UML 2.3 Metamodel

Infrastructure

Package InfrastructureLibrary

Nested Package Summary
Core
Profiles

Package InfrastructureLibrary::Core

Nesting Package:

InfrastructureLibrary

Nested Package Summary
Abstractions
Basic
Constructs
PrimitiveTypes

Package InfrastructureLibrary::Core::Abstractions

Nesting Package:

Core

Imported Packages:

PrimitiveTypes

Nested Package Summary
BehavioralFeatures
Changeabilities
Classifiers
Comments
Constraints
Elements
Expressions
Generalizations
Instances
Literals
Multiplicities
MultiplicityExpressions
Namespaces
Ownerships
Redefinitions
Relationships
StructuralFeatures
Super
TypedElements
Visibilities

Package InfrastructureLibrary::Core::Abstractions::BehavioralFeatures

Nesting Package:

Abstractions

Imported Packages:

Classifiers, TypedElements

Class Summary
BehavioralFeature
Parameter

Association Summary
A_parameter_behavioralFeature

Package InfrastructureLibrary::Core::Abstractions::BehavioralFeatures

Class BehavioralFeature

A behavioral feature is a feature of a classifier that specifies an aspect of the behavior of its instances.

Generalizations:

Feature, Namespace

Owned Association Ends

+ /**parameter** : Parameter [0..*] {ordered, readOnly, union, subsets member}

Specifies the parameters of the BehavioralFeature.

Operations

+ **isDistinguishableFrom** (n : NamedElement, ns : Namespace) : Boolean [1..1] {query}

The query isDistinguishableFrom() determines whether two BehavioralFeatures may coexist in the same Namespace. It specifies that they have to have different signatures.

body (OCL): result = if n.oclIsKindOf(BehavioralFeature) then if ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->notEmpty() then Set{}->including(self)->including(n)->isUnique(bf | bf.parameter->collect(type)) else true endif else true endif

Package InfrastructureLibrary::Core::Abstractions::BehavioralFeatures

Class Parameter

A parameter is a specification of an argument used to pass information into or out of an invocation of a behavioral feature.

Generalizations:

NamedElement, TypedElement

Package InfrastructureLibrary::Core::Abstractions::BehavioralFeatures

Association A_parameter_behavioralFeature

Member Ends:

parameter, behavioralFeature

Owned Association Ends

+ **behavioralFeature** : BehavioralFeature [0..1]

Package InfrastructureLibrary::Core::Abstractions::Changeabilities

Nesting Package:

Abstractions

Imported Packages:

StructuralFeatures

Class Summary
StructuralFeature

Package InfrastructureLibrary::Core::Abstractions::Changeabilities

Class StructuralFeature

StructuralFeature has an attribute that determines whether a client may modify its value.

Attributes

+ **isReadOnly** : Boolean [1..1] = false

States whether the feature's value may be modified by a client.

Package InfrastructureLibrary::Core::Abstractions::Classifiers

Nesting Package:

Abstractions

Imported Packages:

Namespaces, Ownerships

Class Summary
Classifier
Feature

Association Summary
A_feature_featuringClassifier

Package InfrastructureLibrary::Core::Abstractions::Classifiers

Class Classifier

A classifier is a classification of instances - it describes a set of instances that have features in common.

Generalizations:

Namespace

Owned Association Ends

+ /**feature** : Feature [0..*] {readOnly, union, subsets member}

Specifies each feature defined in the classifier.

Operations

+ **allFeatures** () : Feature [0..*] {query}

The query allFeatures() gives all of the features in the namespace of the classifier. In general, through mechanisms such as inheritance, this will be a larger set than feature.

body (OCL): result = member->select(oclIsKindOf(Feature))

Package InfrastructureLibrary::Core::Abstractions::Classifiers

Class Feature

A feature declares a behavioral or structural characteristic of instances of classifiers.

Generalizations:

NamedElement

Specializations:

BehavioralFeature, StructuralFeature

Owned Association Ends

+ /**featuringClassifier** : Classifier [0..*] {readOnly, union}

The Classifiers that have this Feature as a feature.

Package InfrastructureLibrary::Core::Abstractions::Classifiers

Association A_feature_featuringClassifier

Member Ends:

feature, featuringClassifier

Package InfrastructureLibrary::Core::Abstractions::Comments

Nesting Package:

Abstractions

Imported Packages:

Ownerships

Class Summary
Comment

Association Summary
A_annotatedElement_comment

Package InfrastructureLibrary::Core::Abstractions::Comments

Class Comment

A comment is a textual annotation that can be attached to a set of elements.

Generalizations:

Element

Attributes

+ **body** : String [0..1]

Specifies a string that is the comment

Owned Association Ends

+ **annotatedElement** : Element [0..*]

References the Element(s) being commented.

Package InfrastructureLibrary::Core::Abstractions::Comments

Association A_annotatedElement_comment

Member Ends:

annotatedElement, comment

Owned Association Ends

+ **comment** : Comment [0..*]

Package InfrastructureLibrary::Core::Abstractions::Constraints

Nesting Package:

Abstractions

Imported Packages:

Expressions, Namespaces, Ownerships

Class Summary
Constraint
NamedElement
Namespace

Association Summary
A_constrainedElement_constraint
A_member_namespace
A_ownedMember_namespace
A_ownedRule_context
A_specification_owningConstraint

Package InfrastructureLibrary::Core::Abstractions::Constraints

Class Constraint

A constraint is a condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

Generalizations:

NamedElement

Owned Association Ends

+ **constrainedElement** : Element [0..*] {ordered}

The ordered set of Elements referenced by this Constraint.

+ **context** : Namespace [0..1] {subsets namespace}

The Namespace that owns this NamedElement.

+ **specification** : ValueSpecification [1..1] {subsets ownedElement}

A condition that must be true when evaluated in order for the constraint to be satisfied.

Constraints

not_apply_to_self

A constraint cannot be applied to itself.

expression (OCL): not constrainedElement->includes(self)

value_specification_boolean

The value specification for a constraint must evaluate to a Boolean value.

expression (OCL): self.specification().booleanValue().isOclKindOf(Boolean)

Package InfrastructureLibrary::Core::Abstractions::Constraints

Class NamedElement

A named element is an element in a model that may have a name.

Generalizations:

Element

Specializations:

Constraint, Namespace

Owned Association Ends

+ /**namespace** : Namespace [0..1] {readOnly, union, subsets owner}

Specifies the namespace that owns the NamedElement.

Package InfrastructureLibrary::Core::Abstractions::Constraints

Class Namespace

A namespace can own constraints. A constraint associated with a namespace may either apply to the namespace itself, or it may apply to elements in the namespace.

Generalizations:

NamedElement

Owned Association Ends

+ /**member** : NamedElement [0..*] {readOnly, union}

A collection of NamedElements identifiable within the Namespace, either by being owned or by being introduced by importing or inheritance.

+ /**ownedMember** : NamedElement [0..*] {readOnly, union, subsets ownedElement, subsets member}

A collection of NamedElements owned by the Namespace.

+ **ownedRule** : Constraint [0..*] {subsets ownedMember}

Specifies a set of Constraints owned by this Namespace.

Package InfrastructureLibrary::Core::Abstractions::Constraints

Association A_constrainedElement_constraint

Member Ends:

constrainedElement, constraint

Owned Association Ends

+ **constraint** : Constraint [0..*]

Package InfrastructureLibrary::Core::Abstractions::Constraints

Association A_member_namespace

Member Ends:

member, namespace

Owned Association Ends

+ **namespace** : Namespace [0..*]

Package InfrastructureLibrary::Core::Abstractions::Constraints

Association A_ownedMember_namespace

Member Ends:

ownedMember, namespace

Package InfrastructureLibrary::Core::Abstractions::Constraints

Association A_ownedRule_context

Member Ends:

ownedRule, context

Package InfrastructureLibrary::Core::Abstractions::Constraints

Association A_specification_owningConstraint

Member Ends:

specification, owningConstraint

Owned Association Ends

+ **owningConstraint** : Constraint [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::Elements

Nesting Package:

Abstractions

Class Summary
Element

Package InfrastructureLibrary::Core::Abstractions::Elements

Class Element

An element is a constituent of a model.

Specializations:

MultiplicityElement

Package InfrastructureLibrary::Core::Abstractions::Expressions

Nesting Package:

Abstractions

Imported Packages:

Ownerships

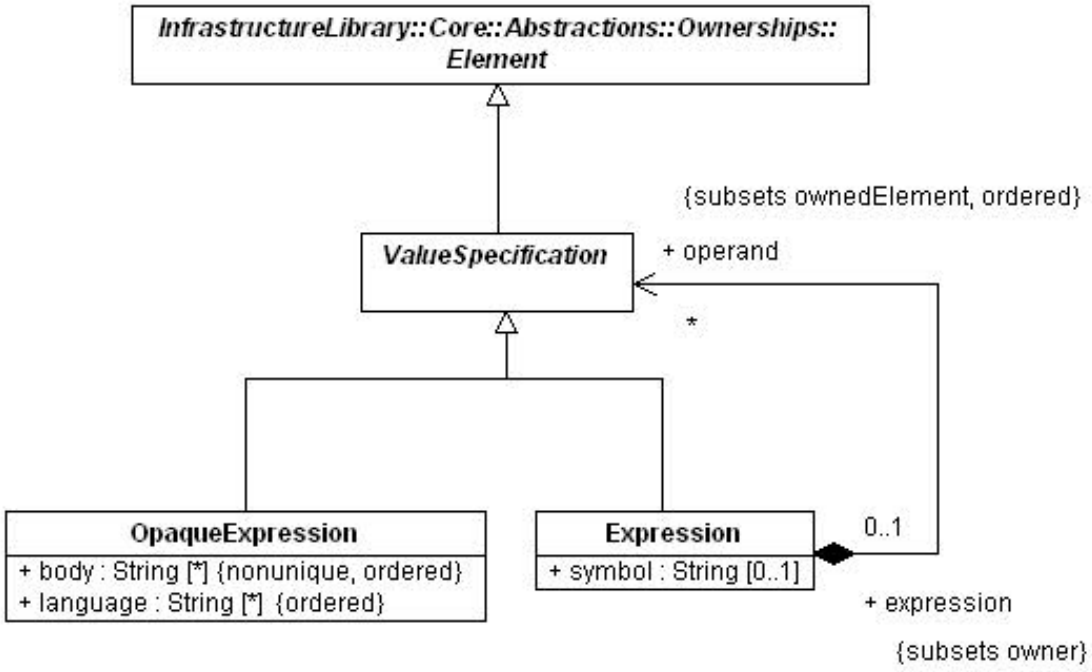
Diagram Summary
Abstractions Expressions

Class Summary
Expression
OpaqueExpression
ValueSpecification

Association Summary
A_operand_expression

Package InfrastructureLibrary::Core::Abstractions::Expressions

Diagram Abstractions Expressions



Classifiers Local to Package:

Expression, OpaqueExpression, ValueSpecification

Classifiers External to Package:

Element

Package InfrastructureLibrary::Core::Abstractions::Expressions

Class Expression

An expression is a structured tree of symbols that denotes a (possibly empty) set of values when evaluated in a context.

Generalizations:

ValueSpecification

Found in Diagrams:

Abstractions Expressions

Attributes

+ **symbol** : String [0..1]

The symbol associated with the node in the expression tree.

Owned Association Ends

+ **operand** : ValueSpecification [0..*] {ordered, subsets ownedElement}

Specifies a sequence of operands.

Package InfrastructureLibrary::Core::Abstractions::Expressions

Class OpaqueExpression

An opaque expression is an uninterpreted textual statement that denotes a (possibly empty) set of values when evaluated in a context.

Generalizations:

ValueSpecification

Found in Diagrams:

Abstractions Expressions

Attributes

+ **body** : String [0..*] {ordered, nonunique}

The text of the expression, possibly in multiple languages.

+ **language** : String [0..*] {ordered}

Specifies the languages in which the expression is stated. The interpretation of the expression body depends on the languages. If the languages are unspecified, they might be implicit from the expression body or the context. Languages are matched to body strings by order.

Package InfrastructureLibrary::Core::Abstractions::Expressions

Class ValueSpecification

A value specification is the specification of a (possibly empty) set of instances, including both objects and data values.

Generalizations:

Element

Specializations:

Expression, InstanceValue, LiteralSpecification, OpaqueExpression

Found in Diagrams:

Abstractions Expressions

Operations

+ **booleanValue** () : Boolean [1..1] {query}

The query booleanValue() gives a single Boolean value when one can be computed.

body (OCL): result = Set{}

+ **integerValue** () : Integer [1..1] {query}

The query integerValue() gives a single Integer value when one can be computed.

body (OCL): result = Set{}

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() determines whether a value specification can be computed in a model. This operation cannot be fully defined in OCL. A conforming implementation is expected to deliver true for this operation for all value specifications that it can compute, and to compute all of those for which the operation is true. A conforming implementation is expected to be able to compute the value of all literals.

body (OCL): result = false

+ **isNull** () : Boolean [1..1] {query}

The query isNull() returns true when it can be computed that the value is null.

body (OCL): result = false

+ **stringValue** () : String [1..1] {query}

The query stringValue() gives a single String value when one can be computed.

body (OCL): result = Set{}

+ **unlimitedValue** () : UnlimitedNatural [1..1] {query}

The query unlimitedValue() gives a single UnlimitedNatural value when one can be computed.

Package InfrastructureLibrary::Core::Abstractions::Expressions

Class ValueSpecification

body (OCL): result = Set{}

Package InfrastructureLibrary::Core::Abstractions::Expressions

Association A_operand_expression

Member Ends:

operand, expression

Found in Diagrams:

Abstractions Expressions

Owned Association Ends

+ **expression** : Expression [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::Generalizations

Nesting Package:

Abstractions

Imported Packages:

Relationships, Super, TypedElements

Class Summary
Classifier
Generalization

Association Summary
A_general_classifier
A_general_generalization
A_generalization_specific

Package InfrastructureLibrary::Core::Abstractions::Generalizations

Class Classifier

A classifier is a type and can own generalizations, thereby making it possible to define generalization relationships to
other classifiers.

Generalizations:

Type

Owned Association Ends

+ /**general** : Classifier [0..*]

Specifies the general Classifiers for this Classifier.

+ **generalization** : Generalization [0..*] {subsets ownedElement}

Specifies the Generalization relationships for this Classifier. These Generalizations navigate to more general classifiers in the generalization hierarchy.

Operations

+ **conformsTo** (other : Classifier) : Boolean [1..1] {query}

The query conformsTo() gives true for a classifier that defines a type that conforms to another. This is used, for example, in the specification of signature conformance for operations.

body (OCL): result = (self=other) or (self.allParents()->includes(other))

+ **general** () : Classifier [0..*] {query}

The general classifiers are the classifiers referenced by the generalization relationships.

body (OCL): result = self.parents()

+ **parents** () : Classifier [0..*] {query}

The query parents() gives all of the immediate ancestors of a generalized Classifier.

body (OCL): result = generalization.general

Package InfrastructureLibrary::Core::Abstractions::Generalizations

Class Generalization

A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an instance of the general classifier. Thus, the specific classifier indirectly has features of the more general classifier.

Generalizations:

DirectedRelationship

Owned Association Ends

+ **general** : Classifier [1..1] {subsets target}

References the general classifier in the Generalization relationship.

+ **specific** : Classifier [1..1] {subsets source, subsets owner}

References the specializing classifier in the Generalization relationship.

Package InfrastructureLibrary::Core::Abstractions::Generalizations

Association A_general_classifier

Member Ends:

general, classifier

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package InfrastructureLibrary::Core::Abstractions::Generalizations

Association A_general_generalization

Member Ends:

general, generalization

Owned Association Ends

+ **generalization** : Generalization [0..*]

Package InfrastructureLibrary::Core::Abstractions::Generalizations

Association A_generalization_specific

Member Ends:

generalization, specific

Package InfrastructureLibrary::Core::Abstractions::Instances

Nesting Package:

Abstractions

Imported Packages:

Expressions, StructuralFeatures

Class Summary
InstanceSpecification
InstanceValue
Slot

Association Summary
A_classifier_instanceSpecification
A_definingFeature_slot
A_instance_instanceValue
A_slot_owningInstance
A_specification_owningInstanceSpec
A_value_owningSlot

Package InfrastructureLibrary::Core::Abstractions::Instances

Class InstanceSpecification

An instance specification is a model element that represents an instance in a modeled system.

Generalizations:

NamedElement

Owned Association Ends

+ **classifier** : Classifier [0..*]

The classifier or classifiers of the represented instance. If multiple classifiers are specified, the instance is classified by all of them.

+ **slot** : Slot [0..*] {subsets ownedElement}

A slot giving the value or values of a structural feature of the instance. An instance specification can have one slot per structural feature of its classifiers, including inherited features. It is not necessary to model a slot for each structural feature, in which case the instance specification is a partial description.

+ **specification** : ValueSpecification [0..1] {subsets ownedElement}

A specification of how to compute, derive, or construct the instance.

Constraints

no_duplicate_slots

One structural feature (including the same feature inherited from multiple classifiers) is the defining feature of at most one slot in an instance specification.

expression (OCL): classifier->forAll(c | (c.allFeatures()->forAll(f | slot->select(s | s.definingFeature = f)->size() <= 1))

slots_are_defined

The defining feature of each slot is a structural feature (directly or inherited) of a classifier of the instance specification.

expression (OCL): slot->forAll(s | classifier->exists(c | c.allFeatures()->includes(s.definingFeature)))

Package InfrastructureLibrary::Core::Abstractions::Instances

Class InstanceValue

An instance value is a value specification that identifies an instance.

Generalizations:

ValueSpecification

Owned Association Ends

+ **instance** : InstanceSpecification [1..1]

The instance that is the specified value.

Package InfrastructureLibrary::Core::Abstractions::Instances

Class Slot

A slot specifies that an entity modeled by an instance specification has a value or values for a specific structural feature.

Generalizations:

Element

Owned Association Ends

+ **definingFeature** : StructuralFeature [1..1]

The structural feature that specifies the values that may be held by the slot.

+ **owningInstance** : InstanceSpecification [1..1] {subsets owner}

The instance specification that owns this slot.

+ **value** : ValueSpecification [0..*] {ordered, subsets ownedElement}

The value or values corresponding to the defining feature for the owning instance specification.

Package InfrastructureLibrary::Core::Abstractions::Instances

Association A_classifier_instanceSpecification

Member Ends:

classifier, instanceSpecification

Owned Association Ends

+ **instanceSpecification** : InstanceSpecification [0..*]

Package InfrastructureLibrary::Core::Abstractions::Instances

Association A_definingFeature_slot

Member Ends:

definingFeature, slot

Owned Association Ends

+ **slot** : Slot [0..*]

Package InfrastructureLibrary::Core::Abstractions::Instances

Association A_instance_instanceValue

Member Ends:

instance, instanceValue

Owned Association Ends

+ **instanceValue** : InstanceValue [0..*]

Package InfrastructureLibrary::Core::Abstractions::Instances

Association A_slot_owningInstance

Member Ends:

slot, owningInstance

Package InfrastructureLibrary::Core::Abstractions::Instances

Association A_specification_owningInstanceSpec

Member Ends:

specification, owningInstanceSpec

Owned Association Ends

+ **owningInstanceSpec** : InstanceSpecification [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::Instances

Association A_value_owningSlot

Member Ends:

value, owningSlot

Owned Association Ends

+ **owningSlot** : Slot [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::Literals

Nesting Package:

Abstractions

Imported Packages:

Expressions

Class Summary
LiteralBoolean
LiteralInteger
LiteralNull
LiteralSpecification
LiteralString
LiteralUnlimitedNatural

Package InfrastructureLibrary::Core::Abstractions::Literals

Class LiteralBoolean

A literal Boolean is a specification of a Boolean value.

Generalizations:

LiteralSpecification

Attributes

+ **value** : Boolean [1..1] = false

The specified Boolean value.

Operations

+ **booleanValue** () : Boolean [1..1] {query}

The query booleanValue() gives the value.

body (OCL): result = value

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

Package InfrastructureLibrary::Core::Abstractions::Literals

Class LiteralInteger

A literal integer is a specification of an integer value.

Generalizations:

LiteralSpecification

Attributes

+ **value** : Integer [1..1] = 0

The specified Integer value.

Operations

+ **integerValue** () : Integer [1..1] {query}

The query integerValue() gives the value.

body (OCL): result = value

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

Package InfrastructureLibrary::Core::Abstractions::Literals

Class LiteralNull

A literal null specifies the lack of a value.

Generalizations:

LiteralSpecification

Operations

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

+ **isNull** () : Boolean [1..1] {query}

The query isNull() returns true.

body (OCL): result = true

Package InfrastructureLibrary::Core::Abstractions::Literals

Class LiteralSpecification

A literal specification identifies a literal constant being modeled.

Generalizations:

ValueSpecification

Specializations:

LiteralBoolean, LiteralInteger, LiteralNull, LiteralString, LiteralUnlimitedNatural

Package InfrastructureLibrary::Core::Abstractions::Literals

Class LiteralString

A literal string is a specification of a string value.

Generalizations:

LiteralSpecification

Attributes

+ **value** : String [0..1]

The specified String value.

Operations

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

+ **stringValue** () : String [1..1] {query}

The query stringValue() gives the value.

body (OCL): result = value

Package InfrastructureLibrary::Core::Abstractions::Literals

Class LiteralUnlimitedNatural

A literal unlimited natural is a specification of an unlimited natural number.

Generalizations:

LiteralSpecification

Attributes

+ **value** : UnlimitedNatural [1..1] = 0

The specified UnlimitedNatural value.

Operations

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

+ **unlimitedValue** () : UnlimitedNatural [1..1] {query}

The query unlimitedValue() gives the value.

body (OCL): result = value

Package InfrastructureLibrary::Core::Abstractions::Multiplicities

Nesting Package:

Abstractions

Imported Packages:

Elements

Class Summary
MultiplicityElement

Package InfrastructureLibrary::Core::Abstractions::Multiplicities

Class MultiplicityElement

A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. A multiplicity element embeds this information to specify the allowable cardinalities for an instantiation of this element.

Generalizations:

Element

Attributes

+ **isOrdered** : Boolean [1..1] = false

For a multivalued multiplicity, this attribute specifies whether the values in an instantiation of this element are sequentially ordered.

+ **isUnique** : Boolean [1..1] = true

For a multivalued multiplicity, this attributes specifies whether the values in an instantiation of this element are unique.

+ **lower** : Integer [0..1] = 1

Specifies the lower bound of the multiplicity interval.

+ **upper** : UnlimitedNatural [0..1] = 1

Specifies the upper bound of the multiplicity interval.

Operations

+ **includesCardinality** (C : Integer) : Boolean [1..1] {query}

The query includesCardinality() checks whether the specified cardinality is valid for this multiplicity.

precondition (): upperBound()->notEmpty() and lowerBound()->notEmpty() includesCardinality = (lowerBound() <= C) and (upperBound() >= C)

body (OCL): result = (lowerBound() <= C) and (upperBound() >= C)

+ **includesMultiplicity** (M : MultiplicityElement) : Boolean [1..1] {query}

The query includesMultiplicity() checks whether this multiplicity includes all the cardinalities allowed by the specified multiplicity.

precondition (): self.upperBound()->notEmpty() and self.lowerBound()->notEmpty() and M.upperBound()->notEmpty() and M.lowerBound()->notEmpty()

body (OCL): result = (self.lowerBound() <= M.lowerBound()) and (self.upperBound() >= M.

Package InfrastructureLibrary::Core::Abstractions::Multiplicities

Class MultiplicityElement

upperBound())

+ **isMultivalued** () : Boolean [1..1] {query}

The query isMultivalued() checks whether this multiplicity has an upper bound greater than one.

precondition (): upperBound()->notEmpty()

body (OCL): result = upperBound() > 1

+ **lowerBound** () : Integer [1..1] {query}

The query lowerBound() returns the lower bound of the multiplicity as an integer.

body (OCL): result = if lower->notEmpty() then lower else 1 endif

+ **upperBound** () : UnlimitedNatural [1..1] {query}

The query upperBound() returns the upper bound of the multiplicity for a bounded multiplicity as an unlimited natural.

body (OCL): result = if upper->notEmpty() then upper else 1 endif

Constraints

lower_ge_0

The lower bound must be a non-negative integer literal.

expression (OCL): lowerBound()->notEmpty() implies lowerBound() >= 0

upper_ge_lower

The upper bound must be greater than or equal to the lower bound.

expression (OCL): (upperBound()->notEmpty() and lowerBound()->notEmpty()) implies upperBound() >= lowerBound()

Package InfrastructureLibrary::Core::Abstractions::MultiplicityExpressions

Nesting Package:

Abstractions

Imported Packages:

Expressions, Multiplicities

Class Summary
MultiplicityElement

Association Summary
A_lowerValue_owningLower
A_upperValue_owningUpper

Package InfrastructureLibrary::Core::Abstractions::MultiplicityExpressions

Class MultiplicityElement

MultiplicityElement supports the use of value specifications to define each bound of the multiplicity.

Generalizations:

Element

Attributes

+ /**lower** : Integer [0..1] = 1

Specifies the lower bound of the multiplicity interval.

+ /**upper** : UnlimitedNatural [0..1] = 1

Specifies the upper bound of the multiplicity interval.

Owned Association Ends

+ **lowerValue** : ValueSpecification [0..1] {subsets ownedElement}

The specification of the lower bound for this multiplicity.

+ **upperValue** : ValueSpecification [0..1] {subsets ownedElement}

The specification of the upper bound for this multiplicity.

Operations

+ **lower** () : Integer [1..1] {query}

The derived lower attribute must equal the lowerBound.

body (OCL): result = lowerBound()

+ **lowerBound** () : Integer [1..1] {query}

The query lowerBound() returns the lower bound of the multiplicity as an integer.

body (OCL): result = if lowerValue->isEmpty() then 1 else lowerValue.integerValue() endif

+ **upper** () : UnlimitedNatural [1..1] {query}

The derived upper attribute must equal the upperBound.

body (OCL): result = upperBound()

+ **upperBound** () : UnlimitedNatural [1..1] {query}

Package InfrastructureLibrary::Core::Abstractions::MultiplicityExpressions

Class MultiplicityElement

The query upperBound() returns the upper bound of the multiplicity as an unlimited natural.

body (OCL): result = if upperValue->isEmpty() then 1 else upperValue.unlimitedValue() endif

Package InfrastructureLibrary::Core::Abstractions::MultiplicityExpressions

Association A_lowerValue_owningLower

Member Ends:

lowerValue, owningLower

Owned Association Ends

+ **owningLower** : MultiplicityElement [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::MultiplicityExpressions

Association A_upperValue_owningUpper

Member Ends:

upperValue, owningUpper

Owned Association Ends

+ **owningUpper** : MultiplicityElement [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Nesting Package:

Abstractions

Imported Packages:

Ownerships

Class Summary
NamedElement
Namespace

Association Summary
A_member_namespace
A_ownedMember_namespace

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Class NamedElement

A named element is an element in a model that may have a name.

Generalizations:

Element

Specializations:

Feature, InstanceSpecification, Namespace, Parameter, RedefinableElement, Type, TypedElement

Attributes

+ **name** : String [0..1]

The name of the NamedElement.

+ /**qualifiedName** : String [0..1] {readOnly}

A name which allows the NamedElement to be identified within a hierarchy of nested Namespaces. It is constructed from the names of the containing namespaces starting at the root of the hierarchy and ending with the name of the NamedElement itself.

Owned Association Ends

+ /**namespace** : Namespace [0..1] {readOnly, union, subsets owner}

Specifies the namespace that owns the NamedElement.

Operations

+ **allNamespaces** () : Namespace [0..*] {ordered, query}

The query allNamespaces() gives the sequence of namespaces in which the NamedElement is nested, working outwards.

body (OCL): result = if self.namespace->isEmpty() then Sequence{} else self.namespace.allNamespaces()->prepend(self.namespace) endif

+ **isDistinguishableFrom** (n : NamedElement, ns : Namespace) : Boolean [1..1] {query}

The query isDistinguishableFrom() determines whether two NamedElements may logically co-exist within a Namespace. By default, two named elements are distinguishable if (a) they have unrelated types or (b) they have related types but different names.

body (OCL): result = if self.oclIsKindOf(n.oclType) or n.oclIsKindOf(self.oclType) then ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->isEmpty() else true endif

+ **qualifiedName** () : String [1..1] {query}

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Class NamedElement

When there is a name, and all of the containing namespaces have a name, the qualified name is constructed from the names of the containing namespaces.

body (OCL): result = if self.name->notEmpty() and self.allNamespaces()->select(ns | ns.name->isEmpty())->isEmpty() then self.allNamespaces()->iterate(ns : Namespace; result: String = self.name | ns.name->union(self.separator())->union(result)) else Set{} endif

\+ **separator** () : String [1..1] {query}

The query separator() gives the string that is used to separate names when constructing a qualified name.

body (OCL): result = '::'

Constraints

has_no_qualified_name

If there is no name, or one of the containing namespaces has no name, there is no qualified name.

expression (OCL): (self.name->isEmpty() or self.allNamespaces()->select(ns | ns.name->isEmpty())->notEmpty()) implies self.qualifiedName->isEmpty()

has_qualified_name

When there is a name, and all of the containing namespaces have a name, the qualified name is constructed from the names of the containing namespaces.

expression (OCL): (self.name->notEmpty() and self.allNamespaces()->select(ns | ns.name->isEmpty())->isEmpty()) implies self.qualifiedName = self.allNamespaces()->iterate(ns : Namespace; result: String = self.name | ns.name->union(self.separator())->union(result))

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Class Namespace

A namespace is an element in a model that contains a set of named elements that can be identified by name.

