicity [0..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

A send signal action outputs a signal from the executing agent, optionally specifying the target agent and the port used to send the signal.

9.17.1 Attributes

No additional attributes.

9.17.2 Constraints

[1] The target property shall reference a ValuePin.

[2] The value of the target property shall consist of a ValueSpecification that represents a type that conforms to the type `Predefined::Pid`.

[3] The onPort property shall reference a Port of the container <<ActiveClass>> Class of the <<SendSignalAction>> SendSignalAction.

9.17.3 Semantics

A <<SendSignalAction>> SendSignalAction maps to an *Output-node*. The qualifiedName of signal property maps to the *Signal-identifier*. The target property maps to the *Signal-destination*. The onPort property maps to the *Direct-via*. The argument property maps to the *Actual-parameters* list.

9.17.4 References

SDL-2010 [ITU-T Z.101]:

11.13.4 Output

UML-SS [OMG UML]:

11.3.45 SendSignalAction (from BasicActions)

9.18 SetAction

The stereotype SetAction extends the metaclass SendSignalAction with multiplicity [0..1]. The metamodel diagram for the stereotype is defined in Figure 9-2.

The set action gives a timer an expiry time.

9.18.1 Attributes

- timeExpression: SdlExpression

  The time when the timer will expire.

9.18.2 Constraints

[1] The signal property shall refer to a <<Timer>> Signal.

[2] The onPort property shall be empty.

[3] The number of ownedAttribute items of the referenced signal shall be equal to the number of argument items of the <<SetAction>> SendSignalAction.

[4] The type of each argument shall be the same as the type of the corresponding ownedAttribute of the referenced signal.

[5] The type property of the timeExpression shall refer to the type `Predefined::Time`.

9.18.3 Semantics

A <<SetAction>> SendSignalAction maps to a *Set-node*. The signal maps to the *Timer-Identifier*. The argument list maps to the *Expression* list and timeExpression maps to *Time-expression*.

9.18.4 References

SDL-2010 [ITU-T Z.101]:

11.15 Timer

UML-SS [OMG UML]:

11.3.45 SendSignalAction (from BasicActions)

9.19 Stop

The stereotype Stop is a subtype of the stereotype ActivityFinalNode. The metamodel diagram for the stereotype is defined in Figure 9-1.

A stop represents the action to terminate the enclosing <<ActiveClass>> Class instance (the enclosing agent).

9.19.1 Attributes

No additional attributes.

9.19.2 Constraints

No additional constraints.

9.19.3 Semantics

A <<Stop>> ActivityFinalNode maps to a *Stop-node*.

9.19.4 References

SDL-2010 [ITU-T Z.101]:

11.12.2.3 Stop

9.20 TimerConstraint

The stereotype TimerConstraint extends the metaclass OpaqueExpression with a multiplicity of [0..1]. The metamodel diagram for the stereotype is defined in Figure 9-3.

A TimerConstraint represents a timer communication constraint for a remote procedure call. Hence, the application of a TimerConstraint is only possible in the context of a CallOperationAction (see clause 9.7).

NOTE – A direct mapping to the abstract syntax is not possible because in SDL-2010 a remote procedure call is transformed to an implicit exchange of signals.

9.20.1 Attributes

- timer: Signal [1]

  A reference to the timer that shall be monitored for expiry.

- variableList: Property [0..*] {ordered}

  References to the variables that shall receive values of the timer signal.

- connect: NamedElement [0..1]

  An optional reference to a labelled element that specifies where interpretation shall continue when the timer expires before the remote procedure call is finished.

9.20.2 Constraints

No additional constraints.

9.20.3 Semantics

A <<TimerConstraint>> OpaqueExpression has to be considered during the transformation of a CallOperationAction that represents a remote procedure call.

9.20.4 References

SDL-2010 [ITU-T Z.102]:

10.5 Remote procedure

UML-SS [OMG UML]:

7.3.36 OpaqueExpression (from Kernel)

9.21 Variable

The stereotype Variable extends the metaclass Variable with a multiplicity of [1]. The metamodel diagram for the stereotype is defined in Figure 9-3.

A Variable represents an SDL-2010 local variable definition within a loop or compound statement, which is only locally accessible. In SDL-UML, a Variable is usable only in the context of a LoopNode (see clause 9.12) or SequenceNode (see clause 9.16).

NOTE 1 – The stereotype Variable introduces the missing attribute aggregation, which is not supported by the UML metaclass Variable.

NOTE 2 – In contrast to the Property stereotype, the Variable stereotype cannot be used to specify SDL-2010 synonyms because this stereotype represents only local variable definitions (see clause 11.14.1 in [ITU-T Z.102]).

9.21.1 Attributes

- aggregation: AggregationKind [1]

  The aggregation kind of the variable. The default value is none.

9.21.2 Constraints

[1] The type shall be a <<DataTypeDefinition>> Class or an <<Interface>> Interface.

[2] If the upperValue is omitted, the lowerValue shall also be omitted.

[3] If the upperValue is included, the lowerValue shall also be included.

NOTE 1 – The upper and lower bounds of multiplicity are optional in UML-SS.

[4] If the upperValue value is greater than 1 and isOrdered is true, isUnique shall be false.

NOTE 2 – That is because there is not a predefined SDL-2010 data type that is ordered and requires each of its elements to have unique values.

[5] The aggregation of a Variable shall not be of kind shared.

9.21.3 Semantics

A <<Variable>> Variable maps to *Variable-definition*. The aggregation property maps to the *Aggregation-kind* of a *Variable-definition*. If the aggregation is of AggregationKind none, the *Aggregation-kind* is **REF**; otherwise, if the aggregation is of kind composite, the *Aggregation-kind* is **PART**.

NOTE – The aggregation kind '**PART**' is a feature of Basic SDL-2010 (see clause 12.3.1 of [ITU-T Z.101]), whereas the aggregation kind '**REF**' is introduced in [ITU-T Z.107] in order to support object-oriented data.

The name defines the *Variable-name*. The *Sort-reference-identifier* is the *Sort-identifier* of the sort derived from the type property. The *Sort-identifier* is determined as specified in clause 7.13.

9.21.4 References

SDL-2010 [ITU-T Z.101]:

12.3.1 Variable definition

SDL-2010 [ITU-T Z.102]:

11.14.1 Compound and loop statements

SDL-2010 [ITU-T Z.107]:

12.3.1 Variable definition

UML-SS [OMG UML]:

12.3.52 Variable (from StructuredActivities)

10 ValueSpecification

A value specification in SDL-UML is specified as a non-terminal expression or a literal value. An expression is a node in an expression tree that has a number (possibly zero) of operands that themselves specify values and therefore is expressions or literals. A value is represented textually and the syntax shall be a textual notation based on the concrete syntax of SDL-2010 or a notation provided by a tool supplier. Consequently, the components of an expression in SDL-UML usually have a one-to-one correspondence with respective SDL-2010 abstract syntax items that would result from analysing the text as SDL-2010.

In contrast to other parts of SDL-UML, value specification items are defined in terms of metaclasses that are direct or indirect subtypes of the UML ValueSpecification metaclass.