Generalizations:

NamedElement

Specializations:

BehavioralFeature, Classifier, Classifier

Owned Association Ends

+ /**member** : NamedElement [0..*] {readOnly, union}

A collection of NamedElements identifiable within the Namespace, either by being owned or by being introduced by importing or inheritance.

+ /**ownedMember** : NamedElement [0..*] {readOnly, union, subsets ownedElement, subsets member}

A collection of NamedElements owned by the Namespace.

Operations

+ **getNamesOfMember** (element : NamedElement) : String [0..*] {query}

The query getNamesOfMember() gives a set of all of the names that a member would have in a Namespace. In general a member can have multiple names in a Namespace if it is imported more than once with different aliases. Those semantics are specified by overriding the getNamesOfMember operation. The specification here simply returns a set containing a single name, or the empty set if no name.

body (OCL): result = if member->includes(element) then Set{}->including(element.name) else Set{} endif

+ **membersAreDistinguishable** () : Boolean [1..1] {query}

The Boolean query membersAreDistinguishable() determines whether all of the namespaces members are distinguishable within it.

body (OCL): result = self.member->forAll(memb | self.member->excluding(memb)->forAll(other | memb.isDistinguishableFrom(other, self)))

Constraints

members_distinguishable

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Class Namespace

All the members of a Namespace are distinguishable within it.

expression (OCL): membersAreDistinguishable()

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Association A_member_namespace

Member Ends:

member, namespace

Owned Association Ends

+ **namespace** : Namespace [0..*]

Package InfrastructureLibrary::Core::Abstractions::Namespaces

Association A_ownedMember_namespace

Member Ends:

ownedMember, namespace

Package InfrastructureLibrary::Core::Abstractions::Ownerships

Nesting Package:

Abstractions

Imported Packages:

Elements

Class Summary
Element

Association Summary
A_ownedComment_owningElement
A_ownedElement_owner

Package InfrastructureLibrary::Core::Abstractions::Ownerships

Class Element

An element is a constituent of a model. As such, it has the capability of owning other elements.

Specializations:

Comment, MultiplicityElement, NamedElement, NamedElement, Relationship, Slot, ValueSpecification

Found in Diagrams:

Abstractions Expressions

Owned Association Ends

+ **ownedComment** : Comment [0..*] {subsets ownedElement}

The Comments owned by this element.

+ /**ownedElement** : Element [0..*] {readOnly, union}

The Elements owned by this element.

+ /**owner** : Element [0..1] {readOnly, union}

The Element that owns this element.

Operations

+ **allOwnedElements** () : Element [0..*] {query}

The query allOwnedElements() gives all of the direct and indirect owned elements of an element.

body (OCL): result = ownedElement->union(ownedElement->collect(e | e.allOwnedElements()))

+ **mustBeOwned** () : Boolean [1..1] {query}

The query mustBeOwned() indicates whether elements of this type must have an owner. Subclasses of Element that do not require an owner must override this operation.

body (OCL): result = true

Constraints

has_owner

Elements that must be owned must have an owner.

expression (OCL): self.mustBeOwned() implies owner->notEmpty()

not_own_self

Package InfrastructureLibrary::Core::Abstractions::Ownerships

Class Element

An element may not directly or indirectly own itself.

expression (OCL): not self.allOwnedElements()->includes(self)

Package InfrastructureLibrary::Core::Abstractions::Ownerships

Association A_ownedComment_owningElement

Member Ends:

ownedComment, owningElement

Owned Association Ends

+ **owningElement** : Element [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Abstractions::Ownerships

Association A_ownedElement_owner

Member Ends:

ownedElement, owner

Package InfrastructureLibrary::Core::Abstractions::Redefinitions

Nesting Package:

Abstractions

Imported Packages:

Super

Class Summary
RedefinableElement

Association Summary
A_redefinedElement_redefinableElement
A_redefinitionContext_redefinableElement

Package InfrastructureLibrary::Core::Abstractions::Redefinitions

Class RedefinableElement

A redefinable element is an element that, when defined in the context of a classifier, can be redefined more specifically or differently in the context of another classifier that specializes (directly or indirectly) the context classifier.

Generalizations:

NamedElement

Owned Association Ends

+ /**redefinedElement** : RedefinableElement [0..*] {readOnly, union}

The redefinable element that is being redefined by this element.

+ /**redefinitionContext** : Classifier [0..*] {readOnly, union}

References the contexts that this element may be redefined from.

Operations

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two RedefinableElements in a context in which redefinition is possible, whether redefinition would be logically consistent. By default, this is false; this operation must be overridden for subclasses of RedefinableElement to define the consistency conditions.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = false

+ **isRedefinitionContextValid** (redefined : RedefinableElement) : Boolean [1..1] {query}

The query isRedefinitionContextValid() specifies whether the redefinition contexts of this RedefinableElement are properly related to the redefinition contexts of the specified RedefinableElement to allow this element to redefine the other. By default at least one of the redefinition contexts of this element must be a specialization of at least one of the redefinition contexts of the specified element.

body (OCL): result = redefinitionContext->exists(c | c.allParents()->includes (redefined. redefinitionContext)))

Constraints

redefinition_consistent

A redefining element must be consistent with each redefined element.

Package InfrastructureLibrary::Core::Abstractions::Redefinitions

Class RedefinableElement

expression (OCL): self.redefinedElement->forAll(re | re.isConsistentWith(self))

redefinition_context_valid

At least one of the redefinition contexts of the redefining element must be a specialization of at least one of the redefinition contexts for each redefined element.

expression (OCL): self.redefinedElement->forAll(e | self.isRedefinitionContextValid(e))

Package InfrastructureLibrary::Core::Abstractions::Redefinitions

Association A_redefinedElement_redefinableElement

Member Ends:

redefinedElement, redefinableElement

Owned Association Ends

+ **redefinableElement** : RedefinableElement [0..*]

Package InfrastructureLibrary::Core::Abstractions::Redefinitions

Association A_redefinitionContext_redefinableElement

Member Ends:

redefinitionContext, redefinableElement

Owned Association Ends

+ **redefinableElement** : RedefinableElement [0..*]

Package InfrastructureLibrary::Core::Abstractions::Relationships

Nesting Package:

Abstractions

Imported Packages:

Ownerships

Class Summary
DirectedRelationship
Relationship

Association Summary
A_relatedElement_relationship
A_source_directedRelationship
A_target_directedRelationship

Package InfrastructureLibrary::Core::Abstractions::Relationships

Class DirectedRelationship

A directed relationship represents a relationship between a collection of source model elements and a collection of target model elements.

Generalizations:

Relationship

Specializations:

Generalization

Owned Association Ends

+ /**source** : Element [1..*] {readOnly, union, subsets relatedElement}

Specifies the sources of the DirectedRelationship.

+ /**target** : Element [1..*] {readOnly, union, subsets relatedElement}

Specifies the targets of the DirectedRelationship.

Package InfrastructureLibrary::Core::Abstractions::Relationships

Class Relationship

Relationship is an abstract concept that specifies some kind of relationship between elements.

Generalizations:

Element

Specializations:

DirectedRelationship

Owned Association Ends

+ /**relatedElement** : Element [1..*] {readOnly, union}

Specifies the elements related by the Relationship.

Package InfrastructureLibrary::Core::Abstractions::Relationships

Association A_relatedElement_relationship

Member Ends:

relatedElement, relationship

Owned Association Ends

+ **relationship** : Relationship [0..*]

Package InfrastructureLibrary::Core::Abstractions::Relationships

Association A_source_directedRelationship

Member Ends:

source, directedRelationship

Owned Association Ends

+ **directedRelationship** : DirectedRelationship [0..*]

Package InfrastructureLibrary::Core::Abstractions::Relationships

Association A_target_directedRelationship

Member Ends:

target, directedRelationship

Owned Association Ends

+ **directedRelationship** : DirectedRelationship [0..*]

Package InfrastructureLibrary::Core::Abstractions::StructuralFeatures

Nesting Package:

Abstractions

Imported Packages:

Classifiers, TypedElements

Class Summary
StructuralFeature

Package InfrastructureLibrary::Core::Abstractions::StructuralFeatures

Class StructuralFeature

A structural feature is a typed feature of a classifier that specifies the structure of instances of the classifier.

Generalizations:

Feature, TypedElement

Package InfrastructureLibrary::Core::Abstractions::Super

Nesting Package:

Abstractions

Imported Packages:

Classifiers

Class Summary
Classifier

Association Summary
A_general_classifier
A_inheritedMember_classifier

Package InfrastructureLibrary::Core::Abstractions::Super

Class Classifier

A classifier can specify a generalization hierarchy by referencing its general classifiers.

Generalizations:

Namespace

Attributes

+ **isAbstract** : Boolean [1..1] = false

If true, the Classifier does not provide a complete declaration and can typically not be instantiated. An abstract classifier is intended to be used by other classifiers e.g. as the target of general metarelationships or generalization relationships.

Owned Association Ends

+ **general** : Classifier [0..*]

Specifies the more general classifiers in the generalization hierarchy for this Classifier.

+ **/inheritedMember** : NamedElement [0..*] {readOnly, subsets member}

Specifies all elements inherited by this classifier from the general classifiers.

Operations

+ **allParents** () : Classifier [0..*] {query}

The query allParents() gives all of the direct and indirect ancestors of a generalized Classifier.

body (OCL): result = self.parents()->union(self.parents()->collect(p | p.allParents())

+ **hasVisibilityOf** (n : NamedElement) : Boolean [1..1] {query}

The query hasVisibilityOf() determines whether a named element is visible in the classifier. By default all are visible. It is only called when the argument is something owned by a parent.

precondition (): self.allParents()->collect(c | c.member)->includes(n)

body (OCL): result = if (self.inheritedMember->includes (n)) then (n.visibility <> #private) else true

+ **inherit** (inhs : NamedElement [0..*]) : NamedElement [0..*] {query}

The query inherit() defines how to inherit a set of elements. Here the operation is defined to inherit them all. It is intended to be redefined in circumstances where inheritance is affected by redefinition.

body (OCL): result = inhs

Package InfrastructureLibrary::Core::Abstractions::Super

Class Classifier

+ **inheritableMembers** (c : Classifier) : NamedElement [0..*] {query}

The query inheritableMembers() gives all of the members of a classifier that may be inherited in one of its descendants, subject to whatever visibility restrictions apply.

precondition (): c.allParents()->includes(self)

body (OCL): result = member->select(m | c.hasVisibilityOf(m))

+ **inheritedMember** () : NamedElement [0..*] {query}

The inheritedMember association is derived by inheriting the inheritable members of the parents.

body (OCL): result = self.inherit(self.parents()->collect(p | p.inheritableMembers(self))

+ **maySpecializeType** (c : Classifier) : Boolean [1..1] {query}

The query maySpecializeType() determines whether this classifier may have a generalization relationship to classifiers of the specified type. By default a classifier may specialize classifiers of the same or a more general type. It is intended to be redefined by classifiers that have different specialization constraints.

body (OCL): result = self.oclIsKindOf(c.oclType)

+ **parents** () : Classifier [0..*] {query}

The query parents() gives all of the immediate ancestors of a generalized Classifier.

body (OCL): result = general

Constraints

no_cycles_in_generalization

Generalization hierarchies must be directed and acyclical. A classifier can not be both a transitively general and transitively specific classifier of the same classifier.

expression (OCL): not self.allParents()->includes(self)

specialize_type

A classifier may only specialize classifiers of a valid type.

expression (OCL): self.parents()->forAll(c | self.maySpecializeType(c))

Package InfrastructureLibrary::Core::Abstractions::Super

Association A_general_classifier

Member Ends:

general, classifier

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package InfrastructureLibrary::Core::Abstractions::Super

Association A_inheritedMember_classifier

Member Ends:

inheritedMember, classifier

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package InfrastructureLibrary::Core::Abstractions::TypedElements

Nesting Package:

Abstractions

Imported Packages:

Namespaces

Class Summary
Type
TypedElement

Association Summary
A_type_typedElement

Package InfrastructureLibrary::Core::Abstractions::TypedElements

Class Type

A type constrains the values represented by a typed element.

Generalizations:

NamedElement

Specializations:

Classifier

Operations

+ **conformsTo** (other : Type) : Boolean [1..1] {query}

The query conformsTo() gives true for a type that conforms to another. By default, two types do not conform to each other. This query is intended to be redefined for specific conformance situations.

body (OCL): result = false

Package InfrastructureLibrary::Core::Abstractions::TypedElements

Class TypedElement

A typed element has a type.

Generalizations:

NamedElement

Specializations:

Parameter, StructuralFeature

Owned Association Ends

+ **type** : Type [0..1]

The type of the TypedElement.

Package InfrastructureLibrary::Core::Abstractions::TypedElements

Association A_type_typedElement

Member Ends:

type, typedElement

Owned Association Ends

+ **typedElement** : TypedElement [0..*]

Package InfrastructureLibrary::Core::Abstractions::Visibilities

Nesting Package:

Abstractions

Imported Packages:

Namespaces

Class Summary
NamedElement

Enumeration Summary
VisibilityKind

Package InfrastructureLibrary::Core::Abstractions::Visibilities

Class NamedElement

NamedElement has a visibility attribute.

Attributes

+ **visibility** : VisibilityKind [0..1]

Determines where the NamedElement appears within different Namespaces within the overall model, and its accessibility.

Constraints

visibility_needs_ownership

If a NamedElement is not owned by a Namespace, it does not have a visibility.

expression (OCL): namespace->isEmpty() implies visibility->isEmpty()

Package InfrastructureLibrary::Core::Abstractions::Visibilities

Enumeration VisibilityKind

VisibilityKind is an enumeration type that defines literals to determine the visibility of elements in a model.

Enumeration Literals

package

A package element is owned by a namespace that is not a package, and is visible to elements that are in the same package as its owning namespace. Only named elements that are not owned by packages can be marked as having package visibility. Any element marked as having package visibility is visible to all elements within the nearest enclosing package (given that other owning elements have proper visibility). Outside the nearest enclosing package, an element marked as having package visibility is not visible.

private

A private element is only visible inside the namespace that owns it.

protected

A protected element is visible to elements that have a generalization relationship to the namespace that owns it.

public

A public element is visible to all elements that can access the contents of the namespace that owns it.

Operations

+ **bestVisibility** (vis : VisibilityKind [0..*]) : VisibilityKind [1..1] {query}

The query bestVisibility() examines a set of VisibilityKinds, and returns public as the preferred visibility.

body (OCL): result = if vis->includes(#public) then #public else #private endif

Package InfrastructureLibrary::Core::Basic

Nesting Package:

Core

Imported Packages:

PrimitiveTypes

Class Summary
Class
Comment
DataType
Element
Enumeration
EnumerationLiteral
MultiplicityElement
NamedElement
Operation
Package
Parameter
PrimitiveType
Property
Type
TypedElement

Association Summary
A_annotatedElement_comment
A_nestedPackage_nestingPackage
A_opposite_property
A_ownedAttribute_class
A_ownedComment_owningElement
A_ownedLiteral_enumeration
A_ownedOperation_class
A_ownedParameter_operation
A_ownedType_package
A_raisedException_operation
A_superClass_class
A_type_typedElement

Package InfrastructureLibrary::Core::Basic

Class Class

A class is a type that has objects as its instances.

Generalizations:

Type

Attributes

+ **isAbstract** : Boolean [1..1] = false

True when a class is abstract.

Owned Association Ends

+ **ownedAttribute** : Property [0..*] {ordered}

The attributes owned by a class. These do not include the inherited attributes. Attributes are represented by instances of Property.

+ **ownedOperation** : Operation [0..*] {ordered}

The operations owned by a class. These do not include the inherited operations.

+ **superClass** : Class [0..*]

The immediate superclasses of a class, from which the class inherits.

Package InfrastructureLibrary::Core::Basic

Class Comment

A comment is a textual annotation that can be attached to a set of elements.

Generalizations:

Element

Attributes

+ **body** : String [0..1]

Specifies a string that is the comment.

Owned Association Ends

+ **annotatedElement** : Element [0..*]

References the Element(s) being commented.

Package InfrastructureLibrary::Core::Basic

Class DataType

DataType is an abstract class that acts as a common superclass for different kinds of data types.

Generalizations:

Type

Specializations:

Enumeration, PrimitiveType

Package InfrastructureLibrary::Core::Basic

Class Element

An element is a constituent of a model.

Specializations:

Comment, MultiplicityElement, NamedElement

Owned Association Ends

+ **ownedComment** : Comment [0..*]

The Comments owned by this element.

Package InfrastructureLibrary::Core::Basic

Class Enumeration

An enumeration defines a set of literals that can be used as its values.

Generalizations:

DataType

Owned Association Ends

+ **ownedLiteral** : EnumerationLiteral [0..*] {ordered}

The ordered set of literals for this Enumeration.

Package InfrastructureLibrary::Core::Basic

Class EnumerationLiteral

An enumeration literal is a value of an enumeration.

Generalizations:

NamedElement

Owned Association Ends

+ **enumeration** : Enumeration [0..1]

The Enumeration that this EnumerationLiteral is a member of.

Package InfrastructureLibrary::Core::Basic

Class MultiplicityElement

A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. A multiplicity element embeds this information to specify the allowable cardinalities for an instantiation of this element.

Generalizations:

Element

Specializations:

Operation, Parameter, Property

Attributes

+ **isOrdered** : Boolean [1..1] = false

For a multivalued multiplicity, this attribute specifies whether the values in an instantiation of this element are sequentially ordered.

+ **isUnique** : Boolean [1..1] = true

For a multivalued multiplicity, this attributes specifies whether the values in an instantiation of this element are unique.

+ **lower** : Integer [0..1] = 1

Specifies the lower bound of the multiplicity interval.

+ **upper** : UnlimitedNatural [0..1] = 1

Specifies the upper bound of the multiplicity interval.

Operations

+ **includesCardinality** (C : Integer) : Boolean [1..1] {query}

The query includesCardinality() checks whether the specified cardinality is valid for this multiplicity.

precondition (): upperBound()->notEmpty() and lowerBound()->notEmpty()

body (OCL): result = (lowerBound() <= C) and (upperBound() >= C)

+ **includesMultiplicity** (M : MultiplicityElement) : Boolean [1..1] {query}

The query includesMultiplicity() checks whether this multiplicity includes all the cardinalities allowed by the specified multiplicity.

precondition (): self.upperBound()->notEmpty() and self.lowerBound()->notEmpty() and M.upperBound()->notEmpty() and M.lowerBound()->notEmpty()

Package InfrastructureLibrary::Core::Basic

Class MultiplicityElement

body (OCL): result = (self.lowerBound() <= M.lowerBound()) and (self.upperBound() >= M.upperBound())

+ **isMultivalued** () : Boolean [1..1] {query}

The query isMultivalued() checks whether this multiplicity has an upper bound greater than one.

precondition (): upperBound()->notEmpty()

body (OCL): result = upperBound() > 1

+ **lowerBound** () : Integer [1..1] {query}

The query lowerBound() returns the lower bound of the multiplicity as an integer.

body (OCL): result = if lower->notEmpty() then lower else 1 endif

+ **upperBound** () : UnlimitedNatural [1..1] {query}

The query upperBound() returns the upper bound of the multiplicity for a bounded multiplicity as an unlimited natural.

body (OCL): result = if upper->notEmpty() then upper else 1 endif

Constraints

lower_ge_0

The lower bound must be a non-negative integer literal.

expression (OCL): lowerBound()->notEmpty() implies lowerBound() >= 0

upper_ge_lower

The upper bound must be greater than or equal to the lower bound.

expression (OCL): (upperBound()->notEmpty() and lowerBound()->notEmpty()) implies upperBound() >= lowerBound()

Package InfrastructureLibrary::Core::Basic

Class NamedElement

A named element represents an element with a name.

Generalizations:

Element

Specializations:

EnumerationLiteral, Package, Type, TypedElement

Attributes

+ **name** : String [0..1]

The name of the NamedElement.

Package InfrastructureLibrary::Core::Basic

Class Operation

An operation is owned by a class and may be invoked in the context of objects that are instances of that class. It is a typed element and a multiplicity element.

Generalizations:

MultiplicityElement, TypedElement

Owned Association Ends

+ **class** : Class [0..1]

The class that owns the operation.

+ **ownedParameter** : Parameter [0..*] {ordered}

The parameters to the operation.

+ **raisedException** : Type [0..*]

The exceptions that are declared as possible during an invocation of the operation.

Package InfrastructureLibrary::Core::Basic

Class Package

A package is a container for types and other packages.

Generalizations:

NamedElement

Owned Association Ends

+ **nestedPackage** : Package [0..*]

The set of contained packages.

+ **nestingPackage** : Package [0..1]

The containing package.

+ **ownedType** : Type [0..*]

The set of contained types.

Package InfrastructureLibrary::Core::Basic

Class Parameter

A parameter is a typed element that represents a parameter of an operation.

Generalizations:

MultiplicityElement, TypedElement

Owned Association Ends

+ **operation** : Operation [0..1]

The operation that owns the parameter.

Package InfrastructureLibrary::Core::Basic

Class PrimitiveType

A primitive type is a data type implemented by the underlying infrastructure and made available for modeling.

Generalizations:

DataType

Package InfrastructureLibrary::Core::Basic

Class Property

A property is a typed element that represents an attribute of a class.

Generalizations:

MultiplicityElement, TypedElement

Attributes

+ **default** : String [0..1]

A string that is evaluated to give a default value for the attribute when an object of the owning class is instantiated.

+ **isComposite** : Boolean [1..1] = false

If isComposite is true, the object containing the attribute is a container for the object or value contained in the attribute.

+ **isDerived** : Boolean [1..1] = false

If isDerived is true, the value of the attribute is derived from information elsewhere.

+ **isReadOnly** : Boolean [1..1] = false

If isReadOnly is true, the attribute may not be written to after initialization.

Owned Association Ends

+ **class** : Class [0..1]

The class that owns the property, and of which the property is an attribute.

+ **opposite** : Property [0..1]

Two attributes attr1 and attr2 of two objects o1 and o2 (which may be the same object) may be paired with each other so that o1.attr1 refers to o2 if and only if o2.attr2 refers to o1. In such a case attr1 is the opposite of attr2 and attr2 is the opposite of attr1.

Package InfrastructureLibrary::Core::Basic

Class Type

A type is a named element that is used as the type for a typed element. A type can be contained in a package.

Generalizations:

NamedElement

Specializations:

Class, DataType

Owned Association Ends

+ **package** : Package [0..1]

Specifies the owning package of this classifier, if any.

Package InfrastructureLibrary::Core::Basic

Class TypedElement

A typed element is a kind of named element that represents an element with a type.

Generalizations:

NamedElement

Specializations:

Operation, Parameter, Property

Owned Association Ends

+ **type** : Type [0..1]

The type of the TypedElement.

Package InfrastructureLibrary::Core::Basic

Association A_annotatedElement_comment

Member Ends:

annotatedElement, comment

Owned Association Ends

+ **comment** : Comment [0..*]

Package InfrastructureLibrary::Core::Basic

Association A_nestedPackage_nestingPackage

Member Ends:

nestedPackage, nestingPackage

Package InfrastructureLibrary::Core::Basic

Association A_opposite_property

Member Ends:

opposite, property

Owned Association Ends

+ **property** : Property [0..1]

Package InfrastructureLibrary::Core::Basic

Association A_ownedAttribute_class

Member Ends:

ownedAttribute, class

Package InfrastructureLibrary::Core::Basic

Association A_ownedComment_owningElement

Member Ends:

ownedComment, owningElement

Owned Association Ends

+ **owningElement** : Element [0..1]

Package InfrastructureLibrary::Core::Basic

Association A_ownedLiteral_enumeration

Member Ends:

ownedLiteral, enumeration

Package InfrastructureLibrary::Core::Basic

Association A_ownedOperation_class

Member Ends:

ownedOperation, class

Package InfrastructureLibrary::Core::Basic

Association A_ownedParameter_operation

Member Ends:

ownedParameter, operation

Package InfrastructureLibrary::Core::Basic

Association A_ownedType_package

Member Ends:

ownedType, package

Package InfrastructureLibrary::Core::Basic

Association A_raisedException_operation

Member Ends:

raisedException, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package InfrastructureLibrary::Core::Basic

Association A_superClass_class

Member Ends:

superClass, class

Owned Association Ends

+ **class** : Class [0..*]

Package InfrastructureLibrary::Core::Basic

Association A_type_typedElement

Member Ends:

type, typedElement

Owned Association Ends

+ **typedElement** : TypedElement [0..*]

Package InfrastructureLibrary::Core::Constructs

Nesting Package:

Core

Imported Packages:

PrimitiveTypes

Diagram Summary
Classifiers
Expressions

Class Summary
Association
BehavioralFeature
Class
Classifier
Comment
Constraint
DataType
DirectedRelationship
Element
ElementImport
Enumeration
EnumerationLiteral
Expression
Feature
MultiplicityElement
NamedElement
Namespace
OpaqueExpression
Operation
Package
PackageImport
PackageMerge
PackageableElement
Parameter
PrimitiveType
Property

Package InfrastructureLibrary::Core::Constructs

RedefinableElement
Relationship
StructuralFeature
Type
TypedElement
ValueSpecification

Enumeration Summary
ParameterDirectionKind
VisibilityKind

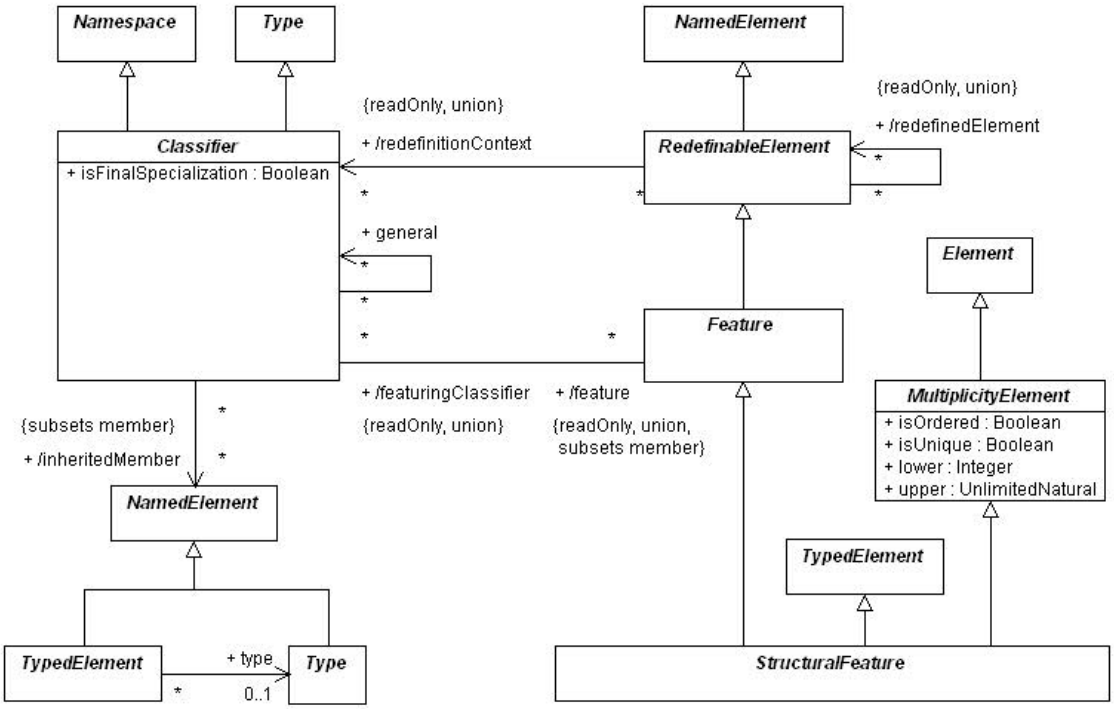
Association Summary
A_annotatedElement_comment
A_attribute_classifier
A_bodyCondition_bodyContext
A_constrainedElement_constraint
A_elementImport_importingNamespace
A_endType_association
A_feature_featuringClassifier
A_general_classifier
A_importedElement_elementImport
A_importedMember_namespace
A_importedPackage_packageImport
A_inheritedMember_classifier
A_memberEnd_association
A_member_namespace
A_mergedPackage_packageMerge
A_navigableOwnedEnd_association
A_nestedPackage_nestingPackage
A_operand_expression
A_opposite_property
A_ownedAttribute_class
A_ownedAttribute_datatype
A_ownedComment_owningElement
A_ownedElement_owner
A_ownedEnd_owningAssociation
A_ownedLiteral_enumeration

Package InfrastructureLibrary::Core::Constructs

A_ownedMember_namespace
A_ownedOperation_class
A_ownedOperation_datatype
A_ownedParameter_operation
A_ownedParameter_ownerFormalParam
A_ownedRule_context
A_ownedType_package
A_packageImport_importingNamespace
A_packageMerge_receivingPackage
A_packagedElement_owningPackage
A_postcondition_postContext
A_precondition_preContext
A_raisedException_behavioralFeature
A_raisedException_operation
A_redefinedElement_redefinableElement
A_redefinedOperation_operation
A_redefinedProperty_property
A_redefinitionContext_redefinableElement
A_relatedElement_relationship
A_source_directedRelationship
A_specification_owningConstraint
A_subsettedProperty_property
A_superClass_class
A_target_directedRelationship
A_type_operation
A_type_typedElement

Package InfrastructureLibrary::Core::Constructs

Diagram Classifiers

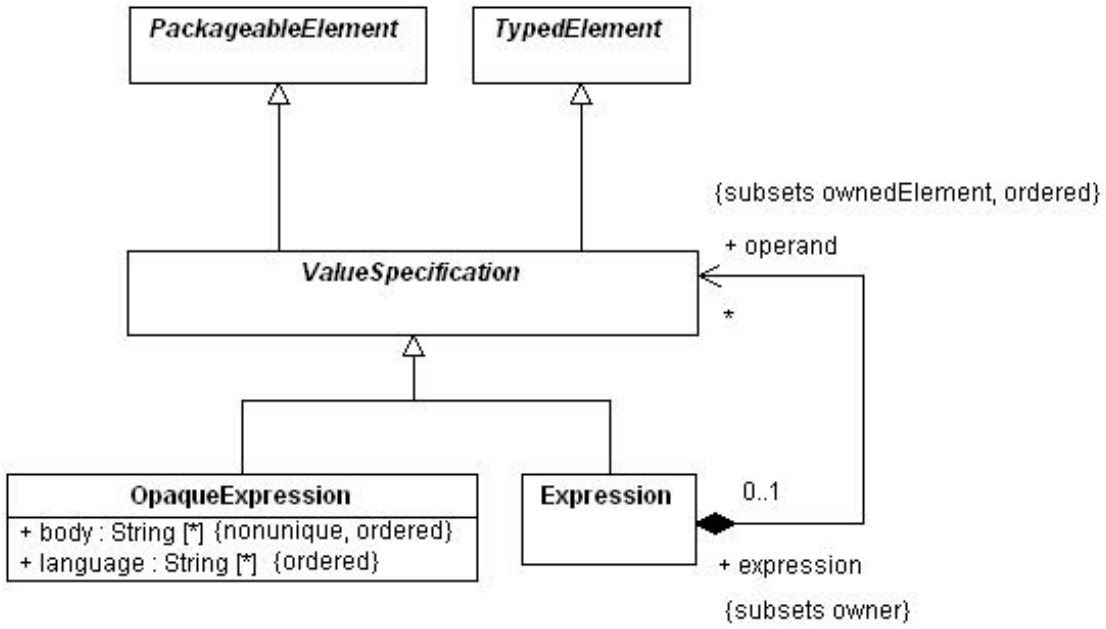


Classifiers Local to Package:

Classifier, Element, Feature, MultiplicityElement, NamedElement, Namespace, RedefinableElement , StructuralFeature, Type, TypedElement

Package InfrastructureLibrary::Core::Constructs

Diagram Expressions



Classifiers Local to Package:

Expression, OpaqueExpression, PackageableElement, TypedElement, ValueSpecification

Package InfrastructureLibrary::Core::Constructs

Class Association

An association describes a set of tuples whose values refer to typed instances. An instance of an association is called a link.A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.