10.1 ValueSpecification metamodel diagrams

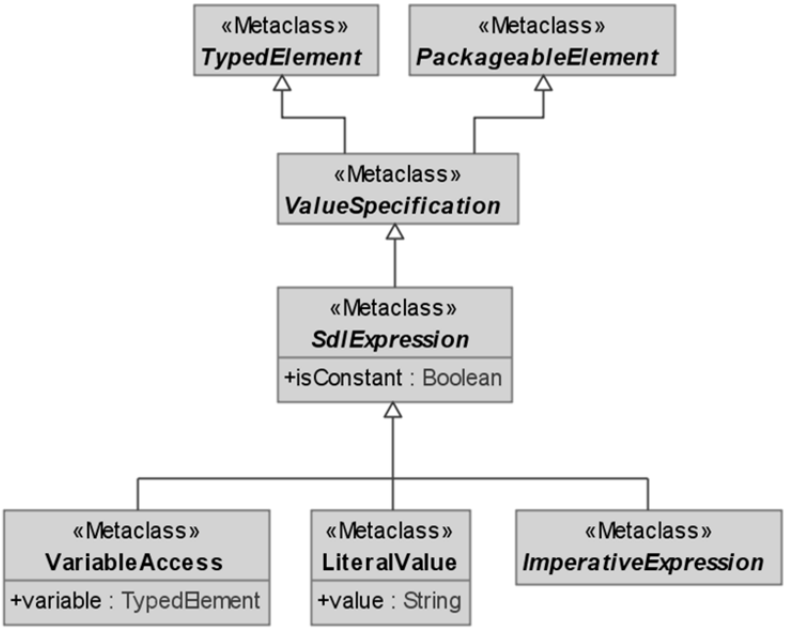

Figure 10-1 – SdlExpression

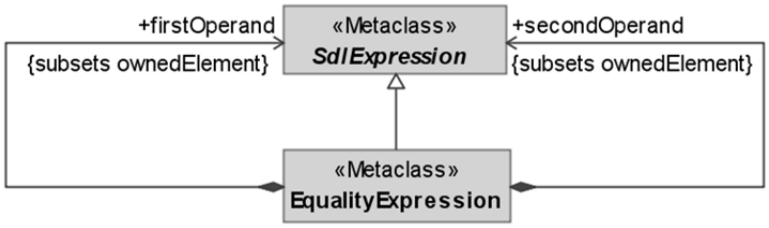

Figure 10-2 – EqualityExpression

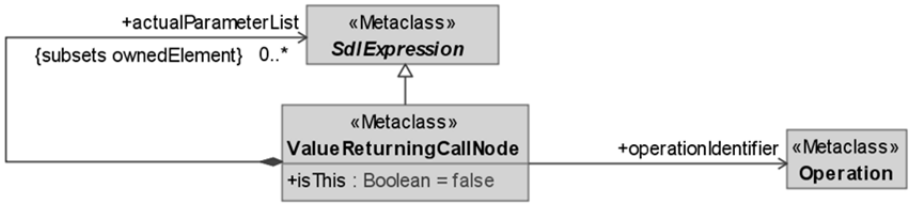

Figure 10-3 – ValueReturningCallNode

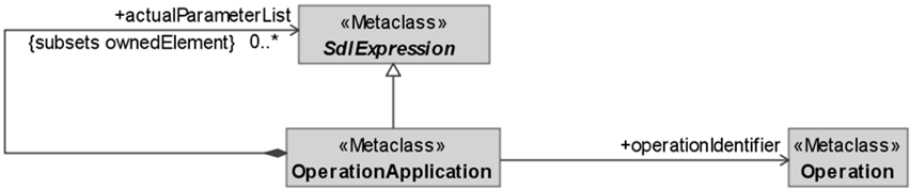

Figure 10-4 – OperationApplication

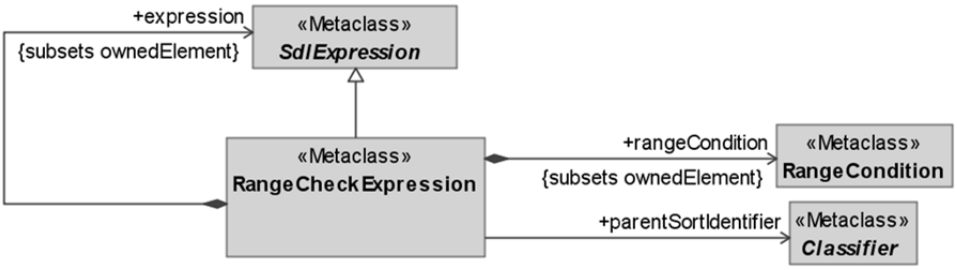

Figure 10-5 – RangeCheckExpression

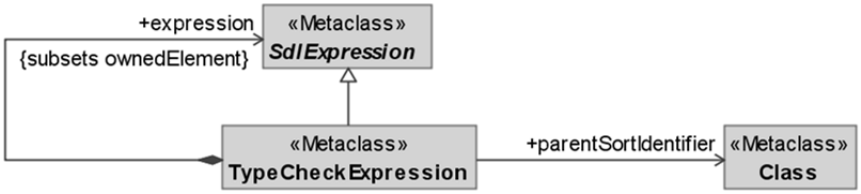

Figure 10-6 – TypeCheckExpression

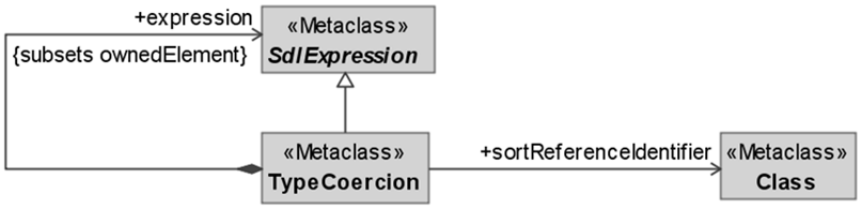

Figure 10-7 – TypeCoercion

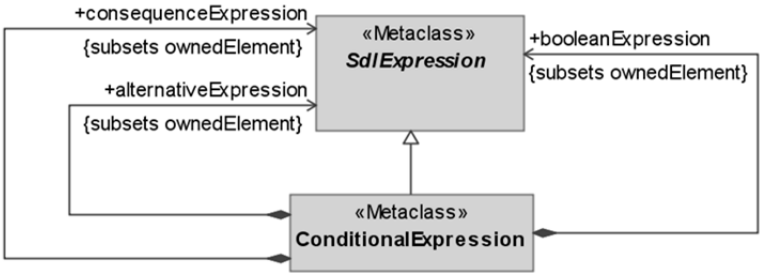

Figure 10-8 – ConditionalExpression

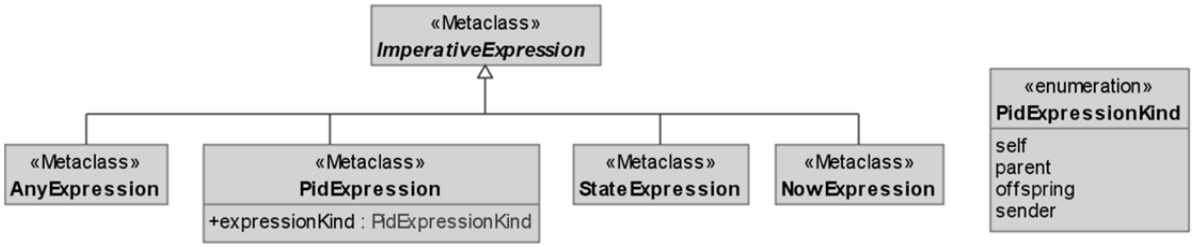

Figure 10-9 – ImperativeExpression

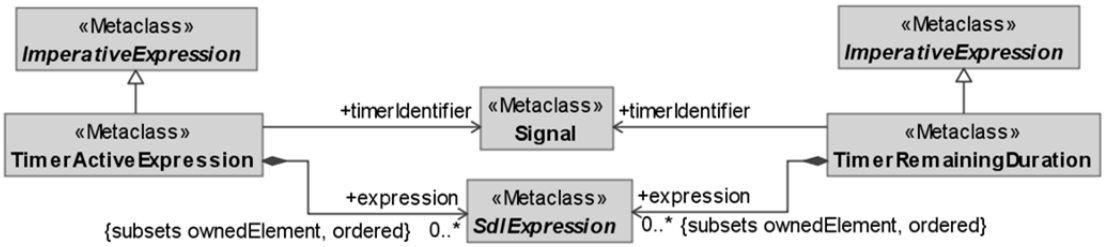

Figure 10-10 – TimerExpression

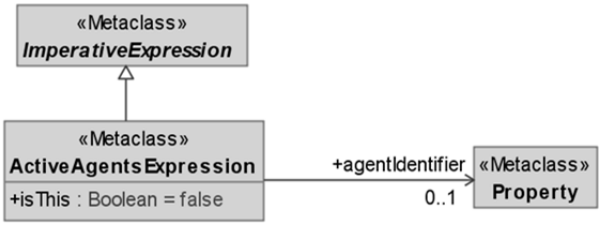

Figure 10-11 – ActiveAgentsExpression

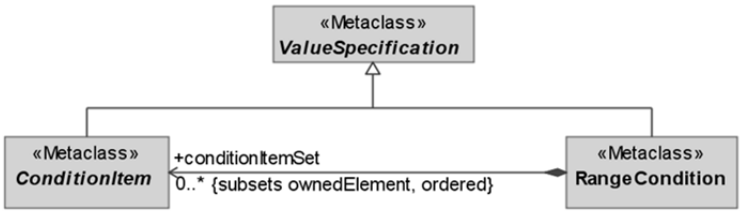

Figure 10-12 – RangeCondition

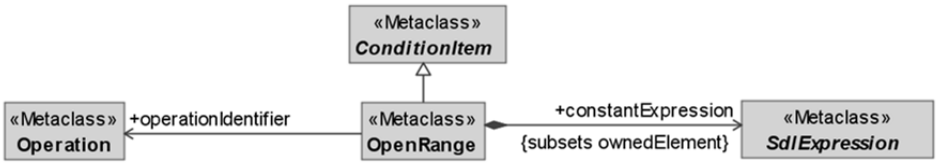

Figure 10-13 – OpenRange

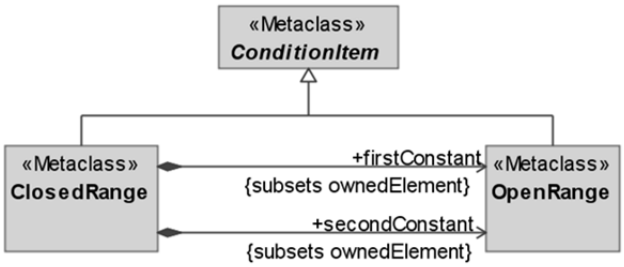

Figure 10-14 – ClosedRange

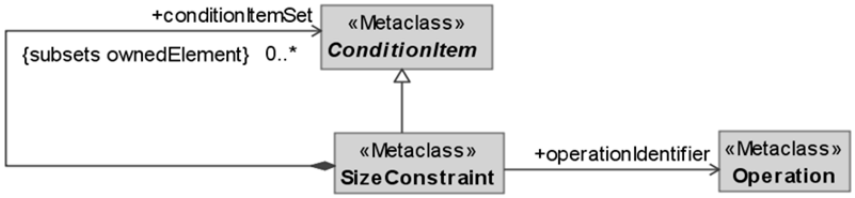

Figure 10-15 – SizeConstraint

10.2 ActiveAgentsExpression

The metaclass ActiveAgentsExpression is a specialization of the metaclass ImperativeExpression (see clause 10.8). The metamodel diagram for the metaclass is defined in Figure 10-11.

The ActiveAgentsExpression metaclass represents an *Active-agents-expression* of the abstract grammar of SDL-2010. Because an *Active-agents-expression* is one alternative of an *Imperative-expression*, it is also an active expression.

10.2.1 Attributes

- agentIdentifier: Property [0..1] {subsets Element::ownedElement}

  the agent instance set, for which the number of active agents is determined.

- isThis: Boolean

  if true, the number of active agents is determined for the enclosing active agent.

10.2.2 Constraints

[1] The type property shall reference the type `Predefined::Natural`.

[2] If isThis is true, the agentIdentifier shall be empty.

10.2.3 Semantics

The ActiveAgentsExpression metaclass maps to an *Active-agents-expression*. If isThis is false, the agentIdentifier property maps to the *Agent-identifier* of the *Active-agents-expression*. Otherwise, the optional **THIS** element is present in the *Active-agent-expression*.

10.2.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4.4 Active agents expression

10.3 AnyExpression

The metaclass AnyExpression is a specialization of the metaclass ImperativeExpression (see clause 10.8). The metamodel diagram for the metaclass is defined in Figure 10-9.

The AnyExpression metaclass represents an *Any-expression* of the abstract grammar of SDL-2010. Because an *Any-expression* is one alternative of an *Imperative-expression*, it is also an active expression.

10.3.1 Attributes

No additional attributes.

10.3.2 Constraints

[1] The type property shall reference a <<DataTypeDefinition>> Class or <<Interface>> Interface.

10.3.3 Semantics

The AnyExpression metaclass maps to an *Any-expression*. The type property maps to the *Sort-reference-identifier* of the *Any-expression*.