Generalizations:

Classifier, Relationship

Specializations:

Extension

Found in Diagrams:

Profile Elements

Attributes

+ **isDerived** : Boolean [1..1] = false

Specifies whether the association is derived from other model elements such as other associations or constraints.

Owned Association Ends

+ /**endType** : Type [1..*] {readOnly, subsets relatedElement}

References the classifiers that are used as types of the ends of the association.

+ **memberEnd** : Property [2..*] {ordered, subsets member}

Each end represents participation of instances of the classifier connected to the end in links of the association.

+ **navigableOwnedEnd** : Property [0..*] {subsets ownedEnd}

The navigable ends that are owned by the association itself.

+ **ownedEnd** : Property [0..*] {ordered, subsets memberEnd, subsets feature, subsets ownedMember }

The ends that are owned by the association itself.

Operations

+ **endType** () : Type [0..*] {ordered, query}

Package InfrastructureLibrary::Core::Constructs

Class Association

endType is derived from the types of the member ends.

body (OCL): result = self.memberEnd->collect(e | e.type)

Constraints

association_ends

Association ends of associations with more than two ends must be owned by the association.

expression (OCL): if memberEnd->size() > 2 then ownedEnd->includesAll(memberEnd)

binary_associations

Only binary associations can be aggregations.

expression (OCL): self.memberEnd->exists(isComposite) implies self.memberEnd->size() = 2

specialized_end_number

An association specializing another association has the same number of ends as the other association.

expression (OCL): parents()->select(oclIsKindOf(Association)).oclAsType(Association)->forAll(p | p.memberEnd->size() = self.memberEnd->size())

specialized_end_types

When an association specializes another association, every end of the specific association corresponds to an end of the general association, and the specific end reaches the same type or a subtype of the more general end.

expression (OCL): Sequence{1..self.memberEnd->size()}-> forAll(i | self.general->select (oclIsKindOf(Association)).oclAsType(Association)-> forAll(ga |self.memberEnd->at(i).type. conformsTo(ga.memberEnd->at(i).type)))

Package InfrastructureLibrary::Core::Constructs

Class BehavioralFeature

A behavioral feature is a feature of a classifier that specifies an aspect of the behavior of its instances.

Generalizations:

Feature, Namespace

Specializations:

Operation

Owned Association Ends

+ **ownedParameter** : Parameter [0..*] {ordered, subsets ownedMember}

Specifies the ordered set of formal parameters of this BehavioralFeature.

+ **raisedException** : Type [0..*]

References the Types representing exceptions that may be raised during an invocation of this feature.

Operations

+ **isDistinguishableFrom** (n : NamedElement, ns : Namespace) : Boolean [1..1] {query}

The query isDistinguishableFrom() determines whether two BehavioralFeatures may coexist in the same Namespace. It specifies that they have to have different signatures.

body (OCL): result = if n.oclIsKindOf(BehavioralFeature) then if ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->notEmpty() then Set{}->include(self)->include(n)->isUnique(bf | bf.parameter->collect(type)) else true endif else true endif

Package InfrastructureLibrary::Core::Constructs

Class Class

A class describes a set of objects that share the same specifications of features, constraints, and semantics.

Generalizations:

Classifier

Attributes

+ **isAbstract** : Boolean [1..1] = false

True when a class is abstract.

Owned Association Ends

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The attributes (i.e. the properties) owned by the class.

+ **ownedOperation** : Operation [0..*] {ordered, subsets feature, subsets ownedMember}

The operations owned by the class.

+ **superClass** : Class [0..*] {redefines general}

This gives the superclasses of a class.

Operations

+ **inherit** (inhs : NamedElement [0..*]) : NamedElement [0..*] {query}

The inherit operation is overridden to exclude redefined properties.

body (OCL): result = inhs->excluding(inh | ownedMember->select(oclIsKindOf (RedefinableElement))->select(redefinedElement->includes(inh)))

Package InfrastructureLibrary::Core::Constructs

Class Classifier

A classifier is a classification of instances - it describes a set of instances that have features in common. A classifier can specify a generalization hierarchy by referencing its general classifiers.

Generalizations:

Namespace, Type

Specializations:

Association, Class, Class, DataType

Found in Diagrams:

Classifiers, Profile Elements

Attributes

+ **isFinalSpecialization** : Boolean [1..1] = false

If true, the Classifier cannot be specialized by generalization. Note that this property is preserved through package merge operations; that is, the capability to specialize a Classifier (i.e., isFinalSpecialization =false) must be preserved in the resulting Classifier of a package merge operation where a Classifier with isFinalSpecialization =false is merged with a matching Classifier with isFinalSpecialization =true: the resulting Classifier will have isFinalSpecialization =false.

Owned Association Ends

+ /**attribute** : Property [0..*] {readOnly, union, subsets feature}

Refers to all of the Properties that are direct (i.e. not inherited or imported) attributes of the classifier.

+ /**feature** : Feature [0..*] {readOnly, union, subsets member}

Note that there may be members of the Classifier that are of the type Feature but are not included in this association, e.g. inherited features.

+ **general** : Classifier [0..*]

References the general classifier in the Generalization relationship.

+ /**inheritedMember** : NamedElement [0..*] {readOnly, subsets member}

Specifies all elements inherited by this classifier from the general classifiers.

Operations

Package InfrastructureLibrary::Core::Constructs

Class Classifier

+ **allFeatures** () : Feature [0..*] {query}

  The query allFeatures() gives all of the features in the namespace of the classifier. In general, through mechanisms such as inheritance, this will be a larger set than feature.

  body (OCL): result = member->select(oclIsKindOf(Feature))

+ **allParents** () : Classifier [0..*] {query}

  The query allParents() gives all of the direct and indirect ancestors of a generalized Classifier.

  body (OCL): result = self.parents()->union(self.parents()->collect(p | p.allParents())

+ **conformsTo** (other : Classifier) : Boolean [1..1] {query}

  The query conformsTo() gives true for a classifier that defines a type that conforms to another. This is used, for example, in the specification of signature conformance for operations.

  body (OCL): result = (self=other) or (self.allParents()->includes(other))

+ **general** () : Classifier [0..*] {query}

  The general classifiers are the classifiers referenced by the generalization relationships.

  body (OCL): result = self.parents()

+ **hasVisibilityOf** (n : NamedElement) : Boolean [1..1] {query}

  The query hasVisibilityOf() determines whether a named element is visible in the classifier. By default all are visible. It is only called when the argument is something owned by a parent.

  precondition (): self.allParents()->collect(c | c.member)->includes(n)

  body (OCL): result = if (self.inheritedMember->includes(n)) then (n.visibility <> #private) else true

+ **inherit** (inhs : NamedElement [0..*]) : NamedElement [0..*] {query}

  The inherit operation is overridden to exclude redefined properties.

  body (OCL): result = inhs

+ **inheritableMembers** (c : Classifier) : NamedElement [0..*] {query}

  The query inheritableMembers() gives all of the members of a classifier that may be inherited in one of its descendants, subject to whatever visibility restrictions apply.

  precondition (): c.allParents()->includes(self)

  body (OCL): result = member->select(m | c.hasVisibilityOf(m))

+ **inheritedMember** () : NamedElement [0..*] {query}

  The inheritedMember association is derived by inheriting the inheritable members of the parents.

  body (OCL): result = self.inherit(self.parents()->collect(p | p.inheritableMembers(self))

Package InfrastructureLibrary::Core::Constructs

Class Classifier

+ **maySpecializeType** (c : Classifier) : Boolean [1..1] {query}

The query maySpecializeType() determines whether this classifier may have a generalization relationship to classifiers of the specified type. By default a classifier may specialize classifiers of the same or a more general type. It is intended to be redefined by classifiers that have different specialization constraints.

body (OCL): result = self.oclIsKindOf(c.oclType)

+ **parents** () : Classifier [0..*] {query}

The query parents() gives all of the immediate ancestors of a generalized Classifier.

body (OCL): result = generalization.general

Constraints

no_cycles_in_generalization

Generalization hierarchies must be directed and acyclical. A classifier can not be both a transitively general and transitively specific classifier of the same classifier.

expression (OCL): not self.allParents()->includes(self)

specialize_type

A classifier may only specialize classifiers of a valid type.

expression (OCL): self.parents()->forAll(c | self.maySpecializeType(c))

Package InfrastructureLibrary::Core::Constructs

Class Comment

A comment is a textual annotation that can be attached to a set of elements.

Generalizations:

Element

Attributes

+ **body** : String [0..1]

Specifies a string that is the comment.

Owned Association Ends

+ **annotatedElement** : Element [0..*]

References the Element(s) being commented.

Package InfrastructureLibrary::Core::Constructs

Class Constraint

A constraint is a condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

Generalizations:

PackageableElement

Owned Association Ends

+ **constrainedElement** : Element [0..*] {ordered}

The ordered set of Elements referenced by this Constraint.

+ **context** : Namespace [0..1] {subsets namespace}

+ **specification** : ValueSpecification [1..1] {subsets ownedElement}

A condition that must be true when evaluated in order for the constraint to be satisfied.

Constraints

not_apply_to_self

A constraint cannot be applied to itself.

expression (OCL): not constrainedElement->includes(self)

value_specification_boolean

The value specification for a constraint must evaluate to a Boolean value.

expression (OCL): self.specification().booleanValue().isOclKindOf(Boolean)

Package InfrastructureLibrary::Core::Constructs

Class DataType

A data type is a type whose instances are identified only by their value. A data type may contain attributes to support the modeling of structured data types.

Generalizations:

Classifier

Specializations:

Enumeration, PrimitiveType

Owned Association Ends

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The Attributes owned by the DataType.

+ **ownedOperation** : Operation [0..*] {ordered, subsets feature, subsets ownedMember}

The Operations owned by the DataType.

Operations

+ **inherit** (inhs : NamedElement [0..*]) : NamedElement [0..*] {query}

The inherit operation is overridden to exclude redefined properties.

body (OCL): result = inhs->excluding(inh | ownedMember->select(oclIsKindOf (RedefinableElement))->select(redefinedElement->includes(inh)))

Package InfrastructureLibrary::Core::Constructs

Class DirectedRelationship

A directed relationship represents a relationship between a collection of source model elements and a collection of target model elements.

Generalizations:

Relationship

Specializations:

ElementImport, PackageImport, PackageMerge, ProfileApplication

Found in Diagrams:

Profile Elements

Owned Association Ends

+ /**source** : Element [1..*] {readOnly, union, subsets relatedElement}

Specifies the sources of the DirectedRelationship.

+ /**target** : Element [1..*] {readOnly, union, subsets relatedElement}

Specifies the targets of the DirectedRelationship.

Package InfrastructureLibrary::Core::Constructs

Class Element

An element is a constituent of a model. As such, it has the capability of owning other elements.

Specializations:

Comment, Image, MultiplicityElement, NamedElement, Relationship

Found in Diagrams:

Classifiers, Profile Elements

Owned Association Ends

+ **ownedComment** : Comment [0..*] {subsets ownedElement}

The Comments owned by this element.

+ /**ownedElement** : Element [0..*] {readOnly, union}

The Elements owned by this element.

+ /**owner** : Element [0..1] {readOnly, union}

The Element that owns this element.

Operations

+ **allOwnedElements** () : Element [0..*] {query}

The query allOwnedElements() gives all of the direct and indirect owned elements of an element.

body (OCL): result = ownedElement->union(ownedElement->collect(e | e.allOwnedElements()))

+ **mustBeOwned** () : Boolean [1..1] {query}

The query mustBeOwned() indicates whether elements of this type must have an owner. Subclasses of Element that do not require an owner must override this operation.

body (OCL): result = true

Constraints

has_owner

Elements that must be owned must have an owner.

expression (OCL): self.mustBeOwned() implies owner->notEmpty()

not_own_self

An element may not directly or indirectly own itself.

Package InfrastructureLibrary::Core::Constructs

Class Element

expression (OCL): not self.allOwnedElements()->includes(self)

Package InfrastructureLibrary::Core::Constructs

Class ElementImport

An element import identifies an element in another package, and allows the element to be referenced using its name without a qualifier.

Generalizations:

DirectedRelationship

Found in Diagrams:

Profile Elements

Attributes

+ **alias** : String [0..1]

Specifies the name that should be added to the namespace of the importing package in lieu of the name of the imported packagable element. The aliased name must not clash with any other member name in the importing package. By default, no alias is used.

+ **visibility** : VisibilityKind [1..1] = public

Specifies the visibility of the imported PackageableElement within the importing Package. The default visibility is the same as that of the imported element. If the imported element does not have a visibility, it is possible to add visibility to the element import.

Owned Association Ends

+ **importedElement** : PackageableElement [1..1] {subsets target}

Specifies the PackageableElement whose name is to be added to a Namespace.

+ **importingNamespace** : Namespace [1..1] {subsets source, subsets owner}

Specifies the Namespace that imports a PackageableElement from another Package.

Operations

+ **getName** () : String [1..1] {query}

The query getName() returns the name under which the imported PackageableElement will be known in the importing namespace.

body (OCL): result = if self.alias->notEmpty() then self.alias else self.importedElement.name endif

Constraints

imported_element_is_public

Package InfrastructureLibrary::Core::Constructs

Class ElementImport

An importedElement has either public visibility or no visibility at all.

expression (OCL): self.importedElement.visibility.notEmpty() implies self.importedElement.visibility = #public

visibility_public_or_private

The visibility of an ElementImport is either public or private.

expression (OCL): self.visibility = #public or self.visibility = #private

Package InfrastructureLibrary::Core::Constructs

Class Enumeration

An enumeration is a data type whose values are enumerated in the model as enumeration literals.

Generalizations:

DataType

Owned Association Ends

+ **ownedLiteral** : EnumerationLiteral [0..*] {ordered, subsets ownedMember}

The ordered set of literals for this Enumeration.

Package InfrastructureLibrary::Core::Constructs

Class EnumerationLiteral

An enumeration literal is a user-defined data value for an enumeration.

Generalizations:

NamedElement

Owned Association Ends

+ **enumeration** : Enumeration [0..1] {subsets namespace}

The Enumeration that this EnumerationLiteral is a member of.

Package InfrastructureLibrary::Core::Constructs

Class Expression

An expression is a structured tree of symbols that denotes a (possibly empty) set of values when evaluated in a context.

Generalizations:

ValueSpecification

Found in Diagrams:

Expressions

Owned Association Ends

+ **operand** : ValueSpecification [0..*] {ordered, subsets ownedElement}

Specifies a sequence of operands.

Package InfrastructureLibrary::Core::Constructs

Class Feature

A feature declares a behavioral or structural characteristic of instances of classifiers.

Generalizations:

RedefinableElement

Specializations:

BehavioralFeature, StructuralFeature

Found in Diagrams:

Classifiers

Owned Association Ends

+ /**featuringClassifier** : Classifier [0..*] {readOnly, union}

The Classifiers that have this Feature as a feature.

Package InfrastructureLibrary::Core::Constructs

Class MultiplicityElement

A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. A multiplicity element embeds this information to specify the allowable cardinalities for an instantiation of this element.

Generalizations:

Element

Specializations:

Parameter, StructuralFeature

Found in Diagrams:

Classifiers

Attributes

+ **isOrdered** : Boolean [1..1] = false

For a multivalued multiplicity, this attribute specifies whether the values in an instantiation of this element are sequentially ordered.

+ **isUnique** : Boolean [1..1] = true

For a multivalued multiplicity, this attributes specifies whether the values in an instantiation of this element are unique.

+ **lower** : Integer [0..1] = 1

Specifies the lower bound of the multiplicity interval.

+ **upper** : UnlimitedNatural [0..1] = 1

Specifies the upper bound of the multiplicity interval.

Operations

+ **includesCardinality** (C : Integer) : Boolean [1..1] {query}

The query includesCardinality() checks whether the specified cardinality is valid for this multiplicity.

precondition (): upperBound()->notEmpty() and lowerBound()->notEmpty()

body (OCL): result = (lowerBound() <= C) and (upperBound() >= C)

+ **includesMultiplicity** (M : MultiplicityElement) : Boolean [1..1] {query}

The query includesMultiplicity() checks whether this multiplicity includes all the cardinalities

Package InfrastructureLibrary::Core::Constructs

Class MultiplicityElement

allowed by the specified multiplicity.

precondition (): self.upperBound()->notEmpty() and self.lowerBound()->notEmpty() and M.upperBound()->notEmpty() and M.lowerBound()->notEmpty()

body (OCL): result = (self.lowerBound() <= M.lowerBound()) and (self.upperBound() >= M.upperBound())

+ **isMultivalued** () : Boolean [1..1] {query}

The query isMultivalued() checks whether this multiplicity has an upper bound greater than one.

precondition (): upperBound()->notEmpty()

body (OCL): result = upperBound() > 1

+ **lowerBound** () : Integer [1..1] {query}

The query lowerBound() returns the lower bound of the multiplicity as an integer.

body (OCL): result = if lower->notEmpty() then lower else 1 endif

+ **upperBound** () : UnlimitedNatural [1..1] {query}

The query upperBound() returns the upper bound of the multiplicity for a bounded multiplicity as an unlimited natural.

body (OCL): result = if upper->notEmpty() then upper else 1 endif

Constraints

lower_ge_0

The lower bound must be a non-negative integer literal.

expression (OCL): lowerBound()->notEmpty() implies lowerBound() >= 0

upper_ge_lower

The upper bound must be greater than or equal to the lower bound.

expression (OCL): (upperBound()->notEmpty() and lowerBound()->notEmpty()) implies upperBound() >= lowerBound()

Package InfrastructureLibrary::Core::Constructs

Class NamedElement

A named element is an element in a model that may have a name.

Generalizations:

Element

Specializations:

EnumerationLiteral, Namespace, PackageableElement, RedefinableElement, Type, TypedElement

Found in Diagrams:

Classifiers

Attributes

+ **name** : String [0..1]

The name of the NamedElement.

+ /**qualifiedName** : String [0..1] {readOnly}

A name which allows the NamedElement to be identified within a hierarchy of nested Namespaces. It is constructed from the names of the containing namespaces starting at the root of the hierarchy and ending with the name of the NamedElement itself.

+ **visibility** : VisibilityKind [0..1]

Determines where the NamedElement appears within different Namespaces within the overall model, and its accessibility.

Owned Association Ends

+ /**namespace** : Namespace [0..1] {readOnly, union, subsets owner}

Specifies the namespace that owns the NamedElement.

Operations

+ **allNamespaces** () : Namespace [0..*] {ordered, query}

The query allNamespaces() gives the sequence of namespaces in which the NamedElement is nested, working outwards.

body (OCL): result = if self.namespace->isEmpty() then Sequence{} else self.namespace.allNamespaces()->prepend(self.namespace) endif

+ **isDistinguishableFrom** (n : NamedElement, ns : Namespace) : Boolean [1..1] {query}

Package InfrastructureLibrary::Core::Constructs

Class NamedElement

The query isDistinguishableFrom() determines whether two NamedElements may logically co-exist within a Namespace. By default, two named elements are distinguishable if (a) they have unrelated types or (b) they have related types but different names.

body (OCL): result = if self.oclIsKindOf(n.oclType) or n.oclIsKindOf(self.oclType) then ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->isEmpty() else true endif

+ **qualifiedName** () : String [1..1] {query}

When there is a name, and all of the containing namespaces have a name, the qualified name is constructed from the names of the containing namespaces.

body (OCL): result = if self.name->notEmpty() and self.allNamespaces()->select(ns | ns.name->isEmpty())->isEmpty() then self.allNamespaces()->iterate(ns : Namespace; result: String = self.name | ns.name->union(self.separator())->union(result)) else Set{} endif

+ **separator** () : String [1..1] {query}

The query separator() gives the string that is used to separate names when constructing a qualified name.

body (OCL): result = '::'

Constraints

has_no_qualified_name

If there is no name, or one of the containing namespaces has no name, there is no qualified name.

expression (OCL): (self.name->isEmpty() or self.allNamespaces()->select(ns | ns.name->isEmpty())->notEmpty()) implies self.qualifiedName->isEmpty()

has_qualified_name

When there is a name, and all of the containing namespaces have a name, the qualified name is constructed from the names of the containing namespaces.

expression (OCL): (self.name->notEmpty() and self.allNamespaces()->select(ns | ns.name->isEmpty())->isEmpty()) implies self.qualifiedName = self.allNamespaces()->iterate(ns : Namespace; result: String = self.name | ns.name->union(self.separator())->union(result))

visibility_needs_ownership

If a NamedElement is not owned by a Namespace, it does not have a visibility.

expression (OCL): namespace->isEmpty() implies visibility->isEmpty()

Package InfrastructureLibrary::Core::Constructs

Class Namespace

A namespace is an element in a model that contains a set of named elements that can be identified by name.

Generalizations:

NamedElement

Specializations:

BehavioralFeature, Classifier, Package, Package

Found in Diagrams:

Classifiers, Profile Elements

Owned Association Ends

+ **elementImport** : ElementImport [0..*] {subsets ownedElement}

References the ElementImports owned by the Namespace.

+ /**importedMember** : PackageableElement [0..*] {readOnly, subsets member}

References the PackageableElements that are members of this Namespace as a result of either PackageImports or ElementImports.

+ /**member** : NamedElement [0..*] {readOnly, union}

A collection of NamedElements identifiable within the Namespace, either by being owned or by being introduced by importing or inheritance.

+ /**ownedMember** : NamedElement [0..*] {readOnly, union, subsets member, subsets ownedElement}

A collection of NamedElements owned by the Namespace.

+ **ownedRule** : Constraint [0..*] {subsets ownedMember}

+ **packageImport** : PackageImport [0..*] {subsets ownedElement}

References the PackageImports owned by the Namespace.

Operations

+ **excludeCollisions** (imps : PackageableElement [0..*]) : PackageableElement [0..*] {query}

The query excludeCollisions() excludes from a set of PackageableElements any that would not be

Package InfrastructureLibrary::Core::Constructs

Class Namespace

distinguishable from each other in this namespace.

body (OCL): result = imps->reject(imp1 | imps.exists(imp2 | not imp1.isDistinguishableFrom (imp2, self)))

+ **getNamesOfMember** (element : NamedElement) : String [0..*] {query}

The query getNamesOfMember() takes importing into account. It gives back the set of names that an element would have in an importing namespace, either because it is owned, or if not owned then imported individually, or if not individually then from a package.

body (OCL): result = if self.ownedMember->includes(element) then Set{}->include(element. name) else let elementImports: ElementImport = self.elementImport->select(ei | ei. importedElement = element) in if elementImports->notEmpty() then elementImports->collect(el | el. getName()) else self.packageImport->select(pi | pi.importedPackage.visibleMembers()->includes (element))->collect(pi | pi.importedPackage.getNamesOfMember(element)) endif endif

+ **importMembers** (imps : PackageableElement [0..*]) : PackageableElement [0..*] {query}

The query importMembers() defines which of a set of PackageableElements are actually imported into the namespace. This excludes hidden ones, i.e., those which have names that conflict with names of owned members, and also excludes elements which would have the same name when imported.

body (OCL): result = self.excludeCollisions(imps)->select(imp | self.ownedMember->forAll(mem | mem.imp.isDistinguishableFrom(mem, self)))

+ **importedMember** () : PackageableElement [0..*] {query}

The importedMember property is derived from the ElementImports and the PackageImports. References the PackageableElements that are members of this Namespace as a result of either PackageImports or ElementImports.

body (OCL): result = self.importMembers(self.elementImport.importedElement.asSet()->union (self.packageImport.importedPackage->collect(p | p.visibleMembers())))

+ **membersAreDistinguishable** () : Boolean [1..1] {query}

The Boolean query membersAreDistinguishable() determines whether all of the namespace's members are distinguishable within it.

body (OCL): result = self.member->forAll(memb | self.member->excluding(memb)->forAll(other | memb.isDistinguishableFrom(other, self)))

Constraints

members_distinguishable

All the members of a Namespace are distinguishable within it.

Package InfrastructureLibrary::Core::Constructs

Class Namespace

expression (OCL): membersAreDistinguishable()

Package InfrastructureLibrary::Core::Constructs

Class OpaqueExpression

An opaque expression is an uninterpreted textual statement that denotes a (possibly empty) set of values when evaluated in a context.

Generalizations:

ValueSpecification

Found in Diagrams:

Expressions

Attributes

+ **body** : String [0..*] {ordered, nonunique}

The text of the expression, possibly in multiple languages.

+ **language** : String [0..*] {ordered}

Specifies the languages in which the expression is stated. The interpretation of the expression body depends on the languages. If the languages are unspecified, they might be implicit from the expression body or the context. Languages are matched to body strings by order.

Constraints

language_body_size

If the language attribute is not empty, then the size of the body and language arrays must be the same.

expression (OCL): language->notEmpty() implies (body->size() = language->size())

Package InfrastructureLibrary::Core::Constructs

Class Operation

An operation is a behavioral feature of a classifier that specifies the name, type, parameters, and constraints for invoking an associated behavior.

Generalizations:

BehavioralFeature

Attributes

+ /**isOrdered** : Boolean [1..1] = false

  This information is derived from the return result for this Operation.

+ **isQuery** : Boolean [1..1] = false

  Specifies whether an execution of the BehavioralFeature leaves the state of the system unchanged (isQuery=true) or whether side effects may occur (isQuery=false).

+ /**isUnique** : Boolean [1..1] = true

  This information is derived from the return result for this Operation.

+ /**lower** : Integer [0..1] = 1

  This information is derived from the return result for this Operation.

+ /**upper** : UnlimitedNatural [0..1] = 1

  This information is derived from the return result for this Operation.

Owned Association Ends

+ **bodyCondition** : Constraint [0..1] {subsets ownedRule}

+ **class** : Class [0..1] {subsets redefinitionContext, subsets namespace, subsets featuringClassifier}

  The class that owns the operation.

+ **datatype** : DataType [0..1] {subsets redefinitionContext, subsets namespace, subsets featuringClassifier}

  The DataType that owns this Operation.

+ **ownedParameter** : Parameter [0..*] {ordered, redefines ownedParameter}

Package InfrastructureLibrary::Core::Constructs

Class Operation

Specifies the ordered set of formal parameters of this BehavioralFeature.

+ **postcondition** : Constraint [0..*] {subsets ownedRule}

+ **precondition** : Constraint [0..*] {subsets ownedRule}

+ **raisedException** : Type [0..*] {redefines raisedException}

References the Types representing exceptions that may be raised during an invocation of this operation.

+ **redefinedOperation** : Operation [0..*] {subsets redefinedElement}

References the Operations that are redefined by this Operation.

+ /**type** : Type [0..1]

This information is derived from the return result for this Operation.

Operations

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two Operations in a context in which redefinition is possible, whether redefinition would be consistent in the sense of maintaining type covariance. Other senses of consistency may be required, for example to determine consistency in the sense of contravariance. Users may define alternative queries under names different from 'isConsistentWith ()', as for example, users may define a query named 'isContravariantWith()'.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = (redefinee.oclIsKindOf(Operation) and let op: Operation = redefinee. oclAsType(Operation) in self.ownedParameter.size() = op.ownedParameter.size() and forAll(i | op. ownedParameter[i].type.conformsTo(self.ownedParameter[i].type)))

+ **isOrdered** () : Boolean [1..1] {query}

If this operation has a return parameter, isOrdered equals the value of isOrdered for that parameter. Otherwise isOrdered is false.

body (OCL): result = if returnResult->size() = 1 then returnResult->any().isOrdered else false endif

+ **isUnique** () : Boolean [1..1] {query}

If this operation has a return parameter, isUnique equals the value of isUnique for that parameter. Otherwise isUnique is true.