10.3.4 References

SDL-2010 [ITU-T Z.104]:

12.3.4.6 Any Expression

10.4 ClosedRange

The metaclass ClosedRange is a specialization of the metaclass ConditionItem (see clause 10.6). The metamodel diagram for the metaclass is defined in Figure 10-14.

A closed range condition constrains a data type with a lower and upper bound. Only if a value for such a constrained data type is within the specified boundaries, is the closed range condition fulfilled.

The ClosedRange maps to the *Closed-range* alternative of a *Condition-item* node in the SDL-2010 abstract syntax.

10.4.1 Attributes

- firstConstant: OpenRange {subsets Element::ownedElement}

  the lower bound value of a closed range.
- secondConstant: OpenRange {subsets Element::ownedElement}

  the upper bound value of a closed range.

10.4.2 Constraints

[1] The operationIdentifier properties of the firstConstant and secondConstant shall reference an operation with a signature as follows:

```
"<="(P, P): Predefined::Boolean
```

where the `P` in the operation signature is the type of the constrained data type.

[2] The type property shall be the type `Predefined::Boolean`.

10.4.3 Semantics

The metaclass ClosedRange maps to a *Closed-range*. The firstConstant property maps to the first *Open-range* and the secondConstant maps to the second *Open-range* of a *Closed-range*.

10.4.4 References

SDL-2010 [ITU-T Z.101]:

12.1.8.2 Constraint

10.5 ConditionalExpression

The metaclass ConditionalExpression is a specialization of the metaclass SdlExpression and its metamodel diagram is defined in Figure 10-8.

The ConditionalExpression metaclass represents a *Conditional-expression* in the SDL-2010 abstract syntax. A conditional expression consists of a Boolean expression, a consequence expression and an alternative expression. If the Boolean expression returns the value true, than the consequence expression is invoked. Otherwise, the alternative expression is invoked.

10.5.1 Attributes

- booleanExpression: SdlExpression (subsets Element::ownedElement)

  the Boolean expression for the decision.
- alternativeExpression: SdlExpression (subsets Element::ownedElement)

  the expression that is evaluated when the result of the decision is false.
- consequenceExpression: SdlExpression (subsets Element::ownedElement)

  the expression that is evaluated when the result of the decision is true.

10.5.2 Constraints

[1] The type of the booleanExpression shall be the type `Predefined::Boolean`.

[2] The alternativeExpression and the consequenceExpression shall have the same type.

[3] The type property of a ConditionalExpression shall be the type `Predefined::Boolean`.

10.5.3 Semantics

A ConditionalExpression maps to a *Conditional-expression*. The booleanExpression maps to the *Boolean-expression*, the consequenceExpression to the *Consequence-expression*, and the alternativeExpression to the *Alternative-expression* of a *Conditional-expression*. If one of the properties of a ConditionalExpression references SdlExpression with isConstant is false, the ConditionalExpression represents an *Active-expression*; otherwise it is a *Constant-Expression*.

10.5.4 References

SDL-2010 [ITU-T Z.101]:

12.2.5 Conditional expression

10.6 ConditionItem

The metaclass ConditionItem is a specialization of the UML metaclass ValueSpecification. This metaclass is abstract and the metamodel diagram for the metaclass is defined in Figure 10-12. Subtypes of this metaclass are the ClosedRange (see 10.4), OpenRange (see 10.11) and SizeConstraint (see 10.18) metaclasses. A ConditionItem is only usable in combination with a RangeCondition.

The ConditionItem metaclass represents a *Condition-item* in the SDL-2010 abstract syntax. The subtypes of this metaclass are mapped to the alternatives *Open-range*, *Closed-range* or *Size-constraint*.

10.6.1 Attributes

No additional attributes.

10.6.2 Constraints

[1] The type property of a ConditionItem shall be the type `Predefined::Boolean`.

[2] The owner of a ConditionItem shall be a RangeCondition.

10.6.3 Semantics

The subtypes of the ConditionItem metaclass define the semantics.

10.6.4 References

SDL-2010 [ITU-T Z.101]:

12.1.8.2 Constraint

UML-SS [OMG UML]:

7.3.55 ValueSpecification (from Kernel)

10.7 EqualityExpression

The metaclass EqualityExpression is a specialization of the metaclass SdlExpression. The metamodel diagram for the metaclass is defined in Figure 10-2.

The EqualityExpression metaclass represents an *Equality-expression* in the SDL-2010 abstract syntax. An equality expression consists of two operands, which are compared. If the result of both operands is equal, than the result of the equality expression is a Boolean value of true; otherwise, the result is a Boolean value of false.

10.7.1 Attributes

- firstOperand : SdlExpression {subsets Element::ownedElement}

  The left-hand expression to be compared for equality.

- secondOperand : SdlExpression {subsets Element::ownedElement}

  The right-hand expression to be compared for equality.

10.7.2 Constraints

[1] The type property of an EqualityExpression Expression shall be of the type `Predefined::Boolean`.

[2] The type of the firstOperand shall conform to the type of the secondOperand or vice versa.

10.7.3 Semantics

An EqualityExpression maps to an *Equality-expression*. The firstOperand maps to the *First-operand* and the secondOperand maps to the *Second-operand* of a *Conditional-expression*. If one of the properties of an EqualityExpression references an SdlExpression with isConstant is true, the EqualityExpression represents a *Constant-expression*; otherwise it is an *Active-Expression*.

10.7.4 References

SDL-2010 [ITU-T Z.101]:

12.2.4 Equality expression

10.8 ImperativeExpression

The metaclass ImperativeExpression is specialization of the metaclass SdlExpression. This metaclass is abstract, and the metamodel diagram for the metaclass is defined in Figure 10-9. Subtypes of ImperativeExpression are the metaclasses StateExpression (see 10.19), AnyExpression (see 10.3), PidExpression (see 10.13), NowExpression (see 10.10), TimerRemainingDuration (see 10.21), TimerActiveExpression (see 10.20) and ActiveAgentsExpression (see 10.2).

The ImperativeExpression metaclass represents an *Imperative-expression* of the abstract grammar of SDL-2010. In addition, an *Imperative-expression* is also an *Active-expression*. An imperative expression is used to access the system clock, special agent variables, the Pid of an agent or the status of timers.

10.8.1 Attributes

No additional attributes.

10.8.2 Constraints

[1] The isConstant property shall be true.

10.8.3 Semantics

A subtype of an ImperativeExpression is always mapped to the *Active-Expression* alternative of an *Expression*. Further semantics and mapping rules are specified in the context of the subtypes of the ImperativeExpression metaclass.

10.8.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4 Imperative expression

10.9 LiteralValue

The metaclass LiteralValue is a subtype of the metaclass SdlExpression. The metamodel diagram for the metaclass is defined in Figure 10-1.

The metaclass LiteralValue represents a *Literal* in the SDL-2010 abstract syntax. A literal represents a concrete value for a particular data type. In addition, a *Literal* is always a *Constant expression* in the SDL-2010 abstract syntax.

10.9.1 Attributes

- value: String

  This represents the concrete value for a data type.

10.9.2 Constraints

[1] The type property shall reference a <<DataTypeDefinition>> Class.

[2] The isConstant property shall be true.

10.9.3 Semantics

The LiteralValue maps to a *Literal* (a *Constant-expression*) in the SDL-2010 abstract syntax. The qualifiedName of the type property of a LiteralValue maps to the *Qualifier* part of the *Literal-identifier*. In addition, the value property of a LiteralValue maps to the *Name* part of the *Literal-identifier*.

10.9.4 References

SDL-2010 [ITU-T Z.101]:

12.2.2 Literal

10.10 NowExpression

The metaclass NowExpression is a subtype of the metaclass ImperativeExpression (see 10.8). The metamodel diagram for the metaclass is defined in Figure 10-9.

The NowExpression metaclass represents a *Now-expression* of the abstract grammar of SDL-2010. With a now expression the current value of the system clock is obtained. In consequence, the type of the result value is always the predefined `Time` type.

10.10.1 Attributes

No additional attributes.

10.10.2 Constraints

[1] The type property shall be the type `Predefined::Time`.

10.10.3 Semantics

The NowExpression metaclass maps to a *Now-expression*.

10.10.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4.1 Now expression

10.11 OpenRange

The metaclass OpenRange is a subtype of the metaclass ConditionItem (see 10.6). The metamodel diagram for the metaclass is defined in Figure 10-13.

An open range condition constrains a data type only with one boundary value (a constant expression) and an associated range operator (an infix operator). The OpenRange metaclass represents the *Open-range* alternative of a *Condition-item* node in the SDL-2010 abstract syntax.

10.11.1 Attributes

- operationIdentifier: Operation

  the operation (infix operator) for the range operator.
- constantExpression: SdlExpression {subsets Element::ownedElement}

  the boundary value of an open range.

10.11.2 Constraints

[1] The operationIdentifier property shall reference an <<Operation>> Operation with a result type of `Predefined::Boolean`.

[2] Each parameter of the referenced operationIdentifier shall the same type as the constrained data type.

[3] The constantExpression property shall only consist of an SdlExpression with isConstant = true.

[4] The type property of the OpenRange shall be the type `Predefined::Boolean`.

10.11.3 Semantics

The OpenRange metaclass maps to an *Open-range*. The operationIdentifier property maps to the *Operation-identifier* and the constantExpression maps to the *Constant-expression* of the *Open-range*. Further semantics is specified in clause 12.1.8.2 of [ITU-T Z.101]

10.11.4 References

SDL-2010 [ITU-T Z.101]:

12.1.8.2 Constraint

10.12 OperationApplication

The metaclass OperationApplication is a subtype of the metaclass SdlExpression. The metamodel diagram for the metaclass is defined in Figure 10-4.

An operation application represents the invocation of an operation of a <<DataTypeDefinition>> Class and maps to an *Operation-application*.

10.12.1 Attributes

- operationIdentifier: Operation

  Identifies the operation to be invoked.
- actualParameterList: SdlExpression [0..*] {subsets Element::ownedElement, ordered}

  the list of actual parameters for the operation application.

10.12.2 Constraints

[1] The type of each item in the actualParameterList shall conform to the type of the corresponding parameter of the operation.

[2] The operationIdentifier property shall identify an <<Operation>> Operation that is owned by a <<DataTypeDefinition>> Class.

[3] The type property of an OperationApplication shall be of the same type as the operation referenced by the operationIdentifier property.

[4] The isConstant property shall be true only if each element in the expression list has an isConstant property of true.

10.12.3 Semantics

The OperationApplication metaclass maps to an *Expression* that is an *Operation-application*. The qualifiedName property of the operationIdentifier maps to the *Operator-identifier*, and each SdlExpression in the actualParameterList maps to an *Expression* of the *Actual-parameters* list of the *Operation-application*.

If all elements in the actualParameterList are expressions with isConstant true, the OperationApplication represents a *Constant-expression*; otherwise, it represents an *Active-expression*.

10.12.4 References

SDL-2010 [ITU-T Z.101]:

12.2.6 Operation application

10.13 PidExpression

The metaclass PidExpression is a subtype of the metaclass ImperativeExpression (see 10.8). The metamodel diagram for the metaclass is defined in Figure 10-9.

The PidExpression metaclass represents a *Pid-expression* of the abstract grammar of SDL-2010. A pid expression accesses one out of four anonymous variables of an agent and returns the associated pid value. A pid expression is always an active expression because it is one alternative of the imperative expression.

10.13.1 Attributes

- expressionKind: PidExpressionKind

  Defines the kind of the pid expression

10.13.2 Constraints

[1] The type property of a PidExpression shall conform to the type `Predefined::Pid`.

10.13.3 Semantics

The PidExpression metaclass maps to one out of four alternatives of a *Pid-expression*. Depending on the value of the expressionKind property, a PidExpression maps to a *Self-expression*, a *Parent-expression*, an *Offspring-expression* or a *Sender-expression*.

10.13.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4.2 Pid expression

10.14 PidExpressionKind

The enumeration type PidExpressionKind determines the kind of a PidExpression. The metamodel diagram is defined in Figure 10-9.

10.14.1 Attributes

No additional attributes.

10.14.2 Constraints

No additional constraints.

10.14.3 Semantics

For the enumeration type PidExpressionKind the following enumeration literals and mappings are defined:

- self representing the *Self-expression*;
- parent representing the *Parent-expression*;
- offspring representing the *Offspring-expression*;
- sender representing the *Sender-expression*.

10.14.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4.2 Pid expression

10.15 RangeCheckExpression

The metaclass RangeCheckExpression is a specialization of the metaclass SdlExpression (see clause 10.17). The metamodel diagram for the metaclass is defined in Figure 10-5.

A range check expression is used in order to check if a given value or expression meets the range conditions criteria.

10.15.1 Attributes

- rangeCondition: RangeCondition {subsets Element::ownedElement}

  the range condition to be verified.
- expression: SdlExpression {subsets Element::ownedElement }

  the value to be verified.
- parentSortIdentifier: Classifier

  a reference to a <<DataTypeDefinition>> Class that defines the parent sort.

10.15.2 Constraints

[1] The type property of a RangeCheckExpression shall be the type `Predefined::Boolean`.

[2] The parentSortIdentifier shall reference a <<DataTypeDefinition>> Class.

[3] The type of the expression shall conform to type identified by the parentSortIdentifier.

10.15.3 Semantics

A RangeCheckExpression maps to a *Range-check-expression*. The rangeCondition property maps to the *Range-condition* and the expression property maps to the *Expression* of a *Range-check-expression*. If the expression property references an SdlExpression with isConstant true, the RangeCheckExpression represents a *Constant-expression*; otherwise it is an *Active-Expression*.

10.15.4 References

SDL-2010 [ITU-T Z.101]:

12.2.7 Range check expression

10.16 RangeCondition

The metaclass RangeCondition is a specialization of the UML metaclass ValueSpecification. The metamodel diagram for the metaclass is defined in Figure 10-12.

A range condition defines a set of values for a range check. A RangeCondition is used in the context of a RangeCheckExpression, a <<Syntype>> Class, a <<Property>> Property, a <<Decision>> ConditionalNode, an <<If>> ConditionalNode or a <<Pseudostate>> Pseudostate with kind choice.

10.16.1 Attributes

- conditionItemSet: ConditionItem [*] {subsets Element::ownedElement}

  References all condition items specified for a range condition.

10.16.2 Constraints

[1] The type property of a RangeCondition shall be the type `Predefined::Boolean`.

10.16.3 Semantics

A RangeCondition maps to a *Range-condition*. The conditionItemSet property maps to the *Condition-item-**set*** of the *Range-condition*.

10.16.4 References

SDL-2010 [ITU-T Z.101]:

12.1.8.2 Constraint

UML-SS [OMG UML]:

7.3.55 ValueSpecification (from Kernel)

10.17 SdlExpression

The metaclass SdlExpression is a specialization of the UML metaclass ValueSpecification. This metaclass is abstract, and the metamodel diagram for the metaclass is defined in Figure 10-1. Subtypes of the metaclass are the metaclasses RangeCheckExpression (see clause 10.15), ConditionalExpression (see clause 10.5), OperationApplication (see clause 10.12), ValueReturningCallNode (see clause 10.24), ImperativeExpression (see clause 10.8) and EqualityExpression (see clause 10.7).

Depending on the isConstant property, the SdlExpression metaclass represents a *Constant-expression* or an *Active-expression* alternative of an *Expression*. For both alternatives of an *Expression* in the SDL-2010 abstract syntax further alternatives exist. The semantics and mapping rules for the different alternatives of an *Expression* depends on the above listed subtypes of the SdlExpression metaclass.

10.17.1 Attributes

- isConstant: Boolean

  True if the expression is a constant expression, otherwise it is an active expression.

10.17.2 Constraints

[1] The type property of an SdlExpression shall not be empty.

10.17.3 Semantics

Its subtypes specify the semantics and mapping rules of an SdlExpression.

10.17.4 References

SDL-2010 [ITU-T Z.101]:

12.2.1 Expressions and expressions as actual parameters

UML-SS [OMG UML]:

7.3.55 ValueSpecification (from Kernel)

10.18 SizeConstraint

The metaclass SizeConstraint is a subtype of the metaclass ConditionItem (see clause 10.6). The metamodel diagram for the metaclass is defined in Figure 10-15. A size constraint is usable only to constrain multi-value data types that own a `length()` operator, e.g., the SDL-UML predefined `String` data type. The SizeConstraint metaclass maps to the *Size-constraint* alternative of the *Condition-item* node in the SDL-2010 abstract syntax.

10.18.1 Attributes

- operationIdentifier: Operation

  the `length()` operator for the verification of the size.
- conditionItemSet: ConditionItem [0..*] {subsets Element::ownedElement}

  references all condition items specified for a size range.

10.18.2 Constraints

[1] The operationIdentifier property shall reference an <<Operation>> Operation with a signature as follows: length(P): `Predefined::Natural`

where the `P` in the operation signature is the type of the constrained data type.

[2] The type property of a SizeConstraint shall be the type `Predefined::Boolean`.

10.18.3 Semantics

A SizeConstraint maps to a *Size-constraint* in the SDL-2010 abstract syntax. The operationIdentifier property maps to the *Operation-identifier* and the conditionItemSet maps to the *Condition-item-**set*** of the *Size-constraint*.

10.18.4 References

SDL-2010 [ITU-T Z.101]:

12.1.8.2 Constraint

10.19 StateExpression

The metaclass StateExpression is a subtype of the metaclass ImperativeExpression (see clause 10.8). The metamodel diagram for the metaclass is defined in Figure 10-9.

The StateExpression metaclass represents a *State-expression* of the abstract grammar of SDL-2010. A state expression returns the name of the most recently entered state in terms of a `Charstring`. Because a *State-expression* is one alternative of an *Imperative-expression*, it is also an active expression.

10.19.1 Attributes

No additional attributes.

10.19.2 Constraints

[1] The type property shall be the type `Predefined::Charstring`.

10.19.3 Semantics

A StateExpression maps to a *State-expression*.

10.19.4 References

SDL-2010 [ITU-T Z.104]:

12.3.4.7 State expression

10.20 TimerActiveExpression

The metaclass TimerActiveExpression is a subtype of the metaclass ImperativeExpression (see clause 10.8). The metamodel diagram for the metaclass is defined in Figure 10-10.

The TimerActiveExpression metaclass represents a *Timer-active-expression* of the abstract grammar of SDL-2010. A timer active expression returns the Boolean value true, if the associated timer is active. Otherwise, a Boolean value of false is returned. A timer active expression is always an active expression, because it is one alternative of the imperative expression.

10.20.1 Attributes

- timerIdentifier: Signal

  Reference to the associated timer

- expression: SdlExpression [0..*] {subsets Element::ownedElement}

  Expression list containing the actual parameters of the associated timer.

10.20.2 Constraints

[1] The type property shall be the type `Predefined::Boolean`.

[2] The type, the order and the number of items in the expression list shall match with the ownedAttribute items of the associated timer.

10.20.3 Semantics

A TimerActiveExpression maps to a *Timer-active-expression*. The timerIdentifier maps to *Timer-identifier* and the expression list maps to the *Expression-list*.

10.20.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4.3 Timer active expression and timer remaining duration

10.21 TimerRemainingDuration

The metaclass TimerRemainingDuration is a subtype of the metaclass ImperativeExpression (see clause 10.8). The metamodel diagram for the metaclass is defined in Figure 10-10.

The TimerRemainingDuration metaclass represents a *Timer-remaining-duration* of the abstract grammar of SDL-2010. A timer remaining duration returns a value of the predefined type `Duration`. The value is the time until the timer will expire. A timer remaining duration is always an active expression because it is one alternative of the imperative expression.

10.21.1 Attributes

- timerIdentifier: Signal

  Reference to the associated timer.

- expression: ValueSpecification [0..*] {subsets Element::ownedElement}

  Expression list containing the actual parameters of the associated timer.

10.21.2 Constraints

[1] The type property shall be of predefined `Duration` type.

[2] The type, the order and the number of items in the expression list shall match with the ownedAttribute items of the associated timer.

10.21.3 Semantics

A TimerRemainingDuration maps to a *Timer-remaining-duration*. The timerIdentifier maps to *Timer-identifier* and the expression list maps to the *Expression-list* of the *Timer-remaining-duration*.

10.21.4 References

SDL-2010 [ITU-T Z.101]:

12.3.4.3 Timer active expression and timer remaining duration

10.22 TypeCheckExpression

The metaclass TypeCheckExpression is a specialization of the metaclass SdlExpression (see clause 10.17). The metamodel diagram for the metaclass is defined in Figure 10-6.