Package InfrastructureLibrary::Core::Constructs

Class Operation

body (OCL): result = if returnResult->size() = 1 then returnResult->any().isUnique else true endif

+ **lower** () : Integer [1..1] {query}

If this operation has a return parameter, lower equals the value of lower for that parameter. Otherwise lower is not defined.

body (OCL): result = if returnResult->size() = 1 then returnResult->any().lower else Set{} endif

+ **returnResult** () : Parameter [0..*] {query}

body (OCL): result = ownedParameter->select (par | par.direction = #return)

+ **type** () : Type [1..1] {query}

If this operation has a return parameter, type equals the value of type for that parameter. Otherwise type is not defined.

body (OCL): result = if returnResult->size() = 1 then returnResult->any().type else Set{} endif

+ **upper** () : UnlimitedNatural [1..1] {query}

If this operation has a return parameter, upper equals the value of upper for that parameter. Otherwise upper is not defined.

body (OCL): result = if returnResult->size() = 1 then returnResult->any().upper else Set{} endif

Constraints

at_most_one_return

An operation can have at most one return parameter; i.e., an owned parameter with the direction set to 'return'

expression (OCL): self.ownedParameter->select(par | par.direction = #return)->size() <= 1

only_body_for_query

A bodyCondition can only be specified for a query operation.

expression (OCL): bodyCondition->notEmpty() implies isQuery

Package InfrastructureLibrary::Core::Constructs

Class Package

A package is used to group elements, and provides a namespace for the grouped elements.

Generalizations:

Namespace, PackageableElement

Owned Association Ends

+ /**nestedPackage** : Package [0..*] {subsets packagedElement}

References the packaged elements that are Packages.

+ **nestingPackage** : Package [0..1] {subsets namespace}

References the Package that owns this Package.

+ /**ownedType** : Type [0..*] {subsets packagedElement}

References the packaged elements that are Types.

+ **packageMerge** : PackageMerge [0..*] {subsets ownedElement}

References the PackageMerges that are owned by this Package.

+ **packagedElement** : PackageableElement [0..*] {subsets ownedMember}

Specifies the packageable elements that are owned by this Package.

Operations

+ **makesVisible** (el : NamedElement) : Boolean [1..1] {query}

The query makesVisible() defines whether a Package makes an element visible outside itself. Elements with no visibility and elements with public visibility are made visible.

precondition (): self.member->includes(el)

body (OCL): result = (ownedMember->includes(el)) or (elementImport-> select(ei|ei.visibility = #public)-> collect(ei|ei.importedElement)->includes(el)) or (packageImport-> select(pi|pi.visibility = #public)-> collect(pi| pi.importedPackage.member->includes(el))->notEmpty())

+ **mustBeOwned** () : Boolean [1..1] {query}

The query mustBeOwned() indicates whether elements of this type must have an owner.

body (OCL): result = false

+ **visibleMembers** () : PackageableElement [0..*] {query}

Package InfrastructureLibrary::Core::Constructs

Class Package

The query visibleMembers() defines which members of a Package can be accessed outside it.

body (OCL): result = member->select(m | self.makesVisible(m))

Constraints

elements_public_or_private

If an element that is owned by a package has visibility, it is public or private.

expression (OCL): self.ownedElements->forAll(e | e.visibility->notEmpty() implies e.visbility = #public or e.visibility = #private)

Package InfrastructureLibrary::Core::Constructs

Class PackageImport

A package import is a relationship that allows the use of unqualified names to refer to package members from other namespaces.

Generalizations:

DirectedRelationship

Found in Diagrams:

Profile Elements

Attributes

+ **visibility** : VisibilityKind [1..1] = public

Specifies the visibility of the imported PackageableElements within the importing Namespace, i.e., whether imported elements will in turn be visible to other packages that use that importingPackage as an importedPackage. If the PackageImport is public, the imported elements will be visible outside the package, while if it is private they will not.

Owned Association Ends

+ **importedPackage** : Package [1..1] {subsets target}

Specifies the Package whose members are imported into a Namespace.

+ **importingNamespace** : Namespace [1..1] {subsets source, subsets owner}

Specifies the Namespace that imports the members from a Package.

Constraints

public_or_private

The visibility of a PackageImport is either public or private.

expression (OCL): self.visibility = #public or self.visibility = #private

Package InfrastructureLibrary::Core::Constructs

Class PackageMerge

A package merge defines how the contents of one package are extended by the contents of another package.

Generalizations:

DirectedRelationship

Owned Association Ends

+ **mergedPackage** : Package [1..1] {subsets target}

References the Package that is to be merged with the receiving package of the PackageMerge.

+ **receivingPackage** : Package [1..1] {subsets source, subsets owner}

References the Package that is being extended with the contents of the merged package of the PackageMerge.

Package InfrastructureLibrary::Core::Constructs

Class PackageableElement

A packageable element indicates a named element that may be owned directly by a package.

Generalizations:

NamedElement

Specializations:

Constraint, Package, Type, ValueSpecification

Found in Diagrams:

Expressions

Package InfrastructureLibrary::Core::Constructs

Class Parameter

A parameter is a specification of an argument used to pass information into or out of an invocation of a behavioral feature.

Generalizations:

MultiplicityElement, TypedElement

Attributes

+ **default** : String [0..1]

Specifies a String that represents a value to be used when no argument is supplied for the Parameter.

+ **direction** : ParameterDirectionKind [1..1] = in

Indicates whether a parameter is being sent into or out of a behavioral element.

Owned Association Ends

+ **operation** : Operation [0..1] {subsets namespace}

References the Operation owning this parameter.

Package InfrastructureLibrary::Core::Constructs

Class PrimitiveType

A primitive type defines a predefined data type, without any relevant substructure (i.e., it has no parts in the context of UML). A primitive datatype may have an algebra and operations defined outside of UML, for example, mathematically.

Generalizations:

DataType

Package InfrastructureLibrary::Core::Constructs

Class Property

A property is a structural feature of a classifier that characterizes instances of the classifier. A property related by ownedAttribute to a classifier (other than an association) represents an attribute and might also represent an association end. It relates an instance of the class to a value or set of values of the type of the attribute. A property related by memberEnd or its specializations to an association represents an end of the association. The type of the property is the type of the end of the association.

Generalizations:

StructuralFeature

Specializations:

ExtensionEnd

Found in Diagrams:

Profile Elements

Attributes

+ **default** : String [0..1]

Specifies a String that represents a value to be used when no argument is supplied for the Property.

+ **isComposite** : Boolean [1..1] = false

If isComposite is true, the object containing the attribute is a container for the object or value contained in the attribute.

+ **isDerived** : Boolean [1..1] = false

If isDerived is true, the value of the attribute is derived from information elsewhere.

+ **isDerivedUnion** : Boolean [1..1] = false

Specifies whether the property is derived as the union of all of the properties that are constrained to subset it.

+ **isReadOnly** : Boolean [1..1] = false

If isReadOnly is true, the attribute may not be written to after initialization.

Owned Association Ends

+ **association** : Association [0..1]

References the association of which this property is a member, if any.

Package InfrastructureLibrary::Core::Constructs

Class Property

+ **class** : Class [0..1] {subsets namespace, subsets featuringClassifier, subsets classifier}

References the Class that owns the Property.

+ **datatype** : DataType [0..1] {subsets namespace, subsets featuringClassifier, subsets classifier}

The DataType that owns this Property.

+ /**opposite** : Property [0..1]

In the case where the property is one navigable end of a binary association with both ends navigable, this gives the other end.

+ **owningAssociation** : Association [0..1] {subsets association, subsets namespace, subsets featuringClassifier}

References the owning association of this property, if any.

+ **redefinedProperty** : Property [0..*] {subsets redefinedElement}

References the properties that are redefined by this property.

+ **subsettedProperty** : Property [0..*]

References the properties of which this property is constrained to be a subset.

Operations

+ **isAttribute** (p : Property) : Boolean [1..1] {query}

The query isAttribute() is true if the Property is defined as an attribute of some classifier.

body (OCL): result = Classifier->allInstances->exists(c | c.attribute->includes(p))

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two Properties in a context in which redefinition is possible, whether redefinition would be logically consistent. A redefining property is consistent with a redefined property if the type of the redefining property conforms to the type of the redefined property, the multiplicity of the redefining property (if specified) is contained in the multiplicity of the redefined property, and the redefining property is derived if the redefined property is derived.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = redefinee.oclIsKindOf(Property) and let prop : Property = redefinee.

Package InfrastructureLibrary::Core::Constructs

Class Property

oclAsType(Property) in (prop.type.conformsTo(self.type) and ((prop.lowerBound()->notEmpty() and self.lowerBound()->notEmpty()) implies prop.lowerBound() >= self.lowerBound()) and ((prop.upperBound()->notEmpty() and self.upperBound()->notEmpty()) implies prop.lowerBound() <= self.lowerBound()) and (self.isDerived implies prop.isDerived) and (self.isComposite implies prop.isComposite))

+ **isNavigable** () : Boolean [1..1] {query}

The query isNavigable() indicates whether it is possible to navigate across the property.

body (OCL): result = not classifier->isEmpty() or association.owningAssociation.navigableOwnedEnd->includes(self)

+ **opposite** () : Property [1..1] {query}

If this property is owned by a class, associated with a binary association, and the other end of the association is also owned by a class, then opposite gives the other end.

body (OCL): result = if owningAssociation->isEmpty() and association.memberEnd->size() = 2 then let otherEnd = (association.memberEnd - self)->any() in if otherEnd.owningAssociation->isEmpty() then otherEnd else Set{} endif else Set {} endif

+ **subsettingContext** () : Classifier [0..*] {query}

The query subsettingContext() gives the context for subsetting a property. It consists, in the case of an attribute, of the corresponding classifier, and in the case of an association end, all of the classifiers at the other ends.

body (OCL): result = if association->notEmpty() then association.endType-type else if classifier->notEmpty then Set{classifier} else Set{} endif endif

Constraints

derived_union_is_derived

A derived union is derived.

expression (OCL): isDerivedUnion implies isDerived

multiplicity_of_composite

A multiplicity of a composite aggregation must not have an upper bound greater than 1.

expression (OCL): isComposite implies (upperBound()->isEmpty() or upperBound() <= 1)

navigable_readonly

Only a navigable property can be marked as readOnly.

expression (OCL): isReadOnly implies isNavigable()

Package InfrastructureLibrary::Core::Constructs

Class Property

redefined_property_inherited

A redefined property must be inherited from a more general classifier containing the redefining property.

expression (OCL): if (redefinedProperty->notEmpty()) then (redefinitionContext->notEmpty() and redefinedProperty->forAll(rp| ((redefinitionContext->collect(fc| fc.allParents()))->asSet())->collect (c| c.allFeatures())->asSet()->includes(rp))

subsetted_property_names

A property may not subset a property with the same name.

expression (OCL): true

subsetting_context_conforms

Subsetting may only occur when the context of the subsetting property conforms to the context of the subsetted property.

expression (OCL): self.subsettedProperty->notEmpty() implies (self.subsettingContext()-> notEmpty() and self.subsettingContext()->forAll (sc | self.subsettedProperty->forAll(sp | sp. subsettingContext()->exists(c | sc.conformsTo(c)))))

subsetting_rules

A subsetting property may strengthen the type of the subsetted property, and its upper bound may be less.

expression (OCL): self.subsettedProperty->forAll(sp | self.type.conformsTo(sp.type) and ((self. upperBound()->notEmpty() and sp.upperBound()->notEmpty()) implies self.upperBound()<=sp. upperBound()))

Package InfrastructureLibrary::Core::Constructs

Class RedefinableElement

A redefinable element is an element that, when defined in the context of a classifier, can be redefined more specifically or differently in the context of another classifier that specializes (directly or indirectly) the context classifier.

Generalizations:

NamedElement

Specializations:

Feature

Found in Diagrams:

Classifiers

Attributes

+ **isLeaf** : Boolean [1..1] = false

Indicates whether it is possible to further redefine a RedefinableElement. If the value is true, then it is not possible to further redefine the RedefinableElement. Note that this property is preserved through package merge operations; that is, the capability to redefine a RedefinableElement (i.e., isLeaf=false) must be preserved in the resulting RedefinableElement of a package merge operation where a RedefinableElement with isLeaf=false is merged with a matching RedefinableElement with isLeaf=true: the resulting RedefinableElement will have isLeaf=false. Default value is false.

Owned Association Ends

+ /**redefinedElement** : RedefinableElement [0..*] {readOnly, union}

The redefinable element that is being redefined by this element.

+ /**redefinitionContext** : Classifier [0..*] {readOnly, union}

References the contexts that this element may be redefined from.

Operations

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two RedefinableElements in a context in which redefinition is possible, whether redefinition would be logically consistent. By default, this is false; this operation must be overridden for subclasses of RedefinableElement to define the consistency conditions.

precondition (): redefinee.isRedefinitionContextValid(self)

Package InfrastructureLibrary::Core::Constructs

Class RedefinableElement

body (OCL): result = false

+ **isRedefinitionContextValid** (redefined : RedefinableElement) : Boolean [1..1] {query}

The query isRedefinitionContextValid() specifies whether the redefinition contexts of this RedefinableElement are properly related to the redefinition contexts of the specified RedefinableElement to allow this element to redefine the other. By default at least one of the redefinition contexts of this element must be a specialization of at least one of the redefinition contexts of the specified element.

body (OCL): result = self.redefinitionContext->exists(c | redefined.redefinitionContext->exists(r | c.allParents()->includes(r)))

Constraints

non_leaf_redefinition

A redefinable element can only redefine non-leaf redefinable elements

expression (OCL): self.redefinedElement->forAll(not isLeaf)

redefinition_consistent

A redefining element must be consistent with each redefined element.

expression (OCL): self.redefinedElement->forAll(re | re.isConsistentWith(self))

redefinition_context_valid

At least one of the redefinition contexts of the redefining element must be a specialization of at least one of the redefinition contexts for each redefined element.

expression (OCL): self.redefinedElement->forAll(e | self.isRedefinitionContextValid(e))

Package InfrastructureLibrary::Core::Constructs

Class Relationship

Relationship is an abstract concept that specifies some kind of relationship between elements.

Generalizations:

Element

Specializations:

Association, DirectedRelationship

Owned Association Ends

+ /**relatedElement** : Element [1..*] {readOnly, union}

Specifies the elements related by the Relationship.

Package InfrastructureLibrary::Core::Constructs

Class StructuralFeature

A structural feature is a typed feature of a classifier that specifies the structure of instances of the classifier.

Generalizations:

Feature, MultiplicityElement, TypedElement

Specializations:

Property

Found in Diagrams:

Classifiers

Package InfrastructureLibrary::Core::Constructs

Class Type

A type is a named element that is used as the type for a typed element. A type can be contained in a package.

Generalizations:

NamedElement, PackageableElement

Specializations:

Classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **package** : Package [0..1] {subsets namespace}

Specifies the owning package of this classifier, if any.

Operations

+ **conformsTo** (other : Type) : Boolean [1..1] {query}

The query conformsTo() gives true for a type that conforms to another. By default, two types do not conform to each other. This query is intended to be redefined for specific conformance situations.

body (OCL): result = false

Package InfrastructureLibrary::Core::Constructs

Class TypedElement

A typed element is a kind of named element that represents an element with a type.

Generalizations:

NamedElement

Specializations:

Parameter, StructuralFeature, ValueSpecification

Found in Diagrams:

Classifiers, Expressions

Owned Association Ends

+ **type** : Type [0..1]

This information is derived from the return result for this Operation.

Package InfrastructureLibrary::Core::Constructs

Class ValueSpecification

A value specification is the specification of a (possibly empty) set of instances, including both objects and data values.

Generalizations:

PackageableElement, TypedElement

Specializations:

Expression, OpaqueExpression

Found in Diagrams:

Expressions

Operations

+ **booleanValue** () : Boolean [1..1] {query}

The query booleanValue() gives a single Boolean value when one can be computed.

body (OCL): result = Set{}

+ **integerValue** () : Integer [1..1] {query}

The query integerValue() gives a single Integer value when one can be computed.

body (OCL): result = Set{}

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() determines whether a value specification can be computed in a model. This operation cannot be fully defined in OCL. A conforming implementation is expected to deliver true for this operation for all value specifications that it can compute, and to compute all of those for which the operation is true. A conforming implementation is expected to be able to compute the value of all literals.

body (OCL): result = false

+ **isNull** () : Boolean [1..1] {query}

The query isNull() returns true when it can be computed that the value is null.

body (OCL): result = false

+ **stringValue** () : String [1..1] {query}

The query stringValue() gives a single String value when one can be computed.

body (OCL): result = Set{}

+ **unlimitedValue** () : UnlimitedNatural [1..1] {query}

The query unlimitedValue() gives a single UnlimitedNatural value when one can be computed.

Package InfrastructureLibrary::Core::Constructs

Class ValueSpecification

body (OCL): result = Set{}

Package InfrastructureLibrary::Core::Constructs

Enumeration ParameterDirectionKind

Parameter direction kind is an enumeration type that defines literals used to specify direction of parameters.

Enumeration Literals

in

Indicates that parameter values are passed into the behavioral element by the caller.

inout

Indicates that parameter values are passed into a behavioral element by the caller and then back out to the caller from the behavioral element.

out

Indicates that parameter values are passed from a behavioral element out to the caller.

return

Indicates that parameter values are passed as return values from a behavioral element back to the caller.

Package InfrastructureLibrary::Core::Constructs

Enumeration VisibilityKind

VisibilityKind is an enumeration type that defines literals to determine the visibility of elements in a model.

Enumeration Literals

package

A package element is owned by a namespace that is not a package, and is visible to elements that are in the same package as its owning namespace. Only named elements that are not owned by packages can be marked as having package visibility. Any element marked as having package visibility is visible to all elements within the nearest enclosing package (given that other owning elements have proper visibility). Outside the nearest enclosing package, an element marked as having package visibility is not visible.

private

A private element is only visible inside the namespace that owns it.

protected

A protected element is visible to elements that have a generalization relationship to the namespace that owns it.

public

A public element is visible to all elements that can access the contents of the namespace that owns it.

Operations

+ **bestVisibility** (vis : VisibilityKind [0..*]) : VisibilityKind [1..1] {query}

The query bestVisibility() examines a set of VisibilityKinds, and returns public as the preferred visibility.

body (OCL): result = if vis->includes(#public) then #public else #private endif

Package InfrastructureLibrary::Core::Constructs

Association A_annotatedElement_comment

Member Ends:

annotatedElement, comment

Owned Association Ends

+ **comment** : Comment [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_attribute_classifier

Member Ends:

attribute, classifier

Owned Association Ends

+ **classifier** : Classifier [0..1] {subsets redefinitionContext}

Package InfrastructureLibrary::Core::Constructs

Association A_bodyCondition_bodyContext

Member Ends:

bodyCondition, bodyContext

Owned Association Ends

+ **bodyContext** : Operation [0..1] {subsets context}

Package InfrastructureLibrary::Core::Constructs

Association A_constrainedElement_constraint

Member Ends:

constrainedElement, constraint

Owned Association Ends

+ **constraint** : Constraint [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_elementImport_importingNamespace

Member Ends:

elementImport, importingNamespace

Package InfrastructureLibrary::Core::Constructs

Association A_endType_association

Member Ends:

endType, association

Owned Association Ends

+ **association** : Association [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_feature_featuringClassifier

Member Ends:

feature, featuringClassifier

Found in Diagrams:

Classifiers

Package InfrastructureLibrary::Core::Constructs

Association A_general_classifier

Member Ends:

general, classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_importedElement_elementImport

Member Ends:

importedElement, elementImport

Owned Association Ends

+ **elementImport** : ElementImport [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_importedMember_namespace

Member Ends:

importedMember, namespace

Owned Association Ends

+ **namespace** : Namespace [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_importedPackage_packageImport

Member Ends:

importedPackage, packageImport

Owned Association Ends

+ **packageImport** : PackageImport [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_inheritedMember_classifier

Member Ends:

inheritedMember, classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_memberEnd_association

Member Ends:

memberEnd, association

Package InfrastructureLibrary::Core::Constructs

Association A_member_namespace

Member Ends:

member, namespace

Owned Association Ends

+ **namespace** : Namespace [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_mergedPackage_packageMerge

Member Ends:

mergedPackage, packageMerge

Owned Association Ends

+ **packageMerge** : PackageMerge [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_navigableOwnedEnd_association

Member Ends:

navigableOwnedEnd, association

Owned Association Ends

+ **association** : Association [0..1]

Package InfrastructureLibrary::Core::Constructs

Association A_nestedPackage_nestingPackage

Member Ends:

nestedPackage, nestingPackage

Package InfrastructureLibrary::Core::Constructs

Association A_operand_expression

Member Ends:

operand, expression

Found in Diagrams:

Expressions

Owned Association Ends

+ **expression** : Expression [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Constructs

Association A_opposite_property

Member Ends:

opposite, property

Owned Association Ends

+ **property** : Property [0..1]

Package InfrastructureLibrary::Core::Constructs

Association A_ownedAttribute_class

Member Ends:

ownedAttribute, class

Package InfrastructureLibrary::Core::Constructs

Association A_ownedAttribute_datatype

Member Ends:

ownedAttribute, datatype

Package InfrastructureLibrary::Core::Constructs

Association A_ownedComment_owningElement

Member Ends:

ownedComment, owningElement

Owned Association Ends

+ **owningElement** : Element [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Constructs

Association A_ownedElement_owner

Member Ends:

ownedElement, owner

Package InfrastructureLibrary::Core::Constructs

Association A_ownedEnd_owningAssociation

Member Ends:

ownedEnd, owningAssociation

Package InfrastructureLibrary::Core::Constructs

Association A_ownedLiteral_enumeration

Member Ends:

ownedLiteral, enumeration

Package InfrastructureLibrary::Core::Constructs

Association A_ownedMember_namespace

Member Ends:

ownedMember, namespace

Package InfrastructureLibrary::Core::Constructs

Association A_ownedOperation_class

Member Ends:

ownedOperation, class

Package InfrastructureLibrary::Core::Constructs

Association A_ownedOperation_datatype

Member Ends:

ownedOperation, datatype

Package InfrastructureLibrary::Core::Constructs

Association A_ownedParameter_operation

Member Ends:

ownedParameter, operation

Package InfrastructureLibrary::Core::Constructs

Association A_ownedParameter_ownerFormalParam

Member Ends:

ownedParameter, ownerFormalParam

Owned Association Ends

+ **ownerFormalParam** : BehavioralFeature [0..1] {subsets namespace}

Package InfrastructureLibrary::Core::Constructs

Association A_ownedRule_context

Member Ends:

ownedRule, context

Package InfrastructureLibrary::Core::Constructs

Association A_ownedType_package

Member Ends:

ownedType, package

Package InfrastructureLibrary::Core::Constructs

Association A_packageImport_importingNamespace

Member Ends:

packageImport, importingNamespace

Package InfrastructureLibrary::Core::Constructs

Association A_packageMerge_receivingPackage

Member Ends:

packageMerge, receivingPackage

Package InfrastructureLibrary::Core::Constructs

Association A_packagedElement_owningPackage

Member Ends:

packagedElement, owningPackage

Owned Association Ends

+ **owningPackage** : Package [0..1] {subsets namespace}

Package InfrastructureLibrary::Core::Constructs

Association A_postcondition_postContext

Member Ends:

postcondition, postContext

Owned Association Ends

+ **postContext** : Operation [0..1] {subsets context}

Package InfrastructureLibrary::Core::Constructs

Association A_precondition_preContext

Member Ends:

precondition, preContext

Owned Association Ends

+ **preContext** : Operation [0..1] {subsets context}

Package InfrastructureLibrary::Core::Constructs

Association A_raisedException_behavioralFeature

Member Ends:

raisedException, behavioralFeature

Owned Association Ends

+ **behavioralFeature** : BehavioralFeature [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_raisedException_operation

Member Ends:

raisedException, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_redefinedElement_redefinableElement

Member Ends:

redefinedElement, redefinableElement

Found in Diagrams:

Classifiers

Owned Association Ends

+ **redefinableElement** : RedefinableElement [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_redefinedOperation_operation

Member Ends:

redefinedOperation, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_redefinedProperty_property

Member Ends:

redefinedProperty, property

Owned Association Ends

+ **property** : Property [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_redefinitionContext_redefinableElement

Member Ends:

redefinitionContext, redefinableElement

Found in Diagrams:

Classifiers

Owned Association Ends

+ **redefinableElement** : RedefinableElement [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_relatedElement_relationship

Member Ends:

relatedElement, relationship

Owned Association Ends

+ **relationship** : Relationship [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_source_directedRelationship

Member Ends:

source, directedRelationship

Owned Association Ends

+ **directedRelationship** : DirectedRelationship [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_specification_owningConstraint

Member Ends:

specification, owningConstraint

Owned Association Ends

+ **owningConstraint** : Constraint [0..1] {subsets owner}

Package InfrastructureLibrary::Core::Constructs

Association A_subsettedProperty_property

Member Ends:

subsettedProperty, property

Owned Association Ends

+ **property** : Property [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_superClass_class

Member Ends:

superClass, class

Owned Association Ends

+ **class** : Class [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_target_directedRelationship

Member Ends:

target, directedRelationship

Owned Association Ends

+ **directedRelationship** : DirectedRelationship [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_type_operation

Member Ends:

type, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package InfrastructureLibrary::Core::Constructs

Association A_type_typedElement

Member Ends:

type, typedElement

Found in Diagrams:

Classifiers

Owned Association Ends

+ **typedElement** : TypedElement [0..*]

Package InfrastructureLibrary::Core::PrimitiveTypes

Nesting Package:

Core

PrimitiveType Summary
Boolean
Integer
String
UnlimitedNatural

Package InfrastructureLibrary::Core::PrimitiveTypes

PrimitiveType Boolean

A Boolean type is used for logical expression, consisting of the predefined values true and false.

Package InfrastructureLibrary::Core::PrimitiveTypes

PrimitiveType Integer

An integer is a primitive type representing integer values.

Package InfrastructureLibrary::Core::PrimitiveTypes

PrimitiveType String

A string is a sequence of characters in some suitable character set used to display information about the model. Character sets may include non-Roman alphabets and characters.

Package InfrastructureLibrary::Core::PrimitiveTypes

PrimitiveType UnlimitedNatural

An unlimited natural is a primitive type representing unlimited natural values.

Package InfrastructureLibrary::Profiles

Nesting Package:

InfrastructureLibrary

Imported Packages:

Constructs, PrimitiveTypes

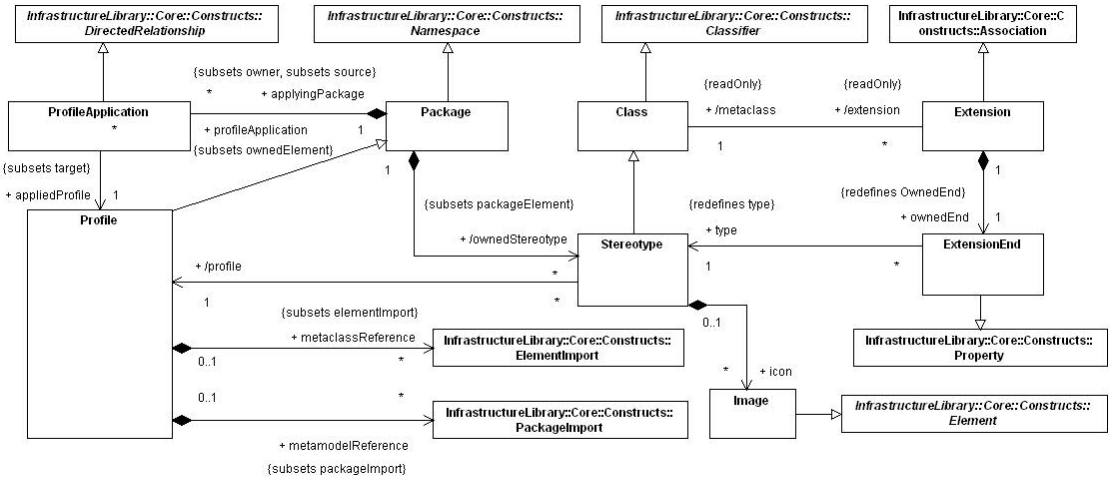
Diagram Summary
Profile Elements

Class Summary
Class
Extension
ExtensionEnd
Image
NamedElement
Package
Profile
ProfileApplication
Stereotype

Association Summary
A_appliedProfile_profileApplication
A_extension_metaclass
A_icon_stereotype
A_metaclassReference_profile
A_metamodelReference_profile
A_ownedEnd_extension
A_ownedStereotype_profile
A_profileApplication_applyingPackage
A_profile_stereotype
A_type_extensionEnd

Package InfrastructureLibrary::Profiles

Diagram Profile Elements



Classifiers Local to Package:

Class, Extension, ExtensionEnd, Image, Package, Profile, ProfileApplication, Stereotype

Classifiers External to Package:

Association, Classifier, DirectedRelationship, Element, ElementImport, Namespace, PackageImport , Property

Package InfrastructureLibrary::Profiles

Class Class

Class has derived association that indicates how it may be extended through one or more stereotypes. Stereotype is the only kind of metaclass that cannot be extended by stereotypes.

Generalizations:

Classifier

Specializations:

Stereotype

Found in Diagrams:

Profile Elements

Owned Association Ends

+ /**extension** : Extension [0..*] {readOnly}

References the Extensions that specify additional properties of the metaclass. The property is derived from the extensions whose memberEnds are typed by the Class.