A type check expression is used to check if the dynamic sort of an expression is sort compatible with the sort introduced by a referenced data type definition.

10.22.1 Attributes

- expression: SdlExpression {subsets Element::ownedElement}

  The expression whose dynamic sort shall be evaluated.

- parentSortIdentifier: Class

  the data type definition that shall be used for a type check.

10.22.2 Constraints

[1] The type property of a RangeCheckExpression shall be of the predefined Boolean type.

[2] The parentSortIdentifier shall reference a <<DataTypeDefinition>> Class or <<Interface>> Interface.

[3] The type of the expression shall conform to the type identified by the parentSortIdentifier.

10.22.3 Semantics

A TypeCheckExpression maps to a *Type-check-expression*. The parentSortIdentifier property maps to the *Parent-sort-identifier* and the expression property maps to the *Expression* of a *Type-check-expression*. If the expression property is an SdlExpression with isConstant true, the TypeCheckExpression represents a *Constant-expression*; otherwise it is an *Active-Expression*.

10.22.4 References

SDL-2010 [ITU-T Z.107]:

12.2.8 Range check expression

10.23 TypeCoercion

The metaclass TypeCoercion is a specialization of the metaclass SdlExpression (see clause 10.17). The metamodel diagram for the metaclass is defined in Figure 10-7.

Type coercion is used in order to change the dynamic sort of an expression.

10.23.1 Attributes

- expression: SdlExpression {subsets Element::ownedElement}

  The expression whose dynamic sort shall be changed.

- sortReferenceIdentifier: Class

  A reference to the data type definition that shall be used as the new dynamic sort.

10.23.2 Constraints

[1] The type and sortReferenceIdentifier properties shall reference the same data type definition.

[2] The sortReferenceIdentifier shall reference a <<DataTypeDefinition>> Class.

[3] The type property of the expression shall refer to the same data type definition as identified by the sortReferenceIdentifier, or it shall be a subtype of that data type definition.

10.23.3 Semantics

A TypeCoercion maps to a *Type-coercion*. The sortReferenceIdentifier property maps to the *Sort-reference-identifier* and the expression property maps to the *Expression* of a *Type-coercion*. If the expression property is an SdlExpression with isConstant true, the TypeCoercion represents a *Constant-expression*; otherwise it is an *Active-Expression*.

10.23.4 References

SDL-2010 [ITU-T Z.107]:

12.2.8.1 Type coercion

10.24 ValueReturningCallNode

The metaclass ValueReturningCallNode is a specialization of the metaclass SdlExpression (see clause 10.17). The metamodel diagram for the metaclass is defined in Figure 10-3.

A value returning procedure call is used to call a procedure that returns a value. The procedure has to be owned by an agent. Hence, in SDL-UML, a value returning procedure call is only used to invoke an <<Operation>> Operation that is owned by an <<ActiveClass>> Class.

10.24.1 Attributes

- operationIdentifier: Operation

  Identifies the procedure to be invoked.

- actualParameterList: SdlExpression [0..*]{subsets Element::ownedElement, ordered}

  the list of actual parameters for the procedure call.

10.24.2 Constraints

[1] The isConstant property shall be false.

[2] In the actualParameterList, the type of each item shall conform to the type of the corresponding parameter of the operation.

[3] The operationIdentifier property shall identify an <<Operation>> Operation that is owned by an <<ActiveClass>> Class.

[4] The type property of a ValueReturningCallNode shall be of the same type as the operation referenced by the operationIdentifier property.

10.24.3 Semantics

A ValueReturningCallNode maps to a *Value-returning-call-node*. The operationIdentifier property maps to the *Procedure-identifier* and each SdlExpression in the actualParameterList maps to an *Expression* of the *Actual-parameters* list of the *Value-returning-call-node*.

10.24.4 References

SDL-2010 [ITU-T Z.101]:

11.13.3 Procedure call

12.3.5 Value returning procedure call

10.25 VariableAccess

The metaclass VariableAccess specializes the metaclass SdlExpression. The metamodel diagram for the metaclass is defined in Figure 10-1.

The metaclass VariableAccess maps to a *Variable-access* in the SDL-2010 abstract syntax. The result of a variable access is the current value of a variable. A *Variable-access* is always an *Active expression* in the SDL-2010 abstract syntax.

10.25.1 Attributes

- variable : TypedElement

  References the Variable or Property that shall be accessed.

10.25.2 Constraints

[1] The variable property shall reference a Variable or a Property.

[2] The type of a VariableAccess shall conform to the type of the referenced Variable or Property.

[3] The isConstant property shall be false.

10.25.3 Semantics

A VariableAccess maps to a *Variable-access* (an *Active-expression*) and the variable property maps to the *Variable-identifier*.

10.25.4 References

SDL-2010 [ITU-T Z.101]:

12.3.2 Variable access

11 Context parameters

Context parameters enable the parameterization of definitions. While context parameters are similar to UML template parameters, their semantics in SDL-UML is aligned to SDL-2010. The metamodel for context parameters of SDL-UML is specified in terms of metaclasses that extend the UML metaclass Element.

NOTE – Remote variable context parameters and remote procedure context parameters are not supported because remote variables and remote procedures are not represented by particular SDL-UML elements.

11.1 Context parameter metamodel diagrams

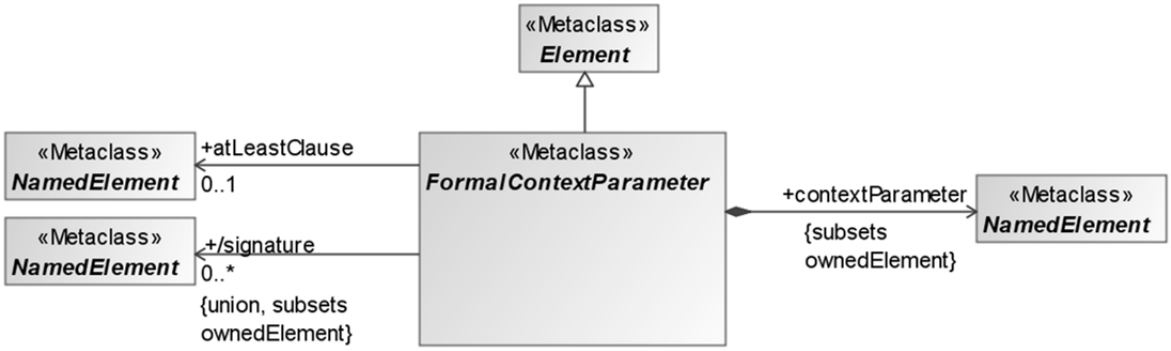

Figure 11-1 – FormalContextParameter

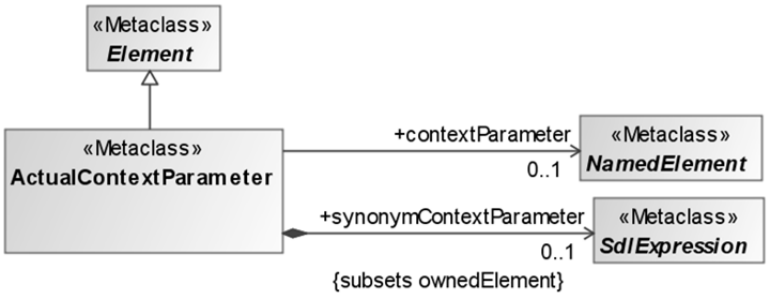

Figure 11-2 – ActualContextParameters

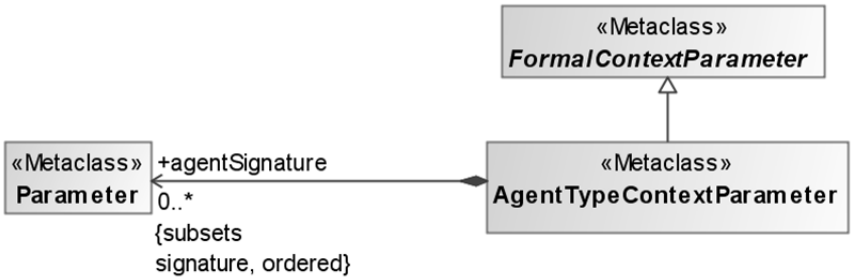

Figure 11-3 – AgentTypeContextParameter

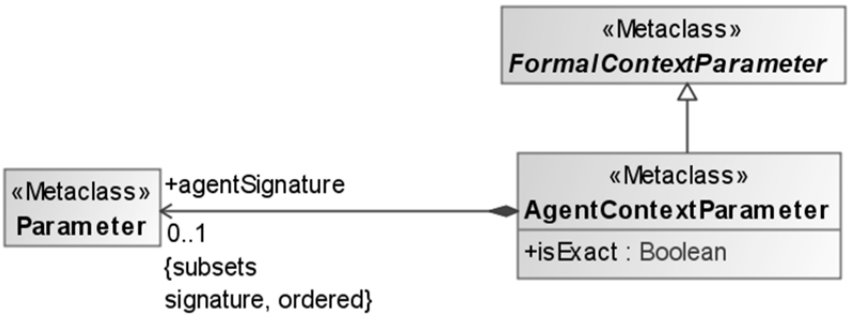

Figure 11-4 – AgentContextParameter

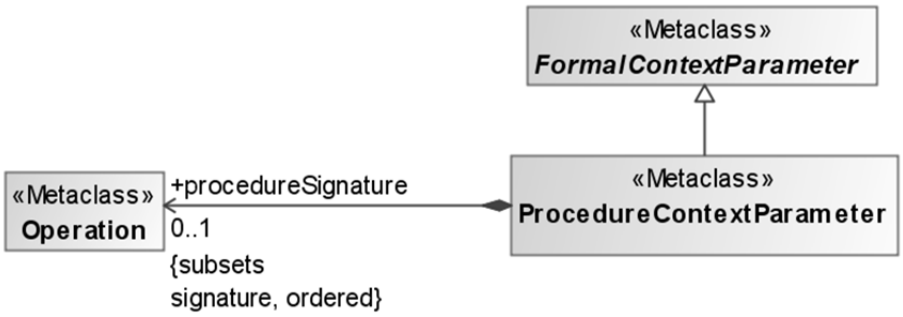

Figure 11-5 – ProcedureContextParameter

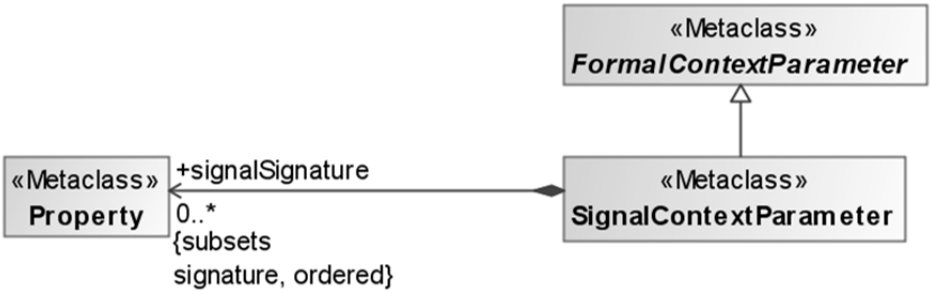

Figure 11-6 – SignalContextParameter

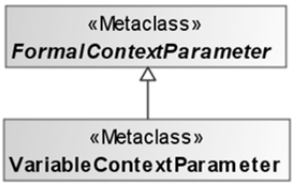

Figure 11-7 – VariableContextParameter

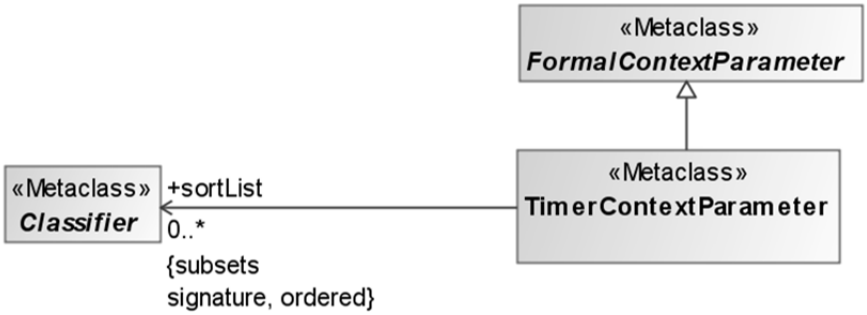

Figure 11-8 – TimerContextParameter

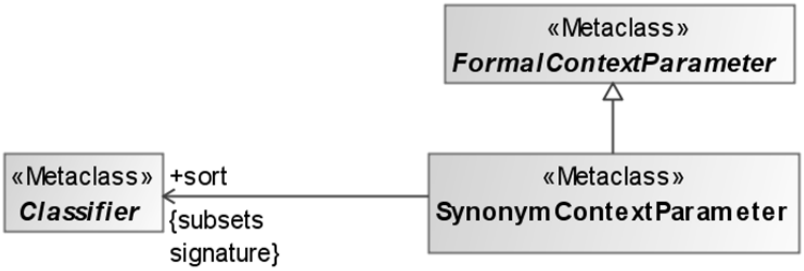

Figure 11-9 – SynonymContextParameter

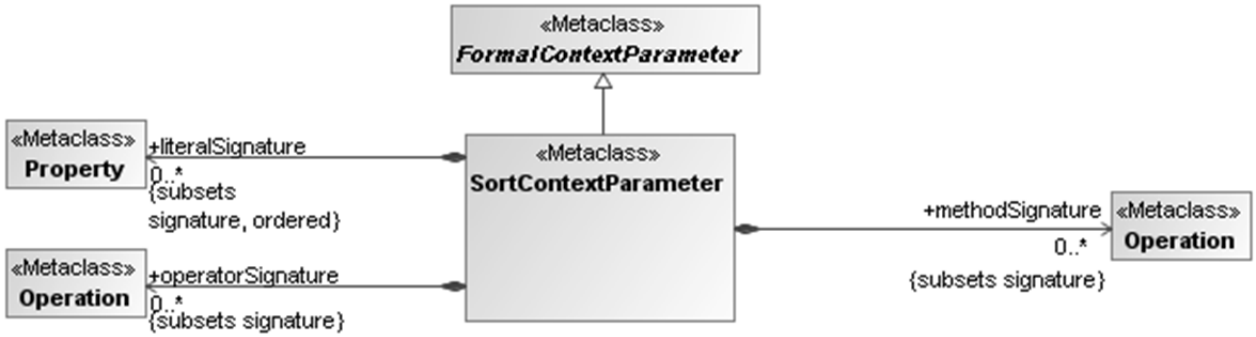

Figure 11-10 – SortContextParameter

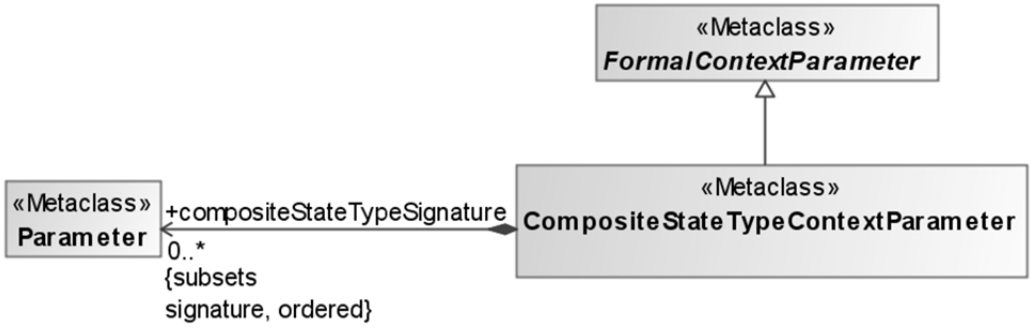

Figure 11-11 – CompositeStateTypeContextParameter

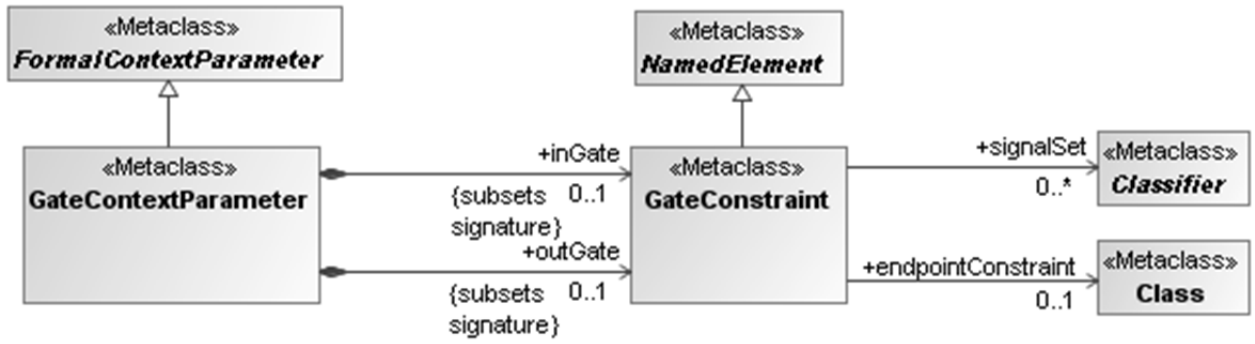

Figure 11-12 – GateContextParameter and GateConstraint

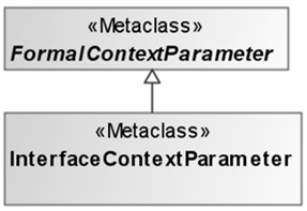

Figure 11-13 – InterfaceContextParameter

11.2 ActualContextParameter

The metaclass ActualContextParameter is a specialization of the UML metaclass Element. The metamodel diagram for the metaclass is defined in Figure 11-2.

The ActualContextParameter metaclass represents an actual context parameter of a parameterized type.

11.2.1 Attributes

- contextParameter: NamedElement [0..1]

  an actual context parameter corresponding to a formal context parameter that is not a SynonymContextParameter.

- synonymContextParameter: SdlExpression [0..1] {subsets Element::ownedElement}

  an actual context parameter corresponding to a formal context parameter that is SynonymContextParameter.

11.2.2 Constraints

[1] If the contextParameter is present, the synonymContextParameter shall be absent.

[2] A contextParameter shall satisfy the constraints for a corresponding actual context paramter defined in the corresponding formal context parameter.

NOTE 1 – A formal context parameter corresponds to an actual context parameter if it is in the same position in the formalContextParameterList of the supertype of the Classifier that owns the actualContextParameterList containing the actual context parameter as the position of that actual context parameter in its actualContextParameterList.

[3] If the synonymContextParameter is present, the contextParameter shall be absent.

[4] If the synonymContextParameter is present, its isConstant property shall be true.

[5] The synonymContextParameter shall conform to the type identified by the sort of the corresponding formal context parameter (see clause 8.3.9 of [ITU-T Z.102]).

NOTE 2 – An actual context parameter shall be of the same type or a subtype of the type identified by the atleast clause of the corresponding formal context parameter (see clause 8.3 of [ITU-T Z.102]).

11.2.3 Semantics

An ActualContextParameter is part of the context parameter concept that is described in clauses 8.1.2 and 8.3 of [ITU-T Z.102]. Before an SDL-UML element with context parameters is mapped to the SDL-2010 abstract syntax, a non-parameterized anonymous type definition is generated by expanding a parameterized type as specified in clause 8.1.2 of [ITU-T Z.102].

11.2.4 References

SDL-2010 [ITU-T Z.102]:

8.1.2 Type expression

8.3 Context parameters

8.3.9 Synonym context parameter

8.3.10 Sort context parameter

UML-SS [OMG UML]:

7.3.34 NamedElement (from Kernel, Dependencies)

11.3 AgentContextParameter

The metaclass AgentContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-4.

The metaclass AgentContextParameter represents an agent context parameter. An agent context parameter specifies parameterization by a process or block agent.

11.3.1 Attributes

- isExact: Boolean

  if true, the contextParameter of the corresponding actual context parameter shall be the type identified by the atLeastClause.

- agentSignature: Parameter [0..*]

  {subsets FormalContextParameter::signature, ordered}

  a <<Parameter>> Parameter list that defines the agent signature constraint.

11.3.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall reference an <<ActiveClass>> Class.