Package InfrastructureLibrary::Profiles

Class Extension

An extension is used to indicate that the properties of a metaclass are extended through a stereotype, and gives the ability to flexibly add (and later remove) stereotypes to classes.

Generalizations:

Association

Found in Diagrams:

Profile Elements

Attributes

+ /**isRequired** : Boolean [1..1] = false {readOnly}

Indicates whether an instance of the extending stereotype must be created when an instance of the extended class is created. The attribute value is derived from the value of the lower property of the ExtensionEnd referenced by Extension::ownedEnd; a lower value of 1 means that isRequired is true, but otherwise it is false. Since the default value of ExtensionEnd::lower is 0, the default value of isRequired is false.

Owned Association Ends

+ /**metaclass** : Class [1..1] {readOnly}

References the Class that is extended through an Extension. The property is derived from the type of the memberEnd that is not the ownedEnd.

+ **ownedEnd** : ExtensionEnd [1..1] {redefines ownedEnd}

References the end of the extension that is typed by a Stereotype.

Operations

+ **isRequired** () : Boolean [1..1] {query}

The query isRequired() is true if the owned end has a multiplicity with the lower bound of 1.

body (OCL): result = (ownedEnd->lowerBound() = 1)

+ **metaclass** () : Class [1..1] {query}

The query metaclass() returns the metaclass that is being extended (as opposed to the extending stereotype).

body (OCL): result = metaclassEnd().type

+ **metaclassEnd** () : Property [1..1] {query}

Package InfrastructureLibrary::Profiles

Class Extension

The query metaclassEnd() returns the Property that is typed by a metaclass (as opposed to a stereotype).

body (OCL): result = memberEnd->reject(ownedEnd)

Constraints

is_binary

An Extension is binary, i.e., it has only two memberEnds.

expression (OCL): memberEnd->size() = 2

non_owned_end

The non-owned end of an Extension is typed by a Class.

expression (OCL): metaclassEnd()->notEmpty() and metaclass()->oclIsKindOf(Class)

Package InfrastructureLibrary::Profiles

Class ExtensionEnd

An extension end is used to tie an extension to a stereotype when extending a metaclass.

Generalizations:

Property

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **lower** : [0..1] = 0 {redefines lower}

This redefinition changes the default multiplicity of association ends, since model elements are usually extended by 0 or 1 instance of the extension stereotype.

+ **type** : Stereotype [1..1] {redefines type}

References the type of the ExtensionEnd. Note that this association restricts the possible types of an ExtensionEnd to only be Stereotypes.

Operations

+ **lowerBound** () : Integer [1..1] {query}

The query lowerBound() returns the lower bound of the multiplicity as an Integer. This is a redefinition of the default
lower bound, which normally, for MultiplicityElements, evaluates to 1 if empty.

body (OCL): result = lowerBound = if lowerValue->isEmpty() then 0 else lowerValue->IntegerValue() endif

Constraints

aggregation

The aggregation of an ExtensionEnd is composite.

expression (OCL): self.aggregation = #composite

multiplicity

The multiplicity of ExtensionEnd is 0..1 or 1.

expression (OCL): (self->lowerBound() = 0 or self->lowerBound() = 1) and self->upperBound() = 1

Package InfrastructureLibrary::Profiles

Class Image

Physical definition of a graphical image.

Generalizations:

Element

Found in Diagrams:

Profile Elements

Attributes

+ **content** : String [0..1]

This contains the serialization of the image according to the format. The value could represent a bitmap, image such as a GIF file, or drawing 'instructions' using a standard such as Scalable Vector Graphic (SVG) (which is XML based).

+ **format** : String [0..1]

This indicates the format of the content - which is how the string content should be interpreted. The following values are reserved: SVG, GIF, PNG, JPG, WMF, EMF, BMP.

In addition the prefix 'MIME: ' is also reserved. This option can be used as an alternative to express the reserved values above, for example "SVG" could instead be expressed as "MIME: image/svg+xml".

+ **location** : String [0..1]

This contains a location that can be used by a tool to locate the image as an alternative to embedding it in the stereotype.

Package InfrastructureLibrary::Profiles

Class NamedElement

Operations

+ **allOwningPackages** () : Package [0..*] {query}

The query allOwningPackages() returns all the directly or indirectly owning packages.

body (OCL): result = self.namespace->select(p | p.oclIsKindOf(Package))->union(p.allOwningPackages())

Package InfrastructureLibrary::Profiles

Class Package

A package can have one or more profile applications to indicate which profiles have been applied. Because a profile is a package, it is possible to apply a profile not only to packages, but also to profiles.

Generalizations:

Namespace

Specializations:

Profile

Found in Diagrams:

Profile Elements

Owned Association Ends

+ /**ownedStereotype** : Stereotype [0..*] {subsets packagedElement}

References the Stereotypes that are owned by the Package

+ **profileApplication** : ProfileApplication [0..*] {subsets ownedElement}

References the ProfileApplications that indicate which profiles have been applied to the Package.

Operations

+ **allApplicableStereotypes** () : Stereotype [0..*] {query}

The query allApplicableStereotypes() returns all the directly or indirectly owned stereotypes, including stereotypes contained in sub-profiles.

body (OCL): result = self.ownedStereotype->union(self.ownedMember-> select(oclIsKindOf (Package)).oclAsType(Package).allApplicableStereotypes()->flatten())->asSet()

+ **containingProfile** () : Profile [0..1] {query}

The query containingProfile() returns the closest profile directly or indirectly containing this package (or this package itself, if it is a profile).

body (OCL): result = if self.oclIsKindOf(Profile) then self.oclAsType(Profile) else self.namespace. oclAsType(Package).containingProfile() endif

Package InfrastructureLibrary::Profiles

Class Profile

A profile defines limited extensions to a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain.

Generalizations:

Package

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **metaclassReference** : ElementImport [0..*] {subsets elementImport}

References a metaclass that may be extended.

+ **metamodelReference** : PackageImport [0..*] {subsets packageImport}

References a package containing (directly or indirectly) metaclasses that may be extended.

Constraints

metaclass_reference_not_specialized

An element imported as a metaclassReference is not specialized or generalized in a Profile.

expression (OCL): self.metaclassReference.importedElement-> select(c | c.oclIsKindOf(Classifier) and (c.generalization.namespace = self or c.specialization.namespace = self))->isEmpty()

references_same_metamodel

All elements imported either as metaclassReferences or through metamodelReferences are members of the same base reference metamodel.

expression (OCL): self.metamodelReference.importedPackage.elementImport.importedElement. allOwningPackages()-> union(self.metaclassReference.importedElement.allOwningPackages())-> notEmpty()

Package InfrastructureLibrary::Profiles

Class ProfileApplication

A profile application is used to show which profiles have been applied to a package.

Generalizations:

DirectedRelationship

Found in Diagrams:

Profile Elements

Attributes

+ **isStrict** : Boolean [1..1] = false

Specifies that the Profile filtering rules for the metaclasses of the referenced metamodel shall be strictly applied.

Owned Association Ends

+ **appliedProfile** : Profile [1..1] {subsets target}

References the Profiles that are applied to a Package through this ProfileApplication.

+ **applyingPackage** : Package [1..1] {subsets source, subsets owner}

The package that owns the profile application.

Package InfrastructureLibrary::Profiles

Class Stereotype

A stereotype defines how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass.

Generalizations:

Class

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **icon** : Image [0..*]

Stereotype can change the graphical appearance of the extended model element by using attached icons. When this association is not null, it references the location of the icon content to be displayed within diagrams presenting the extended model elements.

+ /**profile** : Profile [1..1]

The profile that directly or indirectly contains this stereotype.

Operations

+ **containingProfile** () : Profile [1..1] {query}

The query containingProfile returns the closest profile directly or indirectly containing this stereotype.

body (OCL): result = self.namespace.oclAsType(Package).containingProfile()

+ **profile** () : Profile [1..1] {query}

A stereotype must be contained, directly or indirectly, in a profile.

body (OCL): result = self.containingProfile()

Constraints

generalize

A Stereotype may only generalize or specialize another Stereotype.

expression (OCL): generalization.general->forAll(e |e.oclIsKindOf(Stereotype)) and generalization.specific->forAll(e | e.oclIsKindOf(Stereotype))

name_not_clash

Package InfrastructureLibrary::Profiles

Class Stereotype

Stereotype names should not clash with keyword names for the extended model element.

expression (OCL): true

Package InfrastructureLibrary::Profiles

Association A_appliedProfile_profileApplication

Member Ends:

appliedProfile, profileApplication

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **profileApplication** : ProfileApplication [0..*]

Package InfrastructureLibrary::Profiles

Association A_extension_metaclass

Member Ends:

extension, metaclass

Found in Diagrams:

Profile Elements

Package InfrastructureLibrary::Profiles

Association A_icon_stereotype

Member Ends:

icon, stereotype

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **stereotype** : Stereotype [0..1]

Package InfrastructureLibrary::Profiles

Association A_metaclassReference_profile

Member Ends:

metaclassReference, profile

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **profile** : Profile [0..1]

Package InfrastructureLibrary::Profiles

Association A_metamodelReference_profile

Member Ends:

metamodelReference, profile

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **profile** : Profile [0..1]

Package InfrastructureLibrary::Profiles

Association A_ownedEnd_extension

Member Ends:

ownedEnd, extension

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **extension** : Extension [1..1]

Package InfrastructureLibrary::Profiles

Association A_ownedStereotype_profile

Member Ends:

ownedStereotype, profile

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **profile** : Package [1..1]

Package InfrastructureLibrary::Profiles

Association A_profileApplication_applyingPackage

Member Ends:

profileApplication, applyingPackage

Found in Diagrams:

Profile Elements

Package InfrastructureLibrary::Profiles

Association A_profile_stereotype

Member Ends:

profile, stereotype

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **stereotype** : Stereotype [0..*]

Package InfrastructureLibrary::Profiles

Association A_type_extensionEnd

Member Ends:

type, extensionEnd

Found in Diagrams:

Profile Elements

Owned Association Ends

+ **extensionEnd** : ExtensionEnd [0..*]

L0

Package L0

Merged Packages:

Basic, PrimitiveTypes

L1

Package L1

Merged Packages:

BasicActions, BasicActivities, BasicBehaviors, BasicInteractions, Communications, Dependencies, FundamentalActivities, Interfaces, InternalStructures, Kernel, UseCases

L2

Package L2

Merged Packages:

Artifacts, BasicComponents, BehaviorStateMachines, Fragments, IntermediateActions, IntermediateActivities, InvocationActions, L1, Nodes, Ports, Profiles, SimpleTime, StructuredActions, StructuredActivities, StructuredClasses

L3

Package L3

Merged Packages:

AssociationClasses, Collaborations, CompleteActions, CompleteActivities, CompleteStructuredActivities, ComponentDeployments, ExtraStructuredActivities, InformationFlows, L2, Models, PackagingComponents, PowerTypes, ProtocolStateMachines, StructuredActivities, Templates

LM

Package LM

Merged Packages:

Constructs, PrimitiveTypes

Superstructure

Package UML

Nested Package Summary
Actions
Activities
AuxiliaryConstructs
Classes
CommonBehaviors
Components
CompositeStructures
Deployments
Interactions
StateMachines
UseCases

Package UML::Actions

Nesting Package:

UML

Imported Packages:

Activities

Nested Package Summary
BasicActions
CompleteActions
IntermediateActions
StructuredActions

Package UML::Actions::BasicActions

Nesting Package:

Actions

Imported Packages:

Communications, Kernel

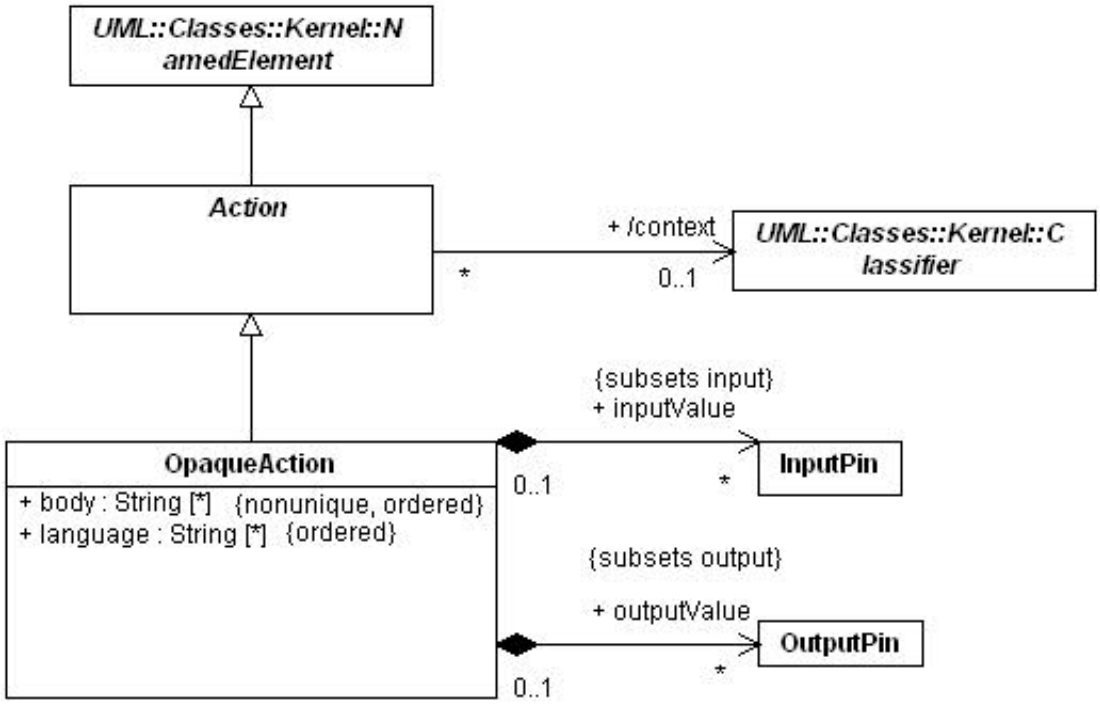
Diagram Summary
Basic Actions

Class Summary
Action
CallAction
CallBehaviorAction
CallOperationAction
InputPin
InvocationAction
MultiplicityElement
OpaqueAction
OutputPin
Pin
SendSignalAction
ValuePin

Association Summary
A_argument_invocationAction
A_behavior_callBehaviorAction
A_context_action
A_inputValue_opaqueAction
A_input_action
A_operation_callOperationAction
A_outputValue_opaqueAction
A_output_action
A_result_callAction
A_signal_sendSignalAction
A_target_callOperationAction
A_target_sendSignalAction
A_value_valuePin

Package UML::Actions::BasicActions

Diagram Basic Actions



Classifiers Local to Package:

Action, InputPin, OpaqueAction, OutputPin

Classifiers External to Package:

Classifier, NamedElement

Package UML::Actions::BasicActions

Class Action

An action is a named element that is the fundamental unit of executable functionality. The execution of an action represents some transformation or processing in the modeled system, be it a computer system or otherwise.

Generalizations:

NamedElement

Specializations:

AcceptEventAction, ClearAssociationAction, CreateObjectAction, DestroyObjectAction, InvocationAction, LinkAction, OpaqueAction, RaiseExceptionAction, ReadExtentAction, ReadIsClassifiedObjectAction, ReadLinkObjectEndAction, ReadLinkObjectEndQualifierAction, ReadSelfAction, ReclassifyObjectAction, ReduceAction, ReplyAction, StartClassifierBehaviorAction, StructuralFeatureAction, StructuredActivityNode, TestIdentityAction, UnmarshallAction, ValueSpecificationAction, VariableAction

Found in Diagrams:

Basic Actions, Complete Structured Activities, Structural Feature Actions, Variable Actions

Owned Association Ends

+ /**context** : Classifier [0..1] {readOnly}

The classifier that owns the behavior of which this action is a part.

+ /**input** : InputPin [0..*] {ordered, readOnly, union, subsets ownedElement}

The ordered set of input pins connected to the Action. These are among the total set of inputs.

+ /**output** : OutputPin [0..*] {ordered, readOnly, union, subsets ownedElement}

The ordered set of output pins connected to the Action. The action places its results onto pins in this set.

Package UML::Actions::BasicActions

Class CallAction

CallAction is an abstract class for actions that invoke behavior and receive return values.

Generalizations:

InvocationAction

Specializations:

CallBehaviorAction, CallOperationAction, StartObjectBehaviorAction

Attributes

+ **isSynchronous** : Boolean [1..1] = true

If true, the call is synchronous and the caller waits for completion of the invoked behavior. If false, the call is asynchronous and the caller proceeds immediately and does not expect a return values.

Owned Association Ends

+ **result** : OutputPin [0..*] {ordered, subsets output}

A list of output pins where the results of performing the invocation are placed.

Constraints

number_and_order

The number and order of argument pins must be the same as the number and order of parameters of the invoked behavior or behavioral feature. Pins are matched to parameters by order.

expression (OCL): true

synchronous_call

Only synchronous call actions can have result pins.

expression (OCL): true

type_ordering_multiplicity

The type, ordering, and multiplicity of an argument pin must be the same as the corresponding parameter of the behavior or behavioral feature.

expression (OCL): true

Package UML::Actions::BasicActions

Class CallBehaviorAction

A call behavior action is a call action that invokes a behavior directly rather than invoking a behavioral feature that, in turn, results in the invocation of that behavior. The argument values of the action are available to the execution of the invoked behavior. For synchronous calls the execution of the call behavior action waits until the execution of the invoked behavior completes and a result is returned on its output pin. The action completes immediately without a result, if the call is asynchronous. In particular, the invoked behavior may be an activity.

Generalizations:

CallAction

Owned Association Ends

+ **behavior** : Behavior [1..1]

The invoked behavior. It must be capable of accepting and returning control.

Constraints

argument_pin_equal_parameter

The number of argument pins and the number of parameters of the behavior of type in and in-out must be equal.

expression (OCL): true

result_pin_equal_parameter

The number of result pins and the number of parameters of the behavior of type return, out, and in-out must be equal.

expression (OCL): true

type_ordering_multiplicity

The type, ordering, and multiplicity of an argument or result pin is derived from the corresponding parameter of the behavior.

expression (OCL): true

Package UML::Actions::BasicActions

Class CallOperationAction

A call operation action is an action that transmits an operation call request to the target object, where it may cause the invocation of associated behavior. The argument values of the action are available to the execution of the invoked behavior. If the action is marked synchronous, the execution of the call operation action waits until the execution of the invoked behavior completes and a reply transmission is returned to the caller; otherwise execution of the action is complete when the invocation of the operation is established and the execution of the invoked operation proceeds concurrently with the execution of the calling behavior. Any values returned as part of the reply transmission are put on the result output pins of the call operation action. Upon receipt of the reply transmission, execution of the call operation action is complete.

Generalizations:

CallAction

Owned Association Ends

+ **operation** : Operation [1..1]

The operation to be invoked by the action execution.

+ **target** : InputPin [1..1] {subsets input}

The target object to which the request is sent. The classifier of the target object is used to dynamically determine a behavior to invoke. This object constitutes the context of the execution of the operation.

Constraints

argument_pin_equal_parameter

The number of argument pins and the number of owned parameters of the operation of type in and in-out must be equal.

expression (OCL): true

result_pin_equal_parameter

The number of result pins and the number of owned parameters of the operation of type return, out, and in-out must be equal.

expression (OCL): true

type_ordering_multiplicity

The type, ordering, and multiplicity of an argument or result pin is derived from the corresponding owned parameter of the operation.

expression (OCL): true

Package UML::Actions::BasicActions

Class CallOperationAction

type_target_pin

The type of the target pin must be the same as the type that owns the operation.

expression (OCL): true

Package UML::Actions::BasicActions

Class InputPin

An input pin is a pin that holds input values to be consumed by an action.

Generalizations:

Pin

Specializations:

ActionInputPin, ValuePin

Found in Diagrams:

Basic Actions, Structural Feature Actions, Variable Actions

Package UML::Actions::BasicActions

Class InvocationAction

InvocationAction is an abstract class for the various actions that invoke behavior.

Generalizations:

Action

Specializations:

BroadcastSignalAction, CallAction, SendObjectAction, SendSignalAction

Owned Association Ends

+ **argument** : InputPin [0..*] {ordered, subsets input}

Specification of the ordered set of argument values that appears during execution.

Package UML::Actions::BasicActions

Class MultiplicityElement

Operations

+ **compatibleWith** (other : MultiplicityElement) : Boolean [1..1] {query}

The operation compatibleWith takes another multiplicity as input. It checks if one multiplicity is compatible with another.

body (OCL): result = Integer.allInstances()->forAll(i : Integer | self.includesCardinality(i) implies other.includesCardinality(i))

+ **is** (lowerbound : Integer, upperbound : Integer) : Boolean [1..1] {query}

The operation is determines if the upper and lower bound of the ranges are the ones given.

body (OCL): result = (lowerbound = self.lowerbound and upperbound = self.upperbound)

Package UML::Actions::BasicActions

Class OpaqueAction

An action with implementation-specific semantics.

Generalizations:

Action

Found in Diagrams:

Basic Actions

Attributes

+ **body** : String [0..*] {ordered}

Specifies the action in one or more languages.

+ **language** : String [0..*] {ordered}

Languages the body strings use, in the same order as the body strings

Owned Association Ends

+ **inputValue** : InputPin [0..*] {subsets input}

Provides input to the action.

+ **outputValue** : OutputPin [0..*] {subsets output}

Takes output from the action.

Package UML::Actions::BasicActions

Class OutputPin

An output pin is a pin that holds output values produced by an action.

Generalizations:

Pin

Found in Diagrams:

Basic Actions, Structural Feature Actions, Variable Actions

Package UML::Actions::BasicActions

Class Pin

A pin is a typed element and multiplicity element that provides values to actions and accept result values from them.

Generalizations:

MultiplicityElement, TypedElement

Specializations:

InputPin, OutputPin

Package UML::Actions::BasicActions

Class SendSignalAction

A send signal action is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity. The argument values are available to the execution of associated behaviors. The requestor continues execution immediately. Any reply message is ignored and is not transmitted to the requestor. If the input is already a signal instance, use a send object action.

Generalizations:

InvocationAction

Owned Association Ends

+ **signal** : Signal [1..1]

The type of signal transmitted to the target object.

+ **target** : InputPin [1..1] {subsets input}

The target object to which the signal is sent.

Constraints

number_order

The number and order of argument pins must be the same as the number and order of attributes in the signal.

expression (OCL): true

type_ordering_multiplicity

The type, ordering, and multiplicity of an argument pin must be the same as the corresponding attribute of the signal.

expression (OCL): true

Package UML::Actions::BasicActions

Class ValuePin

A value pin is an input pin that provides a value by evaluating a value specification.

Generalizations:

InputPin

Owned Association Ends

+ **value** : ValueSpecification [1..1]

Value that the pin will provide.

Constraints

compatible_type

The type of value specification must be compatible with the type of the value pin.

expression (OCL): true

Package UML::Actions::BasicActions

Association A_argument_invocationAction

Member Ends:

argument, invocationAction

Owned Association Ends

+ **invocationAction** : InvocationAction [0..1]

Package UML::Actions::BasicActions

Association A_behavior_callBehaviorAction

Member Ends:

behavior, callBehaviorAction

Owned Association Ends

+ **callBehaviorAction** : CallBehaviorAction [0..*]

Package UML::Actions::BasicActions

Association A_context_action

Member Ends:

context, action

Found in Diagrams:

Basic Actions

Owned Association Ends

+ **action** : Action [0..*]

Package UML::Actions::BasicActions

Association A_inputValue_opaqueAction

Member Ends:

inputValue, opaqueAction

Found in Diagrams:

Basic Actions

Owned Association Ends

+ **opaqueAction** : OpaqueAction [0..1]

Package UML::Actions::BasicActions

Association A_input_action

Member Ends:

input, action

Owned Association Ends

+ **action** : Action [0..1]

Package UML::Actions::BasicActions

Association A_operation_callOperationAction

Member Ends:

operation, callOperationAction

Owned Association Ends

+ **callOperationAction** : CallOperationAction [0..*]

Package UML::Actions::BasicActions

Association A_outputValue_opaqueAction

Member Ends:

outputValue, opaqueAction

Found in Diagrams:

Basic Actions

Owned Association Ends

+ **opaqueAction** : OpaqueAction [0..1]

Package UML::Actions::BasicActions

Association A_output_action

Member Ends:

output, action

Owned Association Ends

+ **action** : Action [0..1]

Package UML::Actions::BasicActions

Association A_result_callAction

Member Ends:

result, callAction

Owned Association Ends

+ **callAction** : CallAction [0..1]

Package UML::Actions::BasicActions

Association A_signal_sendSignalAction

Member Ends:

signal, sendSignalAction

Owned Association Ends

+ **sendSignalAction** : SendSignalAction [0..*]

Package UML::Actions::BasicActions

Association A_target_callOperationAction

Member Ends:

target, callOperationAction

Owned Association Ends

+ **callOperationAction** : CallOperationAction [0..1]

Package UML::Actions::BasicActions

Association A_target_sendSignalAction

Member Ends:

target, sendSignalAction

Owned Association Ends

+ **sendSignalAction** : SendSignalAction [0..1]

Package UML::Actions::BasicActions

Association A_value_valuePin

Member Ends:

value, valuePin

Owned Association Ends

+ **valuePin** : ValuePin [0..1]

Package UML::Actions::CompleteActions

Nesting Package:

Actions

Imported Packages:

AssociationClasses, BasicBehaviors, BehaviorStateMachines, Kernel

Merged Packages:

IntermediateActions, StructuredActions

Class Summary
AcceptCallAction
AcceptEventAction
CreateLinkObjectAction
LinkEndData
QualifierValue
ReadExtentAction
ReadIsClassifiedObjectAction
ReadLinkObjectEndAction
ReadLinkObjectEndQualifierAction
ReclassifyObjectAction
ReduceAction
ReplyAction
StartClassifierBehaviorAction
StartObjectBehaviorAction
UnmarshallAction

Association Summary
A_classifier_readExtentAction
A_classifier_readIsClassifiedObjectAction
A_collection_reduceAction
A_end_readLinkObjectEndAction
A_newClassifier_reclassifyObjectAction
A_object_readIsClassifiedObjectAction
A_object_readLinkObjectEndAction
A_object_readLinkObjectEndQualifierAction
A_object_reclassifyObjectAction
A_object_startClassifierBehaviorAction

Package UML::Actions::CompleteActions

A_object_startObjectBehaviorAction
A_object_unmarshallAction
A_oldClassifier_reclassifyObjectAction
A_qualifier_linkEndData
A_qualifier_qualifierValue
A_qualifier_readLinkObjectEndQualifierAction
A_reducer_reduceAction
A_replyToCall_replyAction
A_replyValue_replyAction
A_result_acceptEventAction
A_result_createLinkObjectAction
A_result_readExtentAction
A_result_readIsClassifiedObjectAction
A_result_readLinkObjectEndAction
A_result_readLinkObjectEndQualifierAction
A_result_reduceAction
A_result_unmarshallAction
A_returnInformation_acceptCallAction
A_returnInformation_replyAction
A_trigger_acceptEventAction
A_unmarshallType_unmarshallAction
A_value_qualifierValue

Package UML::Actions::CompleteActions

Class AcceptCallAction

An accept call action is an accept event action representing the receipt of a synchronous call request. In addition to the normal operation parameters, the action produces an output that is needed later to supply the information to the reply action necessary to return control to the caller. This action is for synchronous calls. If it is used to handle an asynchronous call, execution of the subsequent reply action will complete immediately with no effects.

Generalizations:

AcceptEventAction

Owned Association Ends

+ **returnInformation** : OutputPin [1..1] {subsets output}

Pin where a value is placed containing sufficient information to perform a subsequent reply and return control to the caller. The contents of this value are opaque. It can be passed and copied but it cannot be manipulated by the model.

Constraints

result_pins

The result pins must match the in and inout parameters of the operation specified by the trigger event in number, type, and order.

expression (OCL): true

trigger_call_event

The trigger event must be a CallEvent.

expression (OCL): trigger.event.oclIsKindOf(CallEvent)

unmarshall

isUnmrashall must be true for an AcceptCallAction.

expression (OCL): isUnmarshall = true

Package UML::Actions::CompleteActions

Class AcceptEventAction

A accept event action is an action that waits for the occurrence of an event meeting specified conditions.

Generalizations:

Action

Specializations:

AcceptCallAction

Attributes

+ **isUnmarshall** : Boolean [1..1] = false

Indicates whether there is a single output pin for the event, or multiple output pins for attributes of the event.

Owned Association Ends

+ **result** : OutputPin [0..*] {subsets output}

Pins holding the received event objects or their attributes. Event objects may be copied in transmission, so identity might not be preserved.

+ **trigger** : Trigger [1..*]

The type of events accepted by the action, as specified by triggers. For triggers with signal events, a signal of the specified type or any subtype of the specified signal type is accepted.

Constraints

no_input_pins

AcceptEventActions may have no input pins.

expression (OCL): true

no_output_pins

There are no output pins if the trigger events are only ChangeEvents, or if they are only CallEvents when this action is an instance of AcceptEventAction and not an instance of a descendant of AcceptEventAction (such as AcceptCallAction).

expression (OCL): true

trigger_events

If the trigger events are all TimeEvents, there is exactly one output pin.

Package UML::Actions::CompleteActions

Class AcceptEventAction

expression (OCL): true

unmarshall_signal_events

If isUnmarshall is true, there must be exactly one trigger for events of type SignalEvent. The number of result output pins must be the same as the number of attributes of the signal. The type and ordering of each result output pin must be the same as the corresponding attribute of the signal. The multiplicity of each result output pin must be compatible with the multiplicity of the corresponding attribute.

expression (OCL): true

Package UML::Actions::CompleteActions

Class CreateLinkObjectAction

A create link object action creates a link object.