[2] If present, the atLeastClause shall refer to an <<ActiveClass>> Class that does not represent a system type (see clause 7.2.3).

NOTE – The atLeastClause property represents an agent type identifier in the concrete syntax of SDL-2010.

[3] If the isExact property is true, the type property of the corresponding actual context parameter and the atLeastClause shall refer to the same <<ActiveClass>> Class.

[4] If the isExact property is false, the type of the corresponding actual context parameter shall conform to the <<ActiveClass>> Class that is referenced by the atLeastClause.

[5] If agentSignature is not empty, the formal parameters of the <<ActiveClass>> Class identified by the type of the actual context parameter shall be compatible with the agentSignature.

11.3.3 Semantics

No additional semantics.

11.3.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.2 Agent context parameter

11.4 AgentTypeContextParameter

The metaclass AgentTypeContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-3.

The metaclass AgentTypeContextParameter represents an agent type context parameter. An agent type context parameter specifies parameterization by a process or block type.

11.4.1 Attributes

- agentSignature: Parameter [0..*]

  {subsets FormalContextParameter:: signature, ordered}

  a <<Parameter>> Parameter list that defines the agent signature constraint.

11.4.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall refer to an <<ActiveClass>> Class.

[2] If present, the atLeastClause shall refer to an <<ActiveClass>> Class that does not represent a system type (see clause 7.2.3).

[3] If agentSignature is not empty, the formal parameters of the actual context parameter shall be compatible with the agentSignature.

11.4.3 Semantics

No additional semantics.

11.4.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.1 Agent type context parameter

11.5 CompositeStateTypeContextParameter

The metaclass CompositeStateTypeContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-11.

The metaclass CompositeStateTypeContextParameter represents a composite state type context parameter. A composite state type context parameter specifies parameterization by a composite state type.

11.5.1 Attributes

- compositeStateTypeSignature: Parameter [0..*]

  {subsets FormalContextParameter:: signature, ordered}

  a <<Parameter>> Parameter list that defines the composite state type signature constraint.

11.5.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall refer to a <<StateMachine>> StateMachine.

[2] If present, the atLeastClause shall refer to a <<StateMachine>> StateMachine.

[3] If compositeStateTypeSignature is present, the ownedParameter list of the <<StateMachine>> StateMachine referenced by the corresponding actual context parameter shall conform to the compositeStateTypeSignature.

11.5.3 Semantics

No additional semantics.

11.5.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.11 Composite state type context parameter

11.6 FormalContextParameter

The metaclass FormalContextParameter is a specialization of the UML metaclass Element. This metaclass is abstract, and the metamodel diagram for the metaclass is defined in Figure 11–1.

The FormalContextParameter metaclass is the superClass of all metaclasses representing a specific formal context parameter kind.

When a FormalContextParameter is present in the definition of a Classifier, this Classifier is a parameterized type. A non-parameterized type is obtained as an expansion of a parameterized type

by providing corresponding (in order of occurrence) actual context parameters to replace the use of each formal context parameter in the parameterized type.

11.6.1 Attributes

- contextParameter: NamedElement {subsets Element::ownedElement}

  specifies the element that is used as the formal context parameter.

- atLeastClause: NamedElement [0..1]

  constrains the corresponding actual context parameter for the current formal context parameter.

- /signature: NamedElement [0..*] {union, subsets ::ownedElement}

  constrains the corresponding actual context parameter. This is a derived union.

NOTE – A formal context parameter in the concrete syntax of SDL-2010 optionally has a constraint, which is either an **atleast** constraint or a signature constraint. The atLeastClause represents an **atleast** constraint and the signature defines a signature constraint.

11.6.2 Constraints

[1] If the atLeastClause is present, the signature shall be absent.

[2] The contextParameter of the corresponding actual context parameter shall conform to the type identified by the atLeastClause (see clause 8.3 of [ITU-T Z.102]).

[3] If the signature is present, the atLeastClause shall be absent.

[4] The contextParameter of the corresponding actual context parameter shall be of a type that contains elements that meet the constraints for the elements identified in the signature.

NOTE 1 – An actual context parameter shall be of the same type or a subtype of the type identified by the **atleast** clause of the corresponding formal context parameter (see clause 8.3 of [ITU-T Z.102])

[5] A FormalContextParameter shall not be used as a supertype in a generalization and it shall not be used as an atLeastClause of another FormalContextParameter.

NOTE 2 – It is not allowed to use a formal context parameter as the base type in a type expression or in an **atleast** constraint of a formal context parameter (see clause 8.3 of [ITU-T Z.102]).

11.6.3 Semantics

Before an SDL-UML element having defined context parameters is mapped to the SDL-2010 abstract syntax, all formal context parameters have to be replaced by the corresponding actual context parameters as defined in clause 8.1.2 of [ITU-T Z.102].

11.6.4 References

SDL-2010 [ITU-T Z.102]:

8.1.2 Type expression

8.3 Context parameters

8.3.10 Sort context parameter

8.3.9 Synonym context parameter

UML-SS [OMG UML]:

7.3.34 NamedElement (from Kernel, Dependencies)

11.7 GateContextParameter

The metaclass GateContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-12.

The metaclass GateContextParameter represents a gate context parameter. A gate context parameter specifies parameterization by a gate.

11.7.1 Attributes

- inGate: GateConstraint[0..1] {subsets FormalContextParameter::signature}

  defines a list of signals that are receivable by a specific agent type.

- outGate: GateConstraint[0..1] {subsets FormalContextParameter::signature}

  defines a list of signals that a specific agent type is capable of sending.

11.7.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall reference a <<Port>> Port.

[2] The atLeastClause shall be empty.

[3] It is allowed to omit at most only one of the inGate or outGate properties.

[4] If inGate and outGate are present, both shall have the same endpointConstraint.

[5] If the inGate GateConstraint is present, the signals defined by its signalList (if present) shall contain all those signals defined by the required <<Interface>> Interface of the <<Port>> Port of the corresponding actual context parameter.

[6] If the outGate GateConstraint is present, the signals defined by its signalList shall be included in the set of signals defined by the provided <<Interface>> Interface of the <<Port>> Port that is the corresponding actual context parameter.

11.7.3 Semantics

No additional semantics.

11.7.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.12 Gate context parameter

11.8 GateConstraint

The metaclass GateConstraint is a subtype of the metaclass Element. The metamodel diagram for the metaclass is defined in Figure 11-12.

The metaclass GateConstraint represents a gate constraint of a gate context parameter.

11.8.1 Attributes

- signalSet: Classifier[0..*]

  defines a list of signals that are used to constrain the set of input or output signals of a port.

- endpointConstraint: Class[0..1]

  the source or destination for specified signals of a port.

11.8.2 Constraints

[1] Each item referenced in the signalSet shall be a <<Signal>> Signal or <<Interface>> Interface.

[2] The endpointConstraint property shall reference an <<ActiveClass>> Class.

11.8.3 Semantics

No additional semantics.

11.8.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.12 Gate context parameter

11.9 InterfaceContextParameter

The metaclass InterfaceContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-13.

The metaclass InterfaceContextParameter represents an interface context parameter. An interface context parameter specifies parameterization by an interface.

11.9.1 Attributes

No additional attributes.

11.9.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall refer to an <<Interface>>Interface.

[2] The signature shall be empty.

[3] If present, the atLeastClause shall refer to an <<Interface>> Interface.

[4] If the atLeastClause is present, the <<Interface>>Interface referenced by the corresponding actual context parameter shall conform to the <<Interface>> Interface of the atLeastClause.

11.9.3 Semantics

No additional semantics.

11.9.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.13 Interface context parameter

11.10 ProcedureContextParameter

The metaclass ProcedureContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-5.

The metaclass ProcedureContextParameter represents a procedure context parameter. A procedure context parameter specifies parameterization by a procedure.

11.10.1 Attributes

- procedureSignature: Operation[0..1]

  {subsets FormalContextParameter::signature, ordered}

  the <<Operation>> Operation that defines the procedure signature constraint.

11.10.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall refer to an <<Operation>> Operation.

[2] If present, the atLeastClause shall refer to an <<Operation>> Operation.

[3] Each ownedParameter of the <<Operation>> Operation that is the actual context parameter shall have the same type and the same aggregation as the corresponding ownedParameter of the procedureSignature, and (if present) both shall have the same type.

[4] Each ownedParameter that has a direction of out or inout of the <<Operation>> Operation that is the actual context parameter shall have the same type as the corresponding ownedParameter of the procedureSignature.

11.10.3 Semantics

No additional semantics.

11.10.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.3 Procedure context parameter

11.11 SignalContextParameter

The metaclass SignalContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-6.

The metaclass SignalContextParameter represents a signal context parameter. A signal context parameter specifies parameterization by a signal.

11.11.1 Attributes

- signalSignature [0..*] {subsets FormalContextParameter::signature, ordered}

  a list of items that define the signal signature constraint.

11.11.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall refer to a <<Signal>>Signal.

[2] If present, the atLeastClause shall refer to a <<Signal>> Signal.

[3] Each item of the signalSignature shall be a <<Property>> Property.

[4] If signalSignature is present, each ownedProperty of the <<Signal>> Signal that is the actual context parameter shall have the same type and the same aggregation as the corresponding <<Property>> Property of the signalSignature.

11.11.3 Semantics

No additional semantics.

11.11.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.5 Signal context parameter

11.12 SortContextParameter

The metaclass SortContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-10.

The metaclass SortContextParameter represents a sort context parameter. A sort context parameter specifies parameterization by a type.

11.12.1 Attributes

- literalSignature: Property[0..*]

  {subsets FormalContextParameter::signature, ordered}

  a list of literals that are a part of the sort signature.

- operatorSignature: Operation[0..*]

  {subsets FormalContextParameter::signature }

  a set of operation signatures for operators and that are a part of the sort signature.

- methodSignature: Operation[0..*]

  {subsets FormalContextParameter::signature }

  a set of operation signatures for methods and that are a part of the sort signature.

11.12.2 Constraints

[1] The contextParameter of the corresponding actual context parameter shall reference a <<DataTypeDefinition>> Class or <<Interface>> Interface.

[2] If present, the atLeastClause shall refer to a <<DataTypeDefinition>> Class or <<Interface>> Interface.