Generalizations:

CreateLinkAction

Owned Association Ends

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the result is put.

Constraints

association_class

The association must be an association class.

expression (OCL): self.association().oclIsKindOf(Class)

multiplicity

The multiplicity of the output pin is 1..1.

expression (OCL): self.result.multiplicity.is(1,1)

type_of_result

The type of the result pin must be the same as the association of the action.

expression (OCL): self.result.type = self.association()

Package UML::Actions::CompleteActions

Class LinkEndData

Owned Association Ends

+ **qualifier** : QualifierValue [0..*]

List of qualifier values

Constraints

end_object_input_pin

The end object input pin is not also a qualifier value input pin.

expression (OCL): self.value->excludesAll(self.qualifier.value)

qualifiers

The qualifiers include all and only the qualifiers of the association end.

expression (OCL): self.qualifier->collect(qualifier) = self.end.qualifier

Package UML::Actions::CompleteActions

Class QualifierValue

A qualifier value is not an action. It is an element that identifies links. It gives a single qualifier within a link end data specification.

Generalizations:

Element

Owned Association Ends

+ **qualifier** : Property [1..1]

Attribute representing the qualifier for which the value is to be specified.

+ **value** : InputPin [1..1]

Input pin from which the specified value for the qualifier is taken.

Constraints

multiplicity_of_qualifier

The multiplicity of the qualifier value input pin is "1..1".

expression (OCL): self.value.multiplicity.is(1,1)

qualifier_attribute

The qualifier attribute must be a qualifier of the association end of the link-end data.

expression (OCL): self.LinkEndData.end->collect(qualifier)->includes(self.qualifier)

type_of_qualifier

The type of the qualifier value input pin is the same as the type of the qualifier attribute.

expression (OCL): self.value.type = self.qualifier.type

Package UML::Actions::CompleteActions

Class ReadExtentAction

A read extent action is an action that retrieves the current instances of a classifier.

Generalizations:

Action

Owned Association Ends

+ **classifier** : Classifier [1..1]

The classifier whose instances are to be retrieved.

+ **result** : OutputPin [1..1] {subsets output}

The runtime instances of the classifier.

Constraints

multiplicity_of_result

The multiplicity of the result output pin is 0..*.

expression (OCL): self.result.multiplicity.is(0,#null)

type_is_classifier

The type of the result output pin is the classifier.

expression (OCL): true

Package UML::Actions::CompleteActions

Class ReadIsClassifiedObjectAction

A read is classified object action is an action that determines whether a runtime object is classified by a given classifier.

Generalizations:

Action

Attributes

+ **isDirect** : Boolean [1..1] = false

Indicates whether the classifier must directly classify the input object.

Owned Association Ends

+ **classifier** : Classifier [1..1]

The classifier against which the classification of the input object is tested.

+ **object** : InputPin [1..1] {subsets input}

Holds the object whose classification is to be tested.

+ **result** : OutputPin [1..1] {subsets output}

After termination of the action, will hold the result of the test.

Constraints

boolean_result

The type of the output pin is Boolean

expression (OCL): self.result.type = Boolean

multiplicity_of_input

The multiplicity of the input pin is 1..1.

expression (OCL): self.object.multiplicity.is(1,1)

multiplicity_of_output

The multiplicity of the output pin is 1..1.

expression (OCL): self.result.multiplicity.is(1,1)

no_type

Package UML::Actions::CompleteActions

Class ReadIsClassifiedObjectAction

The input pin has no type.

expression (OCL): self.object.type->isEmpty()

Package UML::Actions::CompleteActions

Class ReadLinkObjectEndAction

A read link object end action is an action that retrieves an end object from a link object.

Generalizations:

Action

Owned Association Ends

+ **end** : Property [1..1]

Link end to be read.

+ **object** : InputPin [1..1] {subsets input}

Gives the input pin from which the link object is obtained.

+ **result** : OutputPin [1..1] {subsets output}

Pin where the result value is placed.

Constraints

association_of_association

The association of the association end must be an association class.

expression (OCL): self.end.Association.oclIsKindOf(AssociationClass)

ends_of_association

The ends of the association must not be static.

expression (OCL): self.end.association.memberEnd->forall(e | not e.isStatic)

multiplicity_of_object

The multiplicity of the object input pin is 1..1.

expression (OCL): self.object.multiplicity.is(1,1)

multiplicity_of_result

The multiplicity of the result output pin is 1..1.

expression (OCL): self.result.multiplicity.is(1,1)

property

The property must be an association end.

expression (OCL): self.end.association.notEmpty()

Package UML::Actions::CompleteActions

Class ReadLinkObjectEndAction

type_of_object

The type of the object input pin is the association class that owns the association end.

expression (OCL): self.object.type = self.end.association

type_of_result

The type of the result output pin is the same as the type of the association end.

expression (OCL): self.result.type = self.end.type

Package UML::Actions::CompleteActions

Class ReadLinkObjectEndQualifierAction

A read link object end qualifier action is an action that retrieves a qualifier end value from a link object.

Generalizations:

Action

Owned Association Ends

+ **object** : InputPin [1..1] {subsets input}

Gives the input pin from which the link object is obtained.

+ **qualifier** : Property [1..1]

The attribute representing the qualifier to be read.

+ **result** : OutputPin [1..1] {subsets output}

Pin where the result value is placed.

Constraints

association_of_association

The association of the association end of the qualifier attribute must be an association class.

expression (OCL): self.qualifier.associationEnd.association.oclIsKindOf(AssociationClass)

ends_of_association

The ends of the association must not be static.

expression (OCL): self.qualifier.associationEnd.association.memberEnd->forall(e | not e.isStatic)

multiplicity_of_object

The multiplicity of the object input pin is 1..1.

expression (OCL): self.object.multiplicity.is(1,1)

multiplicity_of_qualifier

The multiplicity of the qualifier attribute is 1..1.

expression (OCL): self.qualifier.multiplicity.is(1,1)

multiplicity_of_result

The multiplicity of the result output pin is 1..1.

expression (OCL): self.result.multiplicity.is(1,1)

Package UML::Actions::CompleteActions

Class ReadLinkObjectEndQualifierAction

qualifier_attribute

The qualifier attribute must be a qualifier attribute of an association end.

expression (OCL): self.qualifier.associationEnd->size() = 1

same_type

The type of the result output pin is the same as the type of the qualifier attribute.

expression (OCL): self.result.type = self.qualifier.type

type_of_object

The type of the object input pin is the association class that owns the association end that has the given qualifier attribute.

expression (OCL): self.object.type = self.qualifier.associationEnd.association

Package UML::Actions::CompleteActions

Class ReclassifyObjectAction

A reclassify object action is an action that changes which classifiers classify an object.

Generalizations:

Action

Attributes

+ **isReplaceAll** : Boolean [1..1] = false

Specifies whether existing classifiers should be removed before adding the new classifiers.

Owned Association Ends

+ **newClassifier** : Classifier [0..*]

A set of classifiers to be added to the classifiers of the object.

+ **object** : InputPin [1..1] {subsets input}

Holds the object to be reclassified.

+ **oldClassifier** : Classifier [0..*]

A set of classifiers to be removed from the classifiers of the object.

Constraints

classifier_not_abstract

None of the new classifiers may be abstract.

expression (OCL): not self.newClassifier->exists(isAbstract = true)

input_pin

The input pin has no type.

expression (OCL): self.argument.type->size() = 0

multiplicity

The multiplicity of the input pin is 1..1.

expression (OCL): self.argument.multiplicity.is(1,1)

Package UML::Actions::CompleteActions

Class ReduceAction

A reduce action is an action that reduces a collection to a single value by combining the elements of the collection.

Generalizations:

Action

Attributes

+ **isOrdered** : Boolean [1..1] = false

Tells whether the order of the input collection should determine the order in which the behavior is applied to its elements.

Owned Association Ends

+ **collection** : InputPin [1..1] {subsets input}

The collection to be reduced.

+ **reducer** : Behavior [1..1]

Behavior that is applied to two elements of the input collection to produce a value that is the same type as elements of the collection.

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the result is put.

Constraints

input_type_is_collection

The type of the input must be a collection.

expression (OCL): true

output_types_are_compatible

The type of the output must be compatible with the type of the output of the reducer behavior.

expression (OCL): true

reducer_inputs_output

The reducer behavior must have two input parameters and one output parameter, of types compatible with the types of elements of the input collection.

Package UML::Actions::CompleteActions

Class ReduceAction

expression (OCL): true

Package UML::Actions::CompleteActions

Class ReplyAction

A reply action is an action that accepts a set of return values and a value containing return information produced by a previous accept call action. The reply action returns the values to the caller of the previous call, completing execution of the call.

Generalizations:

Action

Owned Association Ends

+ **replyToCall** : Trigger [1..1]

The trigger specifying the operation whose call is being replied to.

+ **replyValue** : InputPin [0..*] {subsets input}

A list of pins containing the reply values of the operation. These values are returned to the caller.

+ **returnInformation** : InputPin [1..1] {subsets input}

A pin containing the return information value produced by an earlier AcceptCallAction.

Constraints

event_on_reply_to_call_trigger

The event on replyToCall trigger must be a CallEvent replyToCallEvent.oclIsKindOf(CallEvent)

expression (OCL): replyToCallEvent.oclIsKindOf(CallEvent)

pins_match_parameter

The reply value pins must match the return, out, and inout parameters of the operation on the event on the trigger in number, type, and order.

expression (OCL): true

Package UML::Actions::CompleteActions

Class StartClassifierBehaviorAction

A start classifier behavior action is an action that starts the classifier behavior of the input.

Generalizations:

Action

Owned Association Ends

+ **object** : InputPin [1..1] {subsets input}

Holds the object on which to start the owned behavior.

Constraints

multiplicity

The multiplicity of the input pin is 1..1

expression (OCL): true

type_has_classifier

If the input pin has a type, then the type must have a classifier behavior.

expression (OCL): true

Package UML::Actions::CompleteActions

Class StartObjectBehaviorAction

StartObjectBehaviorAction is an action that starts the execution either of a directly instantiated behavior or of the classifier behavior of an object. Argument values may be supplied for the input parameters of the behavior. If the behavior is invoked synchronously, then output values may be obtained for output parameters.

Generalizations:

CallAction

Owned Association Ends

+ **object** : InputPin [1..1] {subsets input}

Holds the object which is either a behavior to be started or has a classifier behavior to be started.

Constraints

multiplicity_of_object

The multiplicity of the object input pin must be [1..1].

expression (OCL): true

number_order_arguments

The number and order of the argument pins must be the same as the number and order of the in and in-out parameters of the invoked behavior. Pins are matched to parameters by order.

expression (OCL): true

number_order_results

The number and order of result pins must be the same as the number and order of the in-out, out and return parameters of the invoked behavior. Pins are matched to parameters by order.

expression (OCL): true

type_of_object

The type of the object input pin must be either a Behavior or a BehavioredClassifier with a classifier behavior.

expression (OCL): true

type_ordering_multiplicity_match

The type, ordering, and multiplicity of an argument or result pin must be the same as the corresponding parameter of the behavior.

expression (OCL): true

Package UML::Actions::CompleteActions

Class UnmarshallAction

An unmarshall action is an action that breaks an object of a known type into outputs each of which is equal to a value from a structural feature of the object.

Generalizations:

Action

Owned Association Ends

+ **object** : InputPin [1..1] {subsets input}

The object to be unmarshalled.

+ **result** : OutputPin [1..*] {subsets output}

The values of the structural features of the input object.

+ **unmarshallType** : Classifier [1..1]

The type of the object to be unmarshalled.

Constraints

multiplicity_of_object

The multiplicity of the object input pin is 1..1

expression (OCL): true

multiplicity_of_result

The multiplicity of each result output pin must be compatible with the multiplicity of the corresponding structural features of the unmarshall classifier.

expression (OCL): true

number_of_result

The number of result output pins must be the same as the number of structural features of the unmarshall classifier.

expression (OCL): true

same_type

The type of the object input pin must be the same as the unmarshall classifier.

expression (OCL): true

structural_feature

Package UML::Actions::CompleteActions

Class UnmarshallAction

The unmarshall classifier must have at least one structural feature.

expression (OCL): true

type_and_ordering

The type and ordering of each result output pin must be the same as the corresponding structural feature of the unmarshall classifier.

expression (OCL): true

unmarshallType_is_classifier

unmarshallType must be a Classifier with ordered attributes

expression (OCL): true

Package UML::Actions::CompleteActions

Association A_classifier_readExtentAction

Member Ends:

classifier, readExtentAction

Owned Association Ends

+ **readExtentAction** : ReadExtentAction [0..1]

Package UML::Actions::CompleteActions

Association A_classifier_readIsClassifiedObjectAction

Member Ends:

classifier, readIsClassifiedObjectAction

Owned Association Ends

+ **readIsClassifiedObjectAction** : ReadIsClassifiedObjectAction [0..*]

Package UML::Actions::CompleteActions

Association A_collection_reduceAction

Member Ends:

collection, reduceAction

Owned Association Ends

+ **reduceAction** : ReduceAction [0..1]

Package UML::Actions::CompleteActions

Association A_end_readLinkObjectEndAction

Member Ends:

end, readLinkObjectEndAction

Owned Association Ends

+ **readLinkObjectEndAction** : ReadLinkObjectEndAction [0..1]

Package UML::Actions::CompleteActions

Association A_newClassifier_reclassifyObjectAction

Member Ends:

newClassifier, reclassifyObjectAction

Owned Association Ends

+ **reclassifyObjectAction** : ReclassifyObjectAction [0..*]

Package UML::Actions::CompleteActions

Association A_object_readIsClassifiedObjectAction

Member Ends:

object, readIsClassifiedObjectAction

Owned Association Ends

+ **readIsClassifiedObjectAction** : ReadIsClassifiedObjectAction [0..1]

Package UML::Actions::CompleteActions

Association A_object_readLinkObjectEndAction

Member Ends:

object, readLinkObjectEndAction

Owned Association Ends

+ **readLinkObjectEndAction** : ReadLinkObjectEndAction [0..1]

Package UML::Actions::CompleteActions

Association A_object_readLinkObjectEndQualifierAction

Member Ends:

object, readLinkObjectEndQualifierAction

Owned Association Ends

+ **readLinkObjectEndQualifierAction** : ReadLinkObjectEndQualifierAction [0..1]

Package UML::Actions::CompleteActions

Association A_object_reclassifyObjectAction

Member Ends:

object, reclassifyObjectAction

Owned Association Ends

+ **reclassifyObjectAction** : ReclassifyObjectAction [0..1]

Package UML::Actions::CompleteActions

Association A_object_startClassifierBehaviorAction

Member Ends:

object, startClassifierBehaviorAction

Owned Association Ends

+ **startClassifierBehaviorAction** : StartClassifierBehaviorAction [0..1]

Package UML::Actions::CompleteActions

Association A_object_startObjectBehaviorAction

Member Ends:

object, startObjectBehaviorAction

Owned Association Ends

+ **startObjectBehaviorAction** : StartObjectBehaviorAction [0..1]

Package UML::Actions::CompleteActions

Association A_object_unmarshallAction

Member Ends:

object, unmarshallAction

Owned Association Ends

+ **unmarshallAction** : UnmarshallAction [0..1]

Package UML::Actions::CompleteActions

Association A_oldClassifier_reclassifyObjectAction

Member Ends:

oldClassifier, reclassifyObjectAction

Owned Association Ends

+ **reclassifyObjectAction** : ReclassifyObjectAction [0..*]

Package UML::Actions::CompleteActions

Association A_qualifier_linkEndData

Member Ends:

qualifier, linkEndData

Owned Association Ends

+ **linkEndData** : LinkEndData [1..1]

Package UML::Actions::CompleteActions

Association A_qualifier_qualifierValue

Member Ends:

qualifier, qualifierValue

Owned Association Ends

+ **qualifierValue** : QualifierValue [0..*]

Package UML::Actions::CompleteActions

Association A_qualifier_readLinkObjectEndQualifierAction

Member Ends:

qualifier, readLinkObjectEndQualifierAction

Owned Association Ends

+ **readLinkObjectEndQualifierAction** : ReadLinkObjectEndQualifierAction [0..1]

Package UML::Actions::CompleteActions

Association A_reducer_reduceAction

Member Ends:

reducer, reduceAction

Owned Association Ends

+ **reduceAction** : ReduceAction [0..*]

Package UML::Actions::CompleteActions

Association A_replyToCall_replyAction

Member Ends:

replyToCall, replyAction

Owned Association Ends

+ **replyAction** : ReplyAction [0..1]

Package UML::Actions::CompleteActions

Association A_replyValue_replyAction

Member Ends:

replyValue, replyAction

Owned Association Ends

+ **replyAction** : ReplyAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_acceptEventAction

Member Ends:

result, acceptEventAction

Owned Association Ends

+ **acceptEventAction** : AcceptEventAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_createLinkObjectAction

Member Ends:

result, createLinkObjectAction

Owned Association Ends

+ **createLinkObjectAction** : CreateLinkObjectAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_readExtentAction

Member Ends:

result, readExtentAction

Owned Association Ends

+ **readExtentAction** : ReadExtentAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_readIsClassifiedObjectAction

Member Ends:

result, readIsClassifiedObjectAction

Owned Association Ends

+ **readIsClassifiedObjectAction** : ReadIsClassifiedObjectAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_readLinkObjectEndAction

Member Ends:

result, readLinkObjectEndAction

Owned Association Ends

+ **readLinkObjectEndAction** : ReadLinkObjectEndAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_readLinkObjectEndQualifierAction

Member Ends:

result, readLinkObjectEndQualifierAction

Owned Association Ends

+ **readLinkObjectEndQualifierAction** : ReadLinkObjectEndQualifierAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_reduceAction

Member Ends:

result, reduceAction

Owned Association Ends

+ **reduceAction** : ReduceAction [0..1]

Package UML::Actions::CompleteActions

Association A_result_unmarshallAction

Member Ends:

result, unmarshallAction

Owned Association Ends

+ **unmarshallAction** : UnmarshallAction [0..1]

Package UML::Actions::CompleteActions

Association A_returnInformation_acceptCallAction

Member Ends:

returnInformation, acceptCallAction

Owned Association Ends

+ **acceptCallAction** : AcceptCallAction [0..1]

Package UML::Actions::CompleteActions

Association A_returnInformation_replyAction

Member Ends:

returnInformation, replyAction

Owned Association Ends

+ **replyAction** : ReplyAction [0..1]

Package UML::Actions::CompleteActions

Association A_trigger_acceptEventAction

Member Ends:

trigger, acceptEventAction

Owned Association Ends

+ **acceptEventAction** : AcceptEventAction [0..1]

Package UML::Actions::CompleteActions

Association A_unmarshallType_unmarshallAction

Member Ends:

unmarshallType, unmarshallAction

Owned Association Ends

+ **unmarshallAction** : UnmarshallAction [0..*]

Package UML::Actions::CompleteActions

Association A_value_qualifierValue

Member Ends:

value, qualifierValue

Owned Association Ends

+ **qualifierValue** : QualifierValue [0..1]

Package UML::Actions::IntermediateActions

Nesting Package:

Actions

Imported Packages:

BasicActions, Kernel

Merged Packages:

BasicBehaviors

Diagram Summary

Structural Feature Actions

Class Summary

AddStructuralFeatureValueAction
BroadcastSignalAction
ClearAssociationAction
ClearStructuralFeatureAction
CreateLinkAction
CreateObjectAction
DestroyLinkAction
DestroyObjectAction
LinkAction
LinkEndCreationData
LinkEndData
LinkEndDestructionData
ReadLinkAction
ReadSelfAction
ReadStructuralFeatureAction
RemoveStructuralFeatureValueAction
SendObjectAction
StructuralFeatureAction
TestIdentityAction
ValueSpecificationAction
WriteLinkAction
WriteStructuralFeatureAction

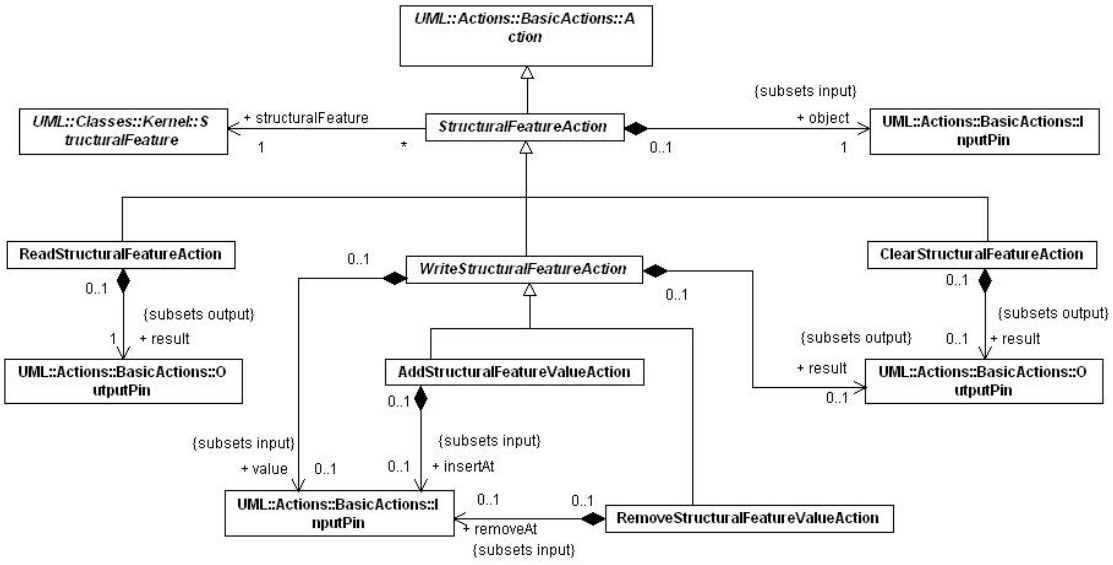
Association Summary

Package UML::Actions::IntermediateActions

A_association_clearAssociationAction
A_classifier_createObjectAction
A_destroyAt_linkEndDestructionData
A_endData_createLinkAction
A_endData_destroyLinkAction
A_endData_linkAction
A_end_linkEndData
A_first_testIdentityAction
A_inputValue_linkAction
A_insertAt_addStructuralFeatureValueAction
A_insertAt_linkEndCreationData
A_object_clearAssociationAction
A_object_structuralFeatureAction
A_removeAt_removeStructuralFeatureValueAction
A_request_sendObjectAction
A_result_clearStructuralFeatureAction
A_result_createObjectAction
A_result_readLinkAction
A_result_readSelfAction
A_result_readStructuralFeatureAction
A_result_testIdentityAction
A_result_valueSpecificationAction
A_result_writeStructuralFeatureAction
A_second_testIdentityAction
A_signal_broadcastSignalAction
A_structuralFeature_structuralFeatureAction
A_target_destroyObjectAction
A_target_sendObjectAction
A_value_linkEndData
A_value_valueSpecificationAction
A_value_writeStructuralFeatureAction

Package UML::Actions::IntermediateActions

Diagram Structural Feature Actions



Classifiers Local to Package:

AddStructuralFeatureValueAction, ClearStructuralFeatureAction, ReadStructuralFeatureAction, RemoveStructuralFeatureValueAction, StructuralFeatureAction, WriteStructuralFeatureAction

Classifiers External to Package:

Action, InputPin, OutputPin, StructuralFeature

Package UML::Actions::IntermediateActions

Class AddStructuralFeatureValueAction

An add structural feature value action is a write structural feature action for adding values to a structural feature.

Generalizations:

WriteStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Attributes

+ **isReplaceAll** : Boolean [1..1] = false

Specifies whether existing values of the structural feature of the object should be removed before adding the new value.

Owned Association Ends

+ **insertAt** : InputPin [0..1] {subsets input}

Gives the position at which to insert a new value or move an existing value in ordered structural features. The type of the pin is UnlimitedNatural, but the value cannot be zero. This pin is omitted for unordered structural features.

Constraints

required_value

A value input pin is required.

expression (OCL): self.value -> notEmpty()

unlimited_natural_and_multiplicity

Actions adding a value to ordered structural features must have a single input pin for the insertion point with type UnlimitedNatural and multiplicity of 1..1, otherwise the action has no input pin for the insertion point.

expression (OCL): let insertAtPins : Collection = self.insertAt in if self.structuralFeature.isOrdered = #false then insertAtPins->size() = 0 else let insertAtPin : InputPin= insertAt->asSequence()->first () in insertAtPins->size() = 1 and insertAtPin.type = UnlimitedNatural and insertAtPin.multiplicity. is(1,1)) endif

Package UML::Actions::IntermediateActions

Class BroadcastSignalAction

A broadcast signal action is an action that transmits a signal instance to all the potential target objects in the system, which may cause the firing of a state machine transitions or the execution of associated activities of a target object. The argument values are available to the execution of associated behaviors. The requestor continues execution immediately after the signals are sent out. It does not wait for receipt. Any reply messages are ignored and are not transmitted to the requestor.

Generalizations:

InvocationAction

Owned Association Ends

+ **signal** : Signal [1..1]

The specification of signal object transmitted to the target objects.

Constraints

number_and_order

The number and order of argument pins must be the same as the number and order of attributes in the signal.

expression (OCL): true

type_ordering_multiplicity

The type, ordering, and multiplicity of an argument pin must be the same as the corresponding attribute of the signal.

expression (OCL): true

Package UML::Actions::IntermediateActions

Class ClearAssociationAction

A clear association action is an action that destroys all links of an association in which a particular object participates.

Generalizations:

Action

Owned Association Ends

+ **association** : Association [1..1]

Association to be cleared.

+ **object** : InputPin [1..1] {subsets input}

Gives the input pin from which is obtained the object whose participation in the association is to be cleared.

Constraints

multiplicity

The multiplicity of the input pin is 1..1.

expression (OCL): self.object.multiplicity.is(1,1)

same_type

The type of the input pin must be the same as the type of at least one of the association ends of the association.

expression (OCL): self.association->exists(end.type = self.object.type)

Package UML::Actions::IntermediateActions

Class ClearStructuralFeatureAction

A clear structural feature action is a structural feature action that removes all values of a structural feature.

Generalizations:

StructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **result** : OutputPin [0..1] {subsets output}

Gives the output pin on which the result is put.

Constraints

multiplicity_of_result

The multiplicity of the result output pin must be 1..1.

expression (OCL): result->notEmpty() implies self.result.multiplicity.is(1,1)

type_of_result

The type of the result output pin is the same as the type of the inherited object input pin.

expression (OCL): result->notEmpty() implies self.result.type = self.object.type

Package UML::Actions::IntermediateActions

Class CreateLinkAction

A create link action is a write link action for creating links.

Generalizations:

WriteLinkAction

Specializations:

CreateLinkObjectAction

Owned Association Ends

+ **endData** : LinkEndCreationData [2..*] {redefines endData}

Specifies ends of association and inputs.

Constraints

association_not_abstract

The association cannot be an abstract classifier.

expression (OCL): self.association().isAbstract = #false

Package UML::Actions::IntermediateActions

Class CreateObjectAction

A create object action is an action that creates an object that conforms to a statically specified classifier and puts it on an output pin at runtime.

Generalizations:

Action

Owned Association Ends

+ **classifier** : Classifier [1..1]

Classifier to be instantiated.

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the result is put.

Constraints

classifier_not_abstract

The classifier cannot be abstract.

expression (OCL): not (self.classifier.isAbstract = #true)

classifier_not_association_class

The classifier cannot be an association class

expression (OCL): not self.classifier.oclIsKindOf(AssociationClass)

multiplicity

The multiplicity of the output pin is 1..1.

expression (OCL): self.result.multiplicity.is(1,1)

same_type

The type of the result pin must be the same as the classifier of the action.

expression (OCL): self.result.type = self.classifier

Package UML::Actions::IntermediateActions

Class DestroyLinkAction

A destroy link action is a write link action that destroys links and link objects.

Generalizations:

WriteLinkAction

Owned Association Ends

+ **endData** : LinkEndDestructionData [2..*] {redefines endData}

Specifies ends of association and inputs.

Package UML::Actions::IntermediateActions

Class DestroyObjectAction

A destroy object action is an action that destroys objects.

Generalizations:

Action

Attributes

+ **isDestroyLinks** : Boolean [1..1] = false

Specifies whether links in which the object participates are destroyed along with the object.

+ **isDestroyOwnedObjects** : Boolean [1..1] = false

Specifies whether objects owned by the object are destroyed along with the object.

Owned Association Ends

+ **target** : InputPin [1..1] {subsets input}

The input pin providing the object to be destroyed.

Constraints

multiplicity

The multiplicity of the input pin is 1..1.

expression (OCL): self.target.multiplicity.is(1,1)

no_type

The input pin has no type.

expression (OCL): self.target.type->size() = 0

Package UML::Actions::IntermediateActions

Class LinkAction

LinkAction is an abstract class for all link actions that identify their links by the objects at the ends of the links and by the qualifiers at ends of the links.

Generalizations:

Action

Specializations:

ReadLinkAction, WriteLinkAction

Owned Association Ends

+ **endData** : LinkEndData [2..*]

Data identifying one end of a link by the objects on its ends and qualifiers.

+ **inputValue** : InputPin [1..*] {subsets input}

Pins taking end objects and qualifier values as input.

Operations

+ **association** () : Association [1..1] {query}

The association operates on LinkAction. It returns the association of the action.

body (OCL): result = self.endData->asSequence().first().end.association

Constraints

not_static

The association ends of the link end data must not be static.

expression (OCL): self.endData->forall(end.oclisKindOf(NavigableEnd) implies end.isStatic = # false

same_association

The association ends of the link end data must all be from the same association and include all and only the association ends of that association.

expression (OCL): self.endData->collect(end) = self.association()->collect(connection))

same_pins

The input pins of the action are the same as the pins of the link end data and insertion pins.

expression (OCL): self.input->asSet() = let ledpins : Set = self.endData->collect(value) in if self.

Package UML::Actions::IntermediateActions

Class LinkAction

oclIsKindOf(LinkEndCreationData) then ledpins->union(self.endData.oclAsType (LinkEndCreationData).insertAt) else ledpins

Package UML::Actions::IntermediateActions

Class LinkEndCreationData

A link end creation data is not an action. It is an element that identifies links. It identifies one end of a link to be created by a create link action.

Generalizations:

LinkEndData

Attributes

+ **isReplaceAll** : Boolean [1..1] = false

Specifies whether the existing links emanating from the object on this end should be destroyed before creating a new link.

Owned Association Ends

+ **insertAt** : InputPin [0..1]

Specifies where the new link should be inserted for ordered association ends, or where an existing link should be moved to. The type of the input is UnlimitedNatural, but the input cannot be zero. This pin is omitted for association ends that are not ordered.

Constraints

create_link_action

LinkEndCreationData can only be end data for CreateLinkAction or one of its specializations.

expression (OCL): self.LinkAction.oclIsKindOf(CreateLinkAction)

single_input_pin

Link end creation data for ordered association ends must have a single input pin for the insertion point with type UnlimitedNatural and multiplicity of 1..1, otherwise the action has no input pin for the insertion point.

expression (OCL): let insertAtPins : Collection = self.insertAt in if self.end.ordering = #unordered then insertAtPins->size() = 0 else let insertAtPin : InputPin = insertAts->asSequence()->first() in insertAtPins->size() = 1 and insertAtPin.type = UnlimitedNatural and insertAtPin.multiplicity.is (1,1)) endif

Package UML::Actions::IntermediateActions

Class LinkEndData

A link end data is not an action. It is an element that identifies links. It identifies one end of a link to be read or written by the children of a link action. A link cannot be passed as a runtime value to or from an action. Instead, a link is identified by its end objects and qualifier values, if any. This requires more than one piece of data, namely, the statically-specified end in the user model, the object on the end, and the qualifier values for that end, if any. These pieces are brought together around a link end data. Each association end is identified separately with an instance of the LinkEndData class.

Generalizations:

Element

Specializations:

LinkEndCreationData, LinkEndDestructionData

Owned Association Ends

+ **end** : Property [1..1]

Association end for which this link-end data specifies values.

+ **value** : InputPin [0..1]

Input pin that provides the specified object for the given end. This pin is omitted if the link-end data specifies an 'open' end for reading.

Constraints

multiplicity

The multiplicity of the end object input pin must be 1..1.

expression (OCL): self.value.multiplicity.is(1,1)

property_is_association_end

The property must be an association end.

expression (OCL): self.end.association->size() = 1

same_type

The type of the end object input pin is the same as the type of the association end.

expression (OCL): self.value.type = self.end.type

Package UML::Actions::IntermediateActions

Class LinkEndDestructionData

A link end destruction data is not an action. It is an element that identifies links. It identifies one end of a link to be destroyed by destroy link action.

Generalizations:

LinkEndData

Attributes

+ **isDestroyDuplicates** : Boolean [1..1] = false

Specifies whether to destroy duplicates of the value in nonunique association ends.

Owned Association Ends

+ **destroyAt** : InputPin [0..1]

Specifies the position of an existing link to be destroyed in ordered nonunique association ends. The type of the pin is UnlimitedNatural, but the value cannot be zero or unlimited.

Constraints

destroy_link_action

LinkEndDestructionData can only be end data for DestroyLinkAction or one of its specializations.

expression (OCL): true

unlimited_natural_and_multiplicity

LinkEndDestructionData for ordered nonunique association ends must have a single destroyAt input pin if isDestroyDuplicates is false. It must be of type UnlimitedNatural and have a multiplicity of 1..1. Otherwise, the action has no input pin for the removal position.

expression (OCL): true

Package UML::Actions::IntermediateActions

Class ReadLinkAction

A read link action is a link action that navigates across associations to retrieve objects on one end.

Generalizations:

LinkAction

Owned Association Ends

+ **result** : OutputPin [1..1] {subsets output}

The pin on which are put the objects participating in the association at the end not specified by the inputs.

Constraints

compatible_multiplicity

The multiplicity of the open association end must be compatible with the multiplicity of the result output pin.

expression (OCL): let openend : Property = self.endData->select(ed | ed.value->size() = 0)->asSequence()->first().end in openend.multiplicity.compatibleWith(self.result.multiplicity)

navigable_open_end

The open end must be navigable.

expression (OCL): let openend : Property = self.endData->select(ed | ed.value->size() = 0)->asSequence()->first().end in openend.isNavigable()

one_open_end

Exactly one link-end data specification (the 'open' end) must not have an end object input pin.

expression (OCL): self.endData->select(ed | ed.value->size() = 0)->size() = 1

type_and_ordering

The type and ordering of the result output pin are same as the type and ordering of the open association end.

expression (OCL): let openend : Property = self.endData->select(ed | ed.value->size() = 0)->asSequence()->first().end in self.result.type = openend.type and self.result.ordering = openend.ordering

visibility

Visibility of the open end must allow access to the object performing the action.

expression (OCL): let host : Classifier = self.context in let openend : Property = self.endData->select(ed | ed.value->size() = 0)->asSequence()->first().end in openend.visibility = #public or self.

Package UML::Actions::IntermediateActions

Class ReadLinkAction

endData->exists(oed | not oed.end = openend and (host = oed.end.participant or (openend.visibility = #protected and host.allSupertypes->includes(oed.end.participant))))

Package UML::Actions::IntermediateActions

Class ReadSelfAction

A read self action is an action that retrieves the host object of an action.

Generalizations:

Action

Owned Association Ends

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the hosting object is placed.

Constraints

contained

The action must be contained in an behavior that has a host classifier.

expression (OCL): self.context->size() = 1

multiplicity

The multiplicity of the result output pin is 1..1.

expression (OCL): self.result.multiplicity.is(1,1)

not_static

If the action is contained in an behavior that is acting as the body of a method, then the operation of the method must not be static.

expression (OCL): true

type

The type of the result output pin is the host classifier.

expression (OCL): self.result.type = self.context

Package UML::Actions::IntermediateActions

Class ReadStructuralFeatureAction

A read structural feature action is a structural feature action that retrieves the values of a structural feature.

Generalizations:

StructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the result is put.

Constraints

multiplicity

The multiplicity of the structural feature must be compatible with the multiplicity of the output pin.

expression (OCL): self.structuralFeature.multiplicity.compatibleWith(self.result.multiplicity)

type_and_ordering

The type and ordering of the result output pin are the same as the type and ordering of the structural feature.

expression (OCL): self.result.type = self.structuralFeature.type and self.result.ordering = self.structuralFeature.ordering

Package UML::Actions::IntermediateActions

Class RemoveStructuralFeatureValueAction

A remove structural feature value action is a write structural feature action that removes values from structural features.

Generalizations:

WriteStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Attributes

+ **isRemoveDuplicates** : Boolean [1..1] = false

Specifies whether to remove duplicates of the value in nonunique structural features.

Owned Association Ends

+ **removeAt** : InputPin [0..1] {subsets input}

Specifies the position of an existing value to remove in ordered nonunique structural features. The type of the pin is UnlimitedNatural, but the value cannot be zero or unlimited.

Constraints

non_unique_removal

Actions removing a value from ordered non-unique structural features must have a single removeAt input pin and no value input pin if isRemoveDuplicates is false. The removeAt pin must be of type Unlimited Natural with multiplicity 1..1. Otherwise, the action has a value input pin and no removeAt input pin.

expression (OCL): if not self.structuralFeature.isOrdered or self.structuralFeature.isUnique or isRemoveDuplicates then self.removeAt -> isEmpty() and self.value -> notEmpty() else self.value -> isEmpty() and self.removeAt -> notEmpty() and self.removeAt.type = UnlimitedNatural and self.removeAt.lower = 1 and self.removeAt.upper = 1 endif

Package UML::Actions::IntermediateActions

Class SendObjectAction

A send object action is an action that transmits an object to the target object, where it may invoke behavior such as the firing of state machine transitions or the execution of an activity. The value of the object is available to the execution of invoked behaviors. The requestor continues execution immediately. Any reply message is ignored and is not transmitted to the requestor.

Generalizations:

InvocationAction

Owned Association Ends

+ **request** : InputPin [1..1] {redefines argument}

The request object, which is transmitted to the target object. The object may be copied in transmission, so identity might not be preserved.

+ **target** : InputPin [1..1] {subsets input}

The target object to which the object is sent.

Package UML::Actions::IntermediateActions

Class StructuralFeatureAction

StructuralFeatureAction is an abstract class for all structural feature actions.

Generalizations:

Action

Specializations:

ClearStructuralFeatureAction, ReadStructuralFeatureAction, WriteStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **object** : InputPin [1..1] {subsets input}

Gives the input pin from which the object whose structural feature is to be read or written is obtained.

+ **structuralFeature** : StructuralFeature [1..1]

Structural feature to be read.

Constraints

multiplicity

The multiplicity of the object input pin must be 1..1.

expression (OCL): self.object.lowerBound()=1 and self.object.upperBound()=1

not_static

The structural feature must not be static.

expression (OCL): self.structuralFeature.isStatic = #false

one_featuring_classifier

A structural feature has exactly one featuringClassifier.

expression (OCL): self.structuralFeature.featuringClassifier->size() = 1

same_type

The structural feature must either be owned by the type of the object input pin, or it must be an owned end of a binary association with the type of the opposite end being the type of the object input pin.

expression (OCL): self.structuralFeature.featuringClassifier.oclAsType(Type)->includes(self.

Package UML::Actions::IntermediateActions

Class StructuralFeatureAction

object.type) or self.structuralFeature.oclAsType(Property).opposite.type = self.object.type

visibility

Visibility of structural feature must allow access to the object performing the action.

expression (OCL): let host : Classifier = self.context in self.structuralFeature.visibility = #public or host = self.structuralFeature.featuringClassifier.type or (self.structuralFeature.visibility = #protected and host.allSupertypes ->includes(self.structuralFeature.featuringClassifier.type)))

Package UML::Actions::IntermediateActions

Class TestIdentityAction

A test identity action is an action that tests if two values are identical objects.

Generalizations:

Action

Owned Association Ends

+ **first** : InputPin [1..1] {subsets input}

Gives the pin on which an object is placed.

+ **result** : OutputPin [1..1] {subsets output}

Tells whether the two input objects are identical.

+ **second** : InputPin [1..1] {subsets input}

Gives the pin on which an object is placed.

Constraints

multiplicity

The multiplicity of the input pins is 1..1.

expression (OCL): self.first.multiplicity.is(1,1) and self.second.multiplicity.is(1,1)

no_type

The input pins have no type.

expression (OCL): self.first.type->size() = 0 and self.second.type->size() = 0

result_is_boolean

The type of the result is Boolean.

expression (OCL): self.result.type.oclIsTypeOf(Boolean)

Package UML::Actions::IntermediateActions

Class ValueSpecificationAction

A value specification action is an action that evaluates a value specification.

Generalizations:

Action

Owned Association Ends

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the result is put.

+ **value** : ValueSpecification [1..1]

Value specification to be evaluated.

Constraints

compatible_type

The type of value specification must be compatible with the type of the result pin.

expression (OCL): true

multiplicity

The multiplicity of the result pin is 1..1

expression (OCL): true

Package UML::Actions::IntermediateActions

Class WriteLinkAction

WriteLinkAction is an abstract class for link actions that create and destroy links.

Generalizations:

LinkAction

Specializations:

CreateLinkAction, DestroyLinkAction

Constraints

allow_access

The visibility of at least one end must allow access to the class using the action.

expression (OCL): true

Package UML::Actions::IntermediateActions

Class WriteStructuralFeatureAction

WriteStructuralFeatureAction is an abstract class for structural feature actions that change structural feature values.

Generalizations:

StructuralFeatureAction

Specializations:

AddStructuralFeatureValueAction, RemoveStructuralFeatureValueAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **result** : OutputPin [0..1] {subsets output}

Gives the output pin on which the result is put.

+ **value** : InputPin [0..1] {subsets input}

Value to be added or removed from the structural feature.

Constraints

input_pin

The type input pin is the same as the classifier of the structural feature.

expression (OCL): self.value -> notEmpty() implies self.value.type.oclIsKindOf(Classifier) and self.structuralFeature.featuringClassifier->includes(self.value.type.oclAsType(Classifier))

multiplicity

The multiplicity of the input pin is 1..1.

expression (OCL): self.value.multiplicity.is(1,1)

multiplicity_of_result

The multiplicity of the result output pin must be 1..1.

expression (OCL): result->notEmpty() implies self.result.multiplicity.is(1,1)

type_of_result

The type of the result output pin is the same as the type of the inherited object input pin.

expression (OCL): result->notEmpty() implies self.result.type = self.object.type

Package UML::Actions::IntermediateActions

Association A_association_clearAssociationAction

Member Ends:

association, clearAssociationAction

Owned Association Ends

+ **clearAssociationAction** : ClearAssociationAction [0..1]

Package UML::Actions::IntermediateActions

Association A_classifier_createObjectAction

Member Ends:

classifier, createObjectAction

Owned Association Ends

+ **createObjectAction** : CreateObjectAction [0..*]

Package UML::Actions::IntermediateActions

Association A_destroyAt_linkEndDestructionData

Member Ends:

destroyAt, linkEndDestructionData

Owned Association Ends

+ **linkEndDestructionData** : LinkEndDestructionData [0..1]

Package UML::Actions::IntermediateActions

Association A_endData_createLinkAction

Member Ends:

endData, createLinkAction

Owned Association Ends

+ **createLinkAction** : CreateLinkAction [1..1]

Package UML::Actions::IntermediateActions

Association A_endData_destroyLinkAction

Member Ends:

endData, destroyLinkAction

Owned Association Ends

+ **destroyLinkAction** : DestroyLinkAction [1..1]

Package UML::Actions::IntermediateActions

Association A_endData_linkAction

Member Ends:

endData, linkAction

Owned Association Ends

+ **linkAction** : LinkAction [1..1]

Package UML::Actions::IntermediateActions

Association A_end_linkEndData

Member Ends:

end, linkEndData

Owned Association Ends

+ **linkEndData** : LinkEndData [0..*]

Package UML::Actions::IntermediateActions

Association A_first_testIdentityAction

Member Ends:

first, testIdentityAction

Owned Association Ends

+ **testIdentityAction** : TestIdentityAction [0..1]

Package UML::Actions::IntermediateActions

Association A_inputValue_linkAction

Member Ends:

inputValue, linkAction

Owned Association Ends

+ **linkAction** : LinkAction [0..1]

Package UML::Actions::IntermediateActions

Association A_insertAt_addStructuralFeatureValueAction

Member Ends:

insertAt, addStructuralFeatureValueAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **addStructuralFeatureValueAction** : AddStructuralFeatureValueAction [0..1]

Package UML::Actions::IntermediateActions

Association A_insertAt_linkEndCreationData

Member Ends:

insertAt, linkEndCreationData

Owned Association Ends

+ **linkEndCreationData** : LinkEndCreationData [0..1]

Package UML::Actions::IntermediateActions

Association A_object_clearAssociationAction

Member Ends:

object, clearAssociationAction

Owned Association Ends

+ **clearAssociationAction** : ClearAssociationAction [0..1]

Package UML::Actions::IntermediateActions

Association A_object_structuralFeatureAction

Member Ends:

object, structuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **structuralFeatureAction** : StructuralFeatureAction [0..1]

Package UML::Actions::IntermediateActions

Association A_removeAt_removeStructuralFeatureValueAction

Member Ends:

removeAt, removeStructuralFeatureValueAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **removeStructuralFeatureValueAction** : RemoveStructuralFeatureValueAction [0..1]

Package UML::Actions::IntermediateActions

Association A_request_sendObjectAction

Member Ends:

request, sendObjectAction

Owned Association Ends

+ **sendObjectAction** : SendObjectAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_clearStructuralFeatureAction

Member Ends:

result, clearStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **clearStructuralFeatureAction** : ClearStructuralFeatureAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_createObjectAction

Member Ends:

result, createObjectAction

Owned Association Ends

+ **createObjectAction** : CreateObjectAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_readLinkAction

Member Ends:

result, readLinkAction

Owned Association Ends

+ **readLinkAction** : ReadLinkAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_readSelfAction

Member Ends:

result, readSelfAction

Owned Association Ends

+ **readSelfAction** : ReadSelfAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_readStructuralFeatureAction

Member Ends:

result, readStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **readStructuralFeatureAction** : ReadStructuralFeatureAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_testIdentityAction

Member Ends:

result, testIdentityAction

Owned Association Ends

+ **testIdentityAction** : TestIdentityAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_valueSpecificationAction

Member Ends:

result, valueSpecificationAction

Owned Association Ends

+ **valueSpecificationAction** : ValueSpecificationAction [0..1]

Package UML::Actions::IntermediateActions

Association A_result_writeStructuralFeatureAction

Member Ends:

result, writeStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **writeStructuralFeatureAction** : WriteStructuralFeatureAction [0..1]

Package UML::Actions::IntermediateActions

Association A_second_testIdentityAction

Member Ends:

second, testIdentityAction

Owned Association Ends

+ **testIdentityAction** : TestIdentityAction [0..1]

Package UML::Actions::IntermediateActions

Association A_signal_broadcastSignalAction

Member Ends:

signal, broadcastSignalAction

Owned Association Ends

+ **broadcastSignalAction** : BroadcastSignalAction [0..*]

Package UML::Actions::IntermediateActions

Association A_structuralFeature_structuralFeatureAction

Member Ends:

structuralFeature, structuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **structuralFeatureAction** : StructuralFeatureAction [0..*]

Package UML::Actions::IntermediateActions

Association A_target_destroyObjectAction

Member Ends:

target, destroyObjectAction

Owned Association Ends

+ **destroyObjectAction** : DestroyObjectAction [0..1]

Package UML::Actions::IntermediateActions

Association A_target_sendObjectAction

Member Ends:

target, sendObjectAction

Owned Association Ends

+ **sendObjectAction** : SendObjectAction [0..1]

Package UML::Actions::IntermediateActions

Association A_value_linkEndData

Member Ends:

value, linkEndData

Owned Association Ends

+ **linkEndData** : LinkEndData [0..1]

Package UML::Actions::IntermediateActions

Association A_value_valueSpecificationAction

Member Ends:

value, valueSpecificationAction

Owned Association Ends

+ **valueSpecificationAction** : ValueSpecificationAction [0..1]

Package UML::Actions::IntermediateActions

Association A_value_writeStructuralFeatureAction

Member Ends:

value, writeStructuralFeatureAction

Found in Diagrams:

Structural Feature Actions

Owned Association Ends

+ **writeStructuralFeatureAction** : WriteStructuralFeatureAction [0..1]

Package UML::Actions::StructuredActions

Nesting Package:

Actions

Imported Packages:

BasicActions, StructuredActivities

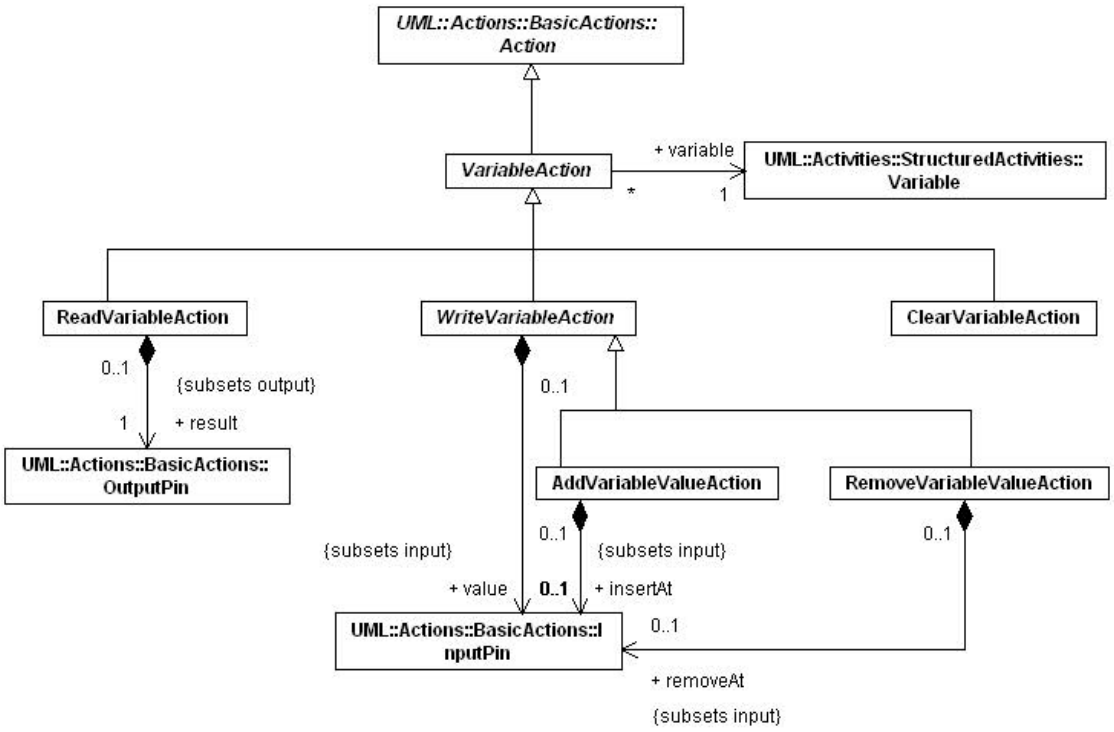
Diagram Summary
Variable Actions

Class Summary
ActionInputPin
AddVariableValueAction
ClearVariableAction
RaiseExceptionAction
ReadVariableAction
RemoveVariableValueAction
VariableAction
WriteVariableAction

Association Summary
A_exception_raiseExceptionAction
A_fromAction_actionInputPin
A_insertAt_addVariableValueAction
A_removeAt_removeVariableValueAction
A_result_readVariableAction
A_value_writeVariableAction
A_variable_variableAction

Package UML::Actions::StructuredActions

Diagram Variable Actions



Classifiers Local to Package:

AddVariableValueAction, ClearVariableAction, ReadVariableAction, RemoveVariableValueAction , VariableAction, WriteVariableAction

Classifiers External to Package:

Action, InputPin, OutputPin, Variable

Package UML::Actions::StructuredActions

Class ActionInputPin

An action input pin is a kind of pin that executes an action to determine the values to input to another.

Generalizations:

InputPin

Owned Association Ends

+ **fromAction** : Action [1..1] {subsets ownedElement}

The action used to provide values.

Constraints

input_pin

The fromAction of an action input pin must only have action input pins as input pins.

expression (OCL): true

no_control_or_data_flow

The fromAction of an action input pin cannot have control or data flows coming into or out of it or its pins.

expression (OCL): true

one_output_pin

The fromAction of an action input pin must have exactly one output pin.

expression (OCL): true

Package UML::Actions::StructuredActions

Class AddVariableValueAction

An add variable value action is a write variable action for adding values to a variable.

Generalizations:

WriteVariableAction

Found in Diagrams:

Variable Actions

Attributes

+ **isReplaceAll** : Boolean [1..1] = false

Specifies whether existing values of the variable should be removed before adding the new value.

Owned Association Ends

+ **insertAt** : InputPin [0..1] {subsets input}

Gives the position at which to insert a new value or move an existing value in ordered variables. The types is UnlimitedINatural, but the value cannot be zero. This pin is omitted for unordered variables.

Constraints

required_value

A value input pin is required.

expression (OCL): self.value -> notEmpty()

single_input_pin

Actions adding values to ordered variables must have a single input pin for the insertion point with type UnlimtedNatural and multiplicity of 1..1, otherwise the action has no input pin for the insertion point.

expression (OCL): let insertAtPins : Collection = self.insertAt in if self.variable.ordering = # unordered then insertAtPins->size() = 0 else let insertAtPin : InputPin = insertAt->asSequence()-> first() in insertAtPins->size() = 1 and insertAtPin.type = UnlimitedNatural and insertAtPin. multiplicity.is(1,1)) endif

Package UML::Actions::StructuredActions

Class ClearVariableAction

A clear variable action is a variable action that removes all values of a variable.

Generalizations:

VariableAction

Found in Diagrams:

Variable Actions

Package UML::Actions::StructuredActions

Class RaiseExceptionAction

A raise exception action is an action that causes an exception to occur. The input value becomes the exception object.

Generalizations:

Action

Owned Association Ends

+ **exception** : InputPin [1..1] {subsets input}

An input pin whose value becomes an exception object.

Package UML::Actions::StructuredActions

Class ReadVariableAction

A read variable action is a variable action that retrieves the values of a variable.

Generalizations:

VariableAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **result** : OutputPin [1..1] {subsets output}

Gives the output pin on which the result is put.

Constraints

compatible_multiplicity

The multiplicity of the variable must be compatible with the multiplicity of the output pin.

expression (OCL): self.variable.multiplicity.compatibleWith(self.result.multiplicity)

type_and_ordering

The type and ordering of the result output pin of a read-variable action are the same as the type and ordering of the variable.

expression (OCL): self.result.type =self.variable.type and self.result.ordering = self.variable.ordering

Package UML::Actions::StructuredActions

Class RemoveVariableValueAction

A remove variable value action is a write variable action that removes values from variables.

Generalizations:

WriteVariableAction

Found in Diagrams:

Variable Actions

Attributes

+ **isRemoveDuplicates** : Boolean [1..1] = false

Specifies whether to remove duplicates of the value in nonunique variables.

Owned Association Ends

+ **removeAt** : InputPin [0..1] {subsets input}

Specifies the position of an existing value to remove in ordered nonunique variables. The type of the pin is UnlimitedNatural, but the value cannot be zero or unlimited.

Constraints

unlimited_natural

Actions removing a value from ordered non-unique variables must have a single removeAt input pin and no value input pin if isRemoveDuplicates is false. The removeAt pin must be of type Unlimited Natural with multiplicity 1..1. Otherwise, the action has a value input pin and no removeAt input pin.

expression (OCL): if not self.variable.isOrdered or self.variable.isUnique or isRemoveDuplicates then self.removeAt -> isEmpty() and self.value -> notEmpty() else self.value -> isEmpty() and self.removeAt -> notEmpty() and self.removeAt.type = UnlimitedNatural and self.removeAt.lower() = 1 and self.removeAt.upper() = 1 endif

Package UML::Actions::StructuredActions

Class VariableAction

VariableAction is an abstract class for actions that operate on a statically specified variable.

Generalizations:

Action

Specializations:

ClearVariableAction, ReadVariableAction, WriteVariableAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **variable** : Variable [1..1]

Variable to be read.

Constraints

scope_of_variable

The action must be in the scope of the variable.

expression (OCL): self.variable.isAccessibleBy(self)

Package UML::Actions::StructuredActions

Class WriteVariableAction

WriteVariableAction is an abstract class for variable actions that change variable values.

Generalizations:

VariableAction

Specializations:

AddVariableValueAction, RemoveVariableValueAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **value** : InputPin [0..1] {subsets input}

Value to be added or removed from the variable.

Constraints

multiplicity

The multiplicity of the input pin is 1..1.

expression (OCL): self.value.multiplicity.is(1,1)

same_type

The type input pin is the same as the type of the variable.

expression (OCL): self.value -> notEmpty() implies self.value.type = self.variable.type

Package UML::Actions::StructuredActions

Association A_exception_raiseExceptionAction

Member Ends:

exception, raiseExceptionAction

Owned Association Ends

+ **raiseExceptionAction** : RaiseExceptionAction [0..1]

Package UML::Actions::StructuredActions

Association A_fromAction_actionInputPin

Member Ends:

fromAction, actionInputPin

Owned Association Ends

+ **actionInputPin** : ActionInputPin [0..1]

Package UML::Actions::StructuredActions

Association A_insertAt_addVariableValueAction

Member Ends:

insertAt, addVariableValueAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **addVariableValueAction** : AddVariableValueAction [0..1]

Package UML::Actions::StructuredActions

Association A_removeAt_removeVariableValueAction

Member Ends:

removeAt, removeVariableValueAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **removeVariableValueAction** : RemoveVariableValueAction [0..1]

Package UML::Actions::StructuredActions

Association A_result_readVariableAction

Member Ends:

result, readVariableAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **readVariableAction** : ReadVariableAction [0..1]

Package UML::Actions::StructuredActions

Association A_value_writeVariableAction

Member Ends:

value, writeVariableAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **writeVariableAction** : WriteVariableAction [0..1]

Package UML::Actions::StructuredActions

Association A_variable_variableAction

Member Ends:

variable, variableAction

Found in Diagrams:

Variable Actions

Owned Association Ends

+ **variableAction** : VariableAction [0..*]

Package UML::Activities

Nesting Package:

UML

Imported Packages:

CommonBehaviors, CompositeStructures, StateMachines

Nested Package Summary
BasicActivities
CompleteActivities
CompleteStructuredActivities
ExtraStructuredActivities
FundamentalActivities
IntermediateActivities
StructuredActivities

Package UML::Activities::BasicActivities

Nesting Package:

Activities

Merged Packages:

BasicBehaviors, FundamentalActivities

Class Summary
Activity
ActivityEdge
ActivityFinalNode
ActivityGroup
ActivityNode
ActivityParameterNode
ControlFlow
ControlNode
InitialNode
ObjectFlow
ObjectNode
Pin
ValuePin

Association Summary
A_containedEdge_inGroup
A_edge_activity
A_outgoing_source
A_parameter_activityParameterNode
A_redefinedEdge_activityEdge
A_redefinedNode_activityNode
A_target_incoming

Package UML::Activities::BasicActivities

Class Activity

Generalizations:

Behavior

Attributes

+ **isReadOnly** : Boolean [1..1] = false

If true, this activity must not make any changes to variables outside the activity or to objects. (This is an assertion, not an executable property. It may be used by an execution engine to optimize model execution. If the assertion is violated by the action, then the model is ill-formed.) The default is false (an activity may make nonlocal changes).