[3] If the signature is not empty, each item defined by the literalSignature, operatorSignature and methodSignature shall match with a corresponding item of the current actual context parameter.

11.12.3 Semantics

No additional semantics.

11.12.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.10 Sort context parameter

11.13 SynonymContextParameter

The metaclass SynonymContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-9.

The metaclass SynonymContextParameter represents a synonym context parameter. A synonym context parameter specifies parameterization by a constant value.

11.13.1 Attributes

- sort: Classifier {subsets FormalContextParameter::signature}

11.13.2 Constraints

[1] The corresponding actual context parameter shall be a synonymContextParameter.

[2] The atLeastClause shall be empty.

[3] The sort shall refer to a <<DataTypeDefinition>> Class or <<Interface>> Interface.

[4] The type property of an SdlExpression that is the actual context parameter and the sort property of a SynonymContextParameter shall reference the same type definition.

11.13.3 Semantics

No additional semantics.

11.13.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.9 Synonym context parameter

11.14 TimerContextParameter

The metaclass TimerContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-8.

The metaclass TimerContextParameter represents a timer context parameter. A timer context parameter specifies parameterization by a timer.

11.14.1 Attributes

- sortList: Classifier[0..*] {subsets FormalContextParameter::signature, ordered}

  a list of references to data type or interface definitions that constrain the timer used as the actual context parameter.

11.14.2 Constraints

[1] The contextParameter shall be a <<Timer>> Signal.

[2] The contextParameter of the corresponding actual context parameter shall refer to a <<Timer>> Signal.

[3] The atLeastClause shall be empty.

[4] If sortList is present, each item of the sortList shall refer to a <<DataTypeDefinition>> Class or <<Interface>> Interface.

[5] If sortList is present, each ownedProperty of the <<Timer>> Signal that is the actual context parameter shall have a type that is equal to the corresponding item of the sortList.

11.14.3 Semantics

No additional semantics.

11.14.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.8 Timer context parameter

11.15 VariableContextParameter

The metaclass VariableContextParameter is a subtype of the metaclass FormalContextParameter. The metamodel diagram for the metaclass is defined in Figure 11-7.

The metaclass VariableContextParameter represents a variable context parameter. A variable context parameter specifies parameterization by a variable.

11.15.1 Attributes

No additional attributes.

11.15.2 Constraints

[1] The contextParameter shall be a <<Property>> Property that represents a variable definition (see clause 7.13.3).

[2] The contextParameter of the corresponding actual context parameter shall refer to a <<Property>> Property that represents a variable definition (see clause 7.13.3).

[3] The atLeastClause and the signature shall be empty.

11.15.3 Semantics

No additional semantics.

11.15.4 References

SDL-2010 [ITU-T Z.102]:

8.3 Context parameters

8.3.6 Variable context parameter

12 Predefined data

This clause defines a set of predefined data types as a UML model library for SDL-UML. The data types are contained in a <<Package>> Package named Predefined and they are implicitly available in models with applied SDL-UML profile. In order to mark a data type definition as predefined, all <<DataTypeDefinition>> Classes specified in this clause have an isPredefined property of true.

The predefined data types are divided into non-parameterized types, which are used directly, and parameterized types, which need to have all their context parameters bound before they are usable.

The semantics of the data types and their provided operations are defined in clause 14 of [ITU-T Z.104]), except if a different semantics is explicitly mentioned below.

12.1 Non-parameterized data types

The non-parameterized data types of SDL-2010 are the following types: `Boolean`, `Integer`, `Natural`, `Character`, `String`, `Real`, `Duration`, `Time`, `Bit`, `Bitstring`, `Octet` and `Octetstring`. In SDL-UML, these data types are represented as instances of <<DataTypeDefinition>> Class or <<LiteralType>> Class or <<Syntype>> Class with names equal to those defined in SDL-2010.

12.1.1 Boolean

The predefined data type `Boolean` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.1 of [ITU-T Z.104].

12.1.2 Character

The predefined data type `Character` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.2 of [ITU-T Z.104].

12.1.3 Charstring

The predefined data type `Charstring` is represented as an instance of <<DataTypeDefinition>> Class that is a subtype of the parameterized `String` <<DataTypeDefinition>> Class.

The formalContextParameterList is empty.

The actualContextParameterList consists of:

- An ActualContextParameter with an empty synonymContextParameter and a contextParameter that refers to the Character <<DataTypeDefinition>> Class. This is a concrete binding to the formal context parameter `Itemsort` of `String`.

The SDL-UML data type definition `Charstring` provides the same operations as defined in clause 14.4 of [ITU-T Z.104].

12.1.4 Integer

The predefined data type `Integer` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.5 of [ITU-T Z.104].

12.1.5 Natural syntype

The predefined syntype `Natural` is represented as an instance of <<Syntype>> Class, which has a Dependency association to the constrained `Integer` <<LiteralType>> Class. The constant property of the `Natural` <<Syntype>> Class consists of a RangeCheckExpression representing the concrete syntax expression **`constants`** `>= 0` as defined in clause 14.6 of [ITU-T Z.104].

12.1.6 Real

The predefined data type `Real` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.7 of [ITU-T Z.104].

12.1.7 Duration

The predefined data type `Duration` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.11 of [ITU-T Z.104].

12.1.8 Time

The predefined data type `Time` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.12 of [ITU-T Z.104].

12.1.9 Bit

The predefined data type `Bit` is represented as an instance of <<LiteralType>> Class that is a subtype of Boolean <<LiteralType>> Class. The SDL-UML data type definition `Bit` provides the same literals and operations as defined in clause 14.14 of [ITU-T Z.104].

12.1.10 Bitstring

The predefined data type `Bitstring` is represented as an instance of <<LiteralType>> Class. This SDL-UML data type definition provides the same literals and operations as defined in clause 14.14 of [ITU-T Z.104].

12.1.11 Octet syntype

The predefined syntype `Octet` is represented as an instance of <<Syntype>> Class, which has a Dependency association to the constrained `Bitstring` <<LiteralType>> Class. The constant property of the Octet <<Syntype>> Class consists of a RangeCheckExpression representing the concrete syntax expression **size** = 8 as defined in clause 14.15 of [ITU-T Z.104].

12.1.12 Octetstring

The predefined data type `Octetstring` is represented as an instance of <<DataTypeDefinition>> Class that is a subtype of the parameterized `String` <<DataTypeDefinition>> Class.

The formalContextParameterList is empty.

The actualContextParameterList consists of:

- An ActualContextParameter with an empty synonymContextParameter and a contextParameter that is a reference to the `Octet` <<DataTypeDefinition>> Class. This is a concrete binding to the formal context parameter `Itemsort` of `String`.

The SDL-UML data type definition `Octetstring` provides the same operations as defined in clause 14.15 of [ITU-T Z.104].

12.2 Parameterized data types

This clause provides parameterized data types for SDL-UML predefined types with context parameters. Each of these parameterized data types is an instance of <<DataTypeDefinition>> Class that provides a set of formal context parameters as required for the particular type represented.

12.2.1 Array

The predefined data type `Array` is represented as an instance of <<DataTypeDefinition>> Class.

The formalContextParameterList consists of:

- A SortContextParameter with a contextParameter that is a <<DataTypeDefinition>> Class with the name `Index`.
- A SortContextParameter with a contextParameter that is a <<DataTypeDefinition>> Class with the name `Itemsort`.

The actualContextParameterList is empty.

The SDL-UML data type definition `Array` provides the same operations as defined in clause 14.8 of [ITU-T Z.104].

12.2.2 Bag

The predefined data type `Bag` is represented as an instance of <<DataTypeDefinition>> Class.

The formalContextParameterList consists of:

- A SortContextParameter with a contextParameter that is a <<DataTypeDefinition>> Class with the name `Itemsort`.

The actualContextParameterList is empty.

The SDL-UML data type definition `Bag` provides the same operations as defined in clause 14.13 of [ITU-T Z.104].

12.2.3 Powerset

The predefined data type `Powerset` is represented as an instance of <<DataTypeDefinition>> Class.

The formalContextParameterList consists of:

- A SortContextParameter with a contextParameter that is a <<DataTypeDefinition>> Class with the name `Itemsort`.

The actualContextParameterList is empty.

The SDL-UML data type definition `Powerset` provides the same operations as defined in clause 14.10 of [ITU-T Z.104].

12.2.4 String

The predefined data type `String` is represented as an instance of <<DataTypeDefinition>> Class.

The formalContextParameterList consists of:

- A SortContextParameter with a contextParameter that is a <<DataTypeDefinition>> Class with the name `Itemsort`.

The actualContextParameterList is empty.

The SDL-UML data type definition `String` provides the same operations as defined in clause 14.3 of [ITU-T Z.104].

12.2.5 Vector

The predefined data type `Vector` is represented as an instance of <<DataTypeDefinition>> Class that is a subtype of the parameterized `Array` <<DataTypeDefinition>> Class.

The formalContextParameterList consists of:

- A SortContextParameter with a contextParameter that is a <<DataTypeDefinition>> Class with the name `Itemsort`.
- A SynonymContextParameter with a contextParameter that is a <<Property>> Property with the name `MaxIndex`.

The actualContextParameterList property consists of:

- An ActualContextParameter with an empty synonymContextParameter and a sortContextParameter that is a reference to the `Indexsort` <<DataTypeDefinition>> Class. This is a binding to the formal context parameter `Index` of `Array`.
- An ActualContextParameter with an empty synonymContextParameter and a sortContextParameter that is a reference to the `Itemsort` <<DataTypeDefinition>> Class. This is a binding to the formal context parameter `Itemsort` of `Array`.

In addition, the `Vector` <<DataTypeDefinition>> Class owns the `Indexsort` <<Syntype>> Class as a nestedClassifier. The constant property of the `Indexsort` <<Syntype>> Class consists of a RangeCheckExpression representing the concrete syntax expression **`constants`** `1:MaxIndex` as defined in clause 14.9 of [ITU-T Z.104].

12.3 Pid

The predefined data type `Pid` is represented as an instance of <<Interface>> Interface. This SDL-UML data type definition is the supertype of all interface types (see clause 14.16 of [ITU-T Z.104]).

Bibliography

[b-ITU-T T.50] Recommendation ITU-T T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange*.

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**

Printed in Switzerland
Geneva, 2012