Owned Association Ends

+ **edge** : ActivityEdge [0..*] {subsets ownedElement}

Edges expressing flow between nodes of the activity.

Constraints

activity_parameter_node

The nodes of the activity must include one ActivityParameterNode for each parameter.

expression (OCL): true

autonomous

An activity cannot be autonomous and have a classifier or behavioral feature context at the same time.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ActivityEdge

An activity edge is an abstract class for directed connections between two activity nodes.

Generalizations:

RedefinableElement

Specializations:

ControlFlow, ObjectFlow

Owned Association Ends

+ **activity** : Activity [0..1] {subsets owner}

Activity containing the edge.

+ /**inGroup** : ActivityGroup [0..*] {readOnly, union}

Groups containing the edge.

+ **redefinedEdge** : ActivityEdge [0..*] {subsets redefinedElement}

Inherited edges replaced by this edge in a specialization of the activity.

+ **source** : ActivityNode [1..1]

Node from which tokens are taken when they traverse the edge.

+ **target** : ActivityNode [1..1]

Node to which tokens are put when they traverse the edge.

Constraints

owned

Activity edges may be owned only by activities or groups.

expression (OCL): true

source_and_target

The source and target of an edge must be in the same activity as the edge.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ActivityFinalNode

An activity final node is a final node that stops all flows in an activity.

Generalizations:

ControlNode

Package UML::Activities::BasicActivities

Class ActivityGroup

Specializations:

InterruptibleActivityRegion

Owned Association Ends

+ /**containedEdge** : ActivityEdge [0..*] {readOnly, union}

Edges immediately contained in the group.

Constraints

group_owned

Groups may only be owned by activities or groups.

expression (OCL): true

nodes_and_edges

All nodes and edges of the group must be in the same activity as the group.

expression (OCL): true

not_contained

No node or edge in a group may be contained by its subgroups or its containing groups, transitively.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ActivityNode

Generalizations:

RedefinableElement

Specializations:

ControlNode, ObjectNode

Owned Association Ends

+ **incoming** : ActivityEdge [0..*]

Edges that have the node as target.

+ **outgoing** : ActivityEdge [0..*]

Edges that have the node as source.

+ **redefinedNode** : ActivityNode [0..*] {subsets redefinedElement}

Inherited nodes replaced by this node in a specialization of the activity.

Constraints

owned

Activity nodes can only be owned by activities or groups.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ActivityParameterNode

An activity parameter node is an object node for inputs and outputs to activities.

Generalizations:

ObjectNode

Owned Association Ends

+ **parameter** : Parameter [1..1]

The parameter the object node will be accepting or providing values for.

Constraints

has_parameters

Activity parameter nodes must have parameters from the containing activity.

expression (OCL): true

maximum_one_parameter_node

A parameter with direction other than inout must have at most one activity parameter node in an activity.

expression (OCL): true

maximum_two_parameter_nodes

A parameter with direction inout must have at most two activity parameter nodes in an activity, one with incoming flows and one with outgoing flows.

expression (OCL): true

no_edges

An activity parameter node may have all incoming edges or all outgoing edges, but it must not have both incoming and outgoing edges.

expression (OCL): true

no_incoming_edges

Activity parameter object nodes with no incoming edges and one or more outgoing edges must have a parameter with in or inout direction.

expression (OCL): true

no_outgoing_edges

Activity parameter object nodes with no outgoing edges and one or more incoming edges must have a parameter with out, inout, or return direction.

Package UML::Activities::BasicActivities

Class ActivityParameterNode

expression (OCL): true

same_type

The type of an activity parameter node is the same as the type of its parameter.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ControlFlow

A control flow is an edge that starts an activity node after the previous one is finished.

Generalizations:

ActivityEdge

Constraints

object_nodes

Control flows may not have object nodes at either end, except for object nodes with control type.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ControlNode

A control node is an abstract activity node that coordinates flows in an activity.

Generalizations:

ActivityNode

Specializations:

ActivityFinalNode, DecisionNode, FinalNode, ForkNode, InitialNode, JoinNode, JoinNode, MergeNode

Package UML::Activities::BasicActivities

Class InitialNode

An initial node is a control node at which flow starts when the activity is invoked.

Generalizations:

ControlNode

Constraints

control_edges

Only control edges can have initial nodes as source.

expression (OCL): true

no_incoming_edges

An initial node has no incoming edges.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ObjectFlow

An object flow is an activity edge that can have objects or data passing along it.

Generalizations:

ActivityEdge

Constraints

compatible_types

Object nodes connected by an object flow, with optionally intervening control nodes, must have compatible types. In particular, the downstream object node type must be the same or a supertype of the upstream object node type.

expression (OCL): true

no_actions

Object flows may not have actions at either end.

expression (OCL): true

same_upper_bounds

Object nodes connected by an object flow, with optionally intervening control nodes, must have the same upper bounds.

expression (OCL): true

Package UML::Activities::BasicActivities

Class ObjectNode

An object node is an abstract activity node that is part of defining object flow in an activity.

Generalizations:

ActivityNode, TypedElement

Specializations:

ActivityParameterNode, CentralBufferNode, ExpansionNode, Pin

Constraints

object_flow_edges

All edges coming into or going out of object nodes must be object flow edges.

expression (OCL): true

Package UML::Activities::BasicActivities

Class Pin

A pin is an object node for inputs and outputs to actions.

Generalizations:

ObjectNode

Package UML::Activities::BasicActivities

Class ValuePin

Constraints

no_incoming_edges

Value pins have no incoming edges.

expression (OCL): true

Package UML::Activities::BasicActivities

Association A_containedEdge_inGroup

Member Ends:

containedEdge, inGroup

Package UML::Activities::BasicActivities

Association A_edge_activity

Member Ends:

edge, activity

Package UML::Activities::BasicActivities

Association A_outgoing_source

Member Ends:

outgoing, source

Package UML::Activities::BasicActivities

Association A_parameter_activityParameterNode

Member Ends:

parameter, activityParameterNode

Owned Association Ends

+ **activityParameterNode** : ActivityParameterNode [0..*]

Package UML::Activities::BasicActivities

Association A_redefinedEdge_activityEdge

Member Ends:

redefinedEdge, activityEdge

Owned Association Ends

+ **activityEdge** : ActivityEdge [0..*]

Package UML::Activities::BasicActivities

Association A_redefinedNode_activityNode

Member Ends:

redefinedNode, activityNode

Owned Association Ends

+ **activityNode** : ActivityNode [0..*]

Package UML::Activities::BasicActivities

Association A_target_incoming

Member Ends:

target, incoming

Package UML::Activities::CompleteActivities

Nesting Package:

Activities

Imported Packages:

BasicBehaviors, BehaviorStateMachines, Kernel

Merged Packages:

IntermediateActivities

Diagram Summary
Complete Activities Elements

Class Summary
Action
Activity
ActivityEdge
ActivityGroup
ActivityNode
Behavior
BehavioralFeature
DataStoreNode
InterruptibleActivityRegion
JoinNode
ObjectFlow
ObjectNode
Parameter
ParameterSet
Pin

Enumeration Summary
ObjectNodeOrderingKind
ParameterEffectKind

Association Summary
A_condition_parameterSet
A_containedNode_inGroup
A_inInterruptibleRegion_node

Package UML::Activities::CompleteActivities

A_inState_objectNode
A_interruptingEdge_interrupts
A_joinSpec_joinNode
A_localPostcondition_action
A_localPrecondition_action
A_ownedParameterSet_behavior
A_ownedParameterSet_behavioralFeature
A_parameterSet_parameter
A_selection_objectFlow
A_selection_objectNode
A_transformation_objectFlow
A_upperBound_objectNode
A_weight_activityEdge

Package UML::Activities::CompleteActivities

Diagram Complete Activities Elements

Classifiers Local to Package:

Activity

Package UML::Activities::CompleteActivities

Class Action

An action has pre- and post-conditions.

Generalizations:

NamedElement

Owned Association Ends

+ **localPostcondition** : Constraint [0..*] {subsets ownedElement}

Constraint that must be satisfied when executed is completed.

+ **localPrecondition** : Constraint [0..*] {subsets ownedElement}

Constraint that must be satisfied when execution is started.

Package UML::Activities::CompleteActivities

Class Activity

Found in Diagrams:

Complete Activities Elements

Attributes

+ **isSingleExecution** : Boolean [1..1] = false

If true, all invocations of the activity are handled by the same execution.

Package UML::Activities::CompleteActivities

Class ActivityEdge

Activity edges can be contained in interruptible regions.

Generalizations:

RedefinableElement

Owned Association Ends

+ **interrupts** : InterruptibleActivityRegion [0..1]

Region that the edge can interrupt.

+ **weight** : ValueSpecification [1..1] {subsets ownedElement}

The minimum number of tokens that must traverse the edge at the same time.

Package UML::Activities::CompleteActivities

Class ActivityGroup

Owned Association Ends

+ /**containedNode** : ActivityNode [0..*] {readOnly, union}

Nodes immediately contained in the group.

Package UML::Activities::CompleteActivities

Class ActivityNode

Owned Association Ends

+ /**inGroup** : ActivityGroup [0..*] {readOnly, union}

  Groups containing the node.

+ **inInterruptibleRegion** : InterruptibleActivityRegion [0..*] {subsets inGroup}

  Interruptible regions containing the node.

Package UML::Activities::CompleteActivities

Class Behavior

A behavior owns zero or more parameter sets.

Generalizations:

Class

Owned Association Ends

+ **ownedParameterSet** : ParameterSet [0..*] {subsets ownedMember}

The ParameterSets owned by this Behavior.

Package UML::Activities::CompleteActivities

Class BehavioralFeature

A behavioral feature owns zero or more parameter sets.

Generalizations:

Feature, Namespace

Owned Association Ends

+ **ownedParameterSet** : ParameterSet [0..*] {subsets ownedMember}

The ParameterSets owned by this BehavioralFeature.

Package UML::Activities::CompleteActivities

Class DataStoreNode

A data store node is a central buffer node for non-transient information.

Generalizations:

CentralBufferNode

Package UML::Activities::CompleteActivities

Class InterruptibleActivityRegion

An interruptible activity region is an activity group that supports termination of tokens flowing in the portions of an activity.

Generalizations:

ActivityGroup

Owned Association Ends

+ **interruptingEdge** : ActivityEdge [0..*]

The edges leaving the region that will abort other tokens flowing in the region.

+ **node** : ActivityNode [0..*] {subsets containedNode}

Nodes immediately contained in the group.

Constraints

interrupting_edges

Interrupting edges of a region must have their source node in the region and their target node outside the region in the same activity containing the region.

expression (OCL): true

Package UML::Activities::CompleteActivities

Class JoinNode

Join nodes have a Boolean value specification using the names of the incoming edges to specify the conditions under which the join will emit a token.

Generalizations:

ControlNode

Attributes

+ **isCombineDuplicate** : Boolean [1..1] = true

Tells whether tokens having objects with the same identity are combined into one by the join.

Owned Association Ends

+ **joinSpec** : ValueSpecification [1..1] {subsets ownedElement}

A specification giving the conditions under which the join with emit a token. Default is "and".

Constraints

incoming_object_flow

If a join node has an incoming object flow, it must have an outgoing object flow, otherwise, it must have an outgoing control flow.

expression (OCL): (self.incoming.select(e | e.isTypeOf(ObjectFlow)->notEmpty() implies self.outgoing.isTypeOf(ObjectFlow)) and (self.incoming.select(e | e.isTypeOf(ObjectFlow)->empty() implies self.outgoing.isTypeOf(ControlFlow))

one_outgoing_edge

A join node has one outgoing edge.

expression (OCL): self.outgoing->size() = 1

Package UML::Activities::CompleteActivities

Class ObjectFlow

Object flows have support for multicast/receive, token selection from object nodes, and transformation of tokens.

Attributes

+ **isMulticast** : Boolean [1..1] = false

Tells whether the objects in the flow are passed by multicasting.

+ **isMultireceive** : Boolean [1..1] = false

Tells whether the objects in the flow are gathered from respondents to multicasting.

Owned Association Ends

+ **selection** : Behavior [0..1]

Selects tokens from a source object node.

+ **transformation** : Behavior [0..1]

Changes or replaces data tokens flowing along edge.

Constraints

input_and_output_parameter

A selection behavior has one input parameter and one output parameter. The input parameter must be a bag of elements of the same as or a supertype of the type of source object node. The output parameter must be the same or a subtype of the type of source object node. The behavior cannot have side effects.

expression (OCL): true

is_multicast_or_is_multireceive

isMulticast and isMultireceive cannot both be true.

expression (OCL): true

selection_behaviour

An object flow may have a selection behavior only if has an object node as a source.

expression (OCL): true

target

Package UML::Activities::CompleteActivities

Class ObjectFlow

An edge with constant weight may not target an object node, or lead to an object node downstream with no intervening actions, that has an upper bound less than the weight.

expression (OCL): true

transformation_behaviour

A transformation behavior has one input parameter and one output parameter. The input parameter must be the same as or a supertype of the type of object token coming from the source end. The output parameter must be the same or a subtype of the type of object token expected downstream. The behavior cannot have side effects.

expression (OCL): true

Package UML::Activities::CompleteActivities

Class ObjectNode

Object nodes have support for token selection, limitation on the number of tokens, specifying the state required for tokens, and carrying control values.

Generalizations:

TypedElement

Attributes

+ **isControlType** : Boolean [1..1] = false

Tells whether the type of the object node is to be treated as control.

+ **ordering** : ObjectNodeOrderingKind [1..1] = FIFO

Tells whether and how the tokens in the object node are ordered for selection to traverse edges outgoing from the object node.

Owned Association Ends

+ **inState** : State [0..*]

The required states of the object available at this point in the activity.

+ **selection** : Behavior [0..1]

Selects tokens for outgoing edges.

+ **upperBound** : ValueSpecification [1..1] {subsets ownedElement}

The maximum number of tokens allowed in the node. Objects cannot flow into the node if the upper bound is reached.

Constraints

input_output_parameter

A selection behavior has one input parameter and one output parameter. The input parameter must be a bag of elements of the same type as the object node or a supertype of the type of object node. The output parameter must be the same or a subtype of the type of object node. The behavior cannot have side effects.

expression (OCL): true

selection_behavior

Package UML::Activities::CompleteActivities

Class ObjectNode

If an object node has a selection behavior, then the ordering of the object node is ordered, and vice versa.

expression (OCL): true

Package UML::Activities::CompleteActivities

Class Parameter

Parameters have support for streaming, exceptions, and parameter sets.

Attributes

+ **effect** : ParameterEffectKind [0..1]

Specifies the effect that the owner of the parameter has on values passed in or out of the parameter.

+ **isException** : Boolean [1..1] = false

Tells whether an output parameter may emit a value to the exclusion of the other outputs.

+ **isStream** : Boolean [1..1] = false

Tells whether an input parameter may accept values while its behavior is executing, or whether an output parameter post values while the behavior is executing.

Owned Association Ends

+ **parameterSet** : ParameterSet [0..*]

The parameter sets containing the parameter. See ParameterSet.

Constraints

in_and_out

Only in and inout parameters may have a delete effect. Only out, inout, and return parameters may have a create effect.

expression (OCL): true

not_exception

An input parameter cannot be an exception.

expression (OCL): true

reentrant_behaviors

Reentrant behaviors cannot have stream parameters.

expression (OCL): true

stream_and_exception

A parameter cannot be a stream and exception at the same time.

expression (OCL): true

Package UML::Activities::CompleteActivities

Class ParameterSet

A parameter set is an element that provides alternative sets of inputs or outputs that a behavior may use.

Generalizations:

NamedElement

Owned Association Ends

+ **condition** : Constraint [0..*] {subsets ownedElement}

Constraint that should be satisfied for the owner of the parameters in an input parameter set to start execution using the values provided for those parameters, or the owner of the parameters in an output parameter set to end execution providing the values for those parameters, if all preconditions and conditions on input parameter sets were satisfied.

+ **parameter** : Parameter [1..*]

Parameters in the parameter set.

Constraints

input

If a behavior has input parameters that are in a parameter set, then any inputs that are not in a parameter set must be streaming. Same for output parameters.

expression (OCL): true

same_parameterized_entity

The parameters in a parameter set must all be inputs or all be outputs of the same parameterized entity, and the parameter set is owned by that entity.

expression (OCL): true

two_parameter_sets

Two parameter sets cannot have exactly the same set of parameters.

expression (OCL): true

Package UML::Activities::CompleteActivities

Class Pin

Attributes

+ **isControl** : Boolean [1..1] = false

Tells whether the pins provide data to the actions, or just controls when it executes it.

Constraints

control_pins

Control pins have a control type

expression (OCL): isControl implies isControlType

Package UML::Activities::CompleteActivities

Enumeration ObjectNodeOrderingKind

ObjectNodeOrderingKind is an enumeration indicating queuing order within a node.

Enumeration Literals

FIFO

Indicates that object node tokens are queued in a first in, first out manner.

LIFO

Indicates that object node tokens are queued in a last in, first out manner.

ordered

Indicates that object node tokens are ordered.

unordered

Indicates that object node tokens are unordered.

Package UML::Activities::CompleteActivities

Enumeration ParameterEffectKind

The datatype ParameterEffectKind is an enumeration that indicates the effect of a behavior on values passed in or out of its parameters.

Enumeration Literals

create

Indicates that the behavior creates values.

delete

Indicates that the behavior deletes values.

read

Indicates that the behavior reads values.

update

Indicates that the behavior updates values.

Package UML::Activities::CompleteActivities

Association A_condition_parameterSet

Member Ends:

condition, parameterSet

Owned Association Ends

+ **parameterSet** : ParameterSet [0..1]

Package UML::Activities::CompleteActivities

Association A_containedNode_inGroup

Member Ends:

containedNode, inGroup

Package UML::Activities::CompleteActivities

Association A_inInterruptibleRegion_node

Member Ends:

inInterruptibleRegion, node

Package UML::Activities::CompleteActivities

Association A_inState_objectNode

Member Ends:

inState, objectNode

Owned Association Ends

+ **objectNode** : ObjectNode [0..*]

Package UML::Activities::CompleteActivities

Association A_interruptingEdge_interrupts

Member Ends:

interruptingEdge, interrupts

Package UML::Activities::CompleteActivities

Association A_joinSpec_joinNode

Member Ends:

joinSpec, joinNode

Owned Association Ends

+ **joinNode** : JoinNode [0..1]

Package UML::Activities::CompleteActivities

Association A_localPostcondition_action

Member Ends:

localPostcondition, action

Owned Association Ends

+ **action** : Action [0..1]

Package UML::Activities::CompleteActivities

Association A_localPrecondition_action

Member Ends:

localPrecondition, action

Owned Association Ends

+ **action** : Action [0..1]

Package UML::Activities::CompleteActivities

Association A_ownedParameterSet_behavior

Member Ends:

ownedParameterSet, behavior

Owned Association Ends

+ **behavior** : Behavior [0..1]

Package UML::Activities::CompleteActivities

Association A_ownedParameterSet_behavioralFeature

Member Ends:

ownedParameterSet, behavioralFeature

Owned Association Ends

+ **behavioralFeature** : BehavioralFeature [0..1]

Package UML::Activities::CompleteActivities

Association A_parameterSet_parameter

Member Ends:

parameterSet, parameter

Package UML::Activities::CompleteActivities

Association A_selection_objectFlow

Member Ends:

selection, objectFlow

Owned Association Ends

+ **objectFlow** : ObjectFlow [0..*]

Package UML::Activities::CompleteActivities

Association A_selection_objectNode

Member Ends:

selection, objectNode

Owned Association Ends

+ **objectNode** : ObjectNode [0..*]

Package UML::Activities::CompleteActivities

Association A_transformation_objectFlow

Member Ends:

transformation, objectFlow

Owned Association Ends

+ **objectFlow** : ObjectFlow [0..*]

Package UML::Activities::CompleteActivities

Association A_upperBound_objectNode

Member Ends:

upperBound, objectNode

Owned Association Ends

+ **objectNode** : ObjectNode [0..1]

Package UML::Activities::CompleteActivities

Association A_weight_activityEdge

Member Ends:

weight, activityEdge

Owned Association Ends

+ **activityEdge** : ActivityEdge [0..1]

Package UML::Activities::CompleteStructuredActivities

Nesting Package:

Activities

Merged Packages:

BasicActivities, StructuredActivities

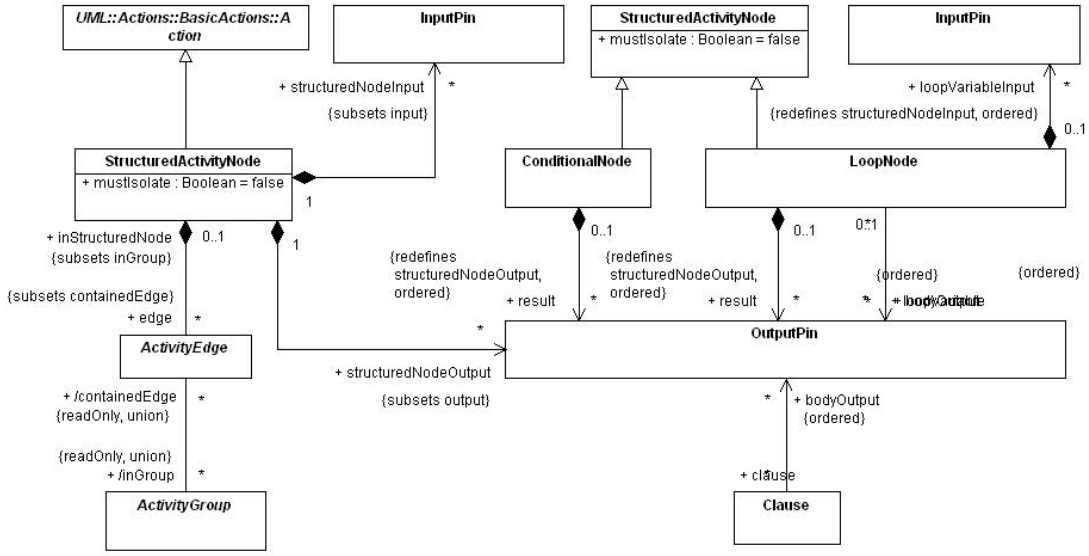
Diagram Summary
Complete Structured Activities

Class Summary
ActivityEdge
ActivityGroup
Clause
ConditionalNode
InputPin
LoopNode
OutputPin
StructuredActivityNode

Association Summary
A_bodyOutput_clause
A_bodyOutput_loopNode
A_containedEdge_inGroup
A_edge_inStructuredNode
A_loopVariableInput_loopNode
A_loopVariable_loopNode
A_result_conditionalNode
A_result_loopNode
A_structuredNodeInput_structuredActivityNode
A_structuredNodeOutput_structuredActivityNode

Package UML::Activities::CompleteStructuredActivities

Diagram Complete Structured Activities



Classifiers Local to Package:

ActivityEdge, ActivityGroup, Clause, ConditionalNode, InputPin, LoopNode, OutputPin, StructuredActivityNode

Classifiers External to Package:

Action

Package UML::Activities::CompleteStructuredActivities

Class ActivityEdge

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **/inGroup** : ActivityGroup [0..*] {readOnly, union}

Groups containing the edge.

+ **inStructuredNode** : StructuredActivityNode [0..1] {subsets inGroup}

Structured activity node containing the edge.

Constraints

structured_node

Activity edges may be owned by at most one structured node.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Class ActivityGroup

Specializations:

StructuredActivityNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ /**containedEdge** : ActivityEdge [0..*] {readOnly, union}

Edges immediately contained in the group.

Package UML::Activities::CompleteStructuredActivities

Class Clause

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **bodyOutput** : OutputPin [0..*] {ordered}

A list of output pins within the body fragment whose values are moved to the result pins of the containing conditional node after execution of the clause body.

Constraints

body_output_pins

The bodyOutput pins are output pins on actions in the body of the clause.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Class ConditionalNode

Generalizations:

StructuredActivityNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **result** : OutputPin [0..*] {ordered, redefines structuredNodeOutput}

A list of output pins that constitute the data flow outputs of the conditional.

Constraints

matching_output_pins

Each clause of a conditional node must have the same number of bodyOutput pins as the conditional node has result output pins, and each clause bodyOutput pin must be compatible with the corresponding result pin (by positional order) in type, multiplicity, ordering and uniqueness.

expression (OCL): true

no_input_pins

A conditional node has no input pins.

expression (OCL): true

result_no_incoming

The result output pins have no incoming edges.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Class InputPin

Found in Diagrams:

Complete Structured Activities

Constraints

outgoing_edges_structured_only

Input pins may have outgoing edges only when they are on actions that are structured nodes, and these edges must target a node contained by the structured node.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Class LoopNode

Generalizations:

StructuredActivityNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **bodyOutput** : OutputPin [0..*] {ordered}

A list of output pins within the body fragment the values of which are moved to the loop variable pins after completion of execution of the body, before the next iteration of the loop begins or before the loop exits.

+ **loopVariable** : OutputPin [0..*] {ordered}

A list of output pins that hold the values of the loop variables during an execution of the loop. When the test fails, the values are movied to the result pins of the loop.

+ **loopVariableInput** : InputPin [0..*] {ordered, redefines structuredNodeInput}

A list of values that are moved into the loop variable pins before the first iteration of the loop.

+ **result** : OutputPin [0..*] {ordered, redefines structuredNodeOutput}

A list of output pins that constitute the data flow output of the entire loop.

Constraints

body_output_pins

The bodyOutput pins are output pins on actions in the body of the loop node.

expression (OCL): true

input_edges

Loop variable inputs must not have outgoing edges.

expression (OCL): true

result_no_incoming

The result output pins have no incoming edges.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Class OutputPin

Found in Diagrams:

Complete Structured Activities

Constraints

incoming_edges_structured_only

Output pins may have incoming edges only when they are on actions that are structured nodes, and these edges may not target a node contained by the structured node.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Class StructuredActivityNode

Because of the concurrent nature of the execution of actions within and across activities, it can be difficult to guarantee the consistent access and modification of object memory. In order to avoid race conditions or other concurrency-related problems, it is sometimes necessary to isolate the effects of a group of actions from the effects of actions outside the group. This may be indicated by setting the mustIsolate attribute to true on a structured activity node. If a structured activity node is "isolated," then any object used by an action within the node cannot be accessed by any action outside the node until the structured activity node as a whole completes. Any concurrent actions that would result in accessing such objects are required to have their execution deferred until the completion of the node.

Generalizations:

Action, ActivityGroup

Specializations:

ConditionalNode, LoopNode

Found in Diagrams:

Complete Structured Activities

Attributes

+ **mustIsolate** : Boolean [1..1] = false

If true, then the actions in the node execute in isolation from actions outside the node.

Owned Association Ends

+ **edge** : ActivityEdge [0..*] {subsets containedEdge}

Edges immediately contained in the structured node.

+ **structuredNodeInput** : InputPin [0..*] {subsets input}

+ **structuredNodeOutput** : OutputPin [0..*] {subsets output}

Constraints

edges

The edges owned by a structured node must have source and target nodes in the structured node, and vice versa.

expression (OCL): true

input_pin_edges

Package UML::Activities::CompleteStructuredActivities

Class StructuredActivityNode

The incoming edges of the input pins of a StructuredActivityNode must have sources that are not within the StructuredActivityNode.

expression (OCL): true

output_pin_edges

The outgoing edges of the output pins of a StructuredActivityNode must have targets that are not within the StructuredActivityNode.

expression (OCL): true

Package UML::Activities::CompleteStructuredActivities

Association A_bodyOutput_clause

Member Ends:

bodyOutput, clause

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **clause** : Clause [0..*]

Package UML::Activities::CompleteStructuredActivities

Association A_bodyOutput_loopNode

Member Ends:

bodyOutput, loopNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..*]

Package UML::Activities::CompleteStructuredActivities

Association A_containedEdge_inGroup

Member Ends:

containedEdge, inGroup

Found in Diagrams:

Complete Structured Activities

Package UML::Activities::CompleteStructuredActivities

Association A_edge_inStructuredNode

Member Ends:

edge, inStructuredNode

Found in Diagrams:

Complete Structured Activities

Package UML::Activities::CompleteStructuredActivities

Association A_loopVariableInput_loopNode

Member Ends:

loopVariableInput, loopNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::CompleteStructuredActivities

Association A_loopVariable_loopNode

Member Ends:

loopVariable, loopNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::CompleteStructuredActivities

Association A_result_conditionalNode

Member Ends:

result, conditionalNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **conditionalNode** : ConditionalNode [0..1]

Package UML::Activities::CompleteStructuredActivities

Association A_result_loopNode

Member Ends:

result, loopNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::CompleteStructuredActivities

Association A_structuredNodeInput_structuredActivityNode

Member Ends:

structuredNodeInput, structuredActivityNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **structuredActivityNode** : StructuredActivityNode [1..1]

Package UML::Activities::CompleteStructuredActivities

Association A_structuredNodeOutput_structuredActivityNode

Member Ends:

structuredNodeOutput, structuredActivityNode

Found in Diagrams:

Complete Structured Activities

Owned Association Ends

+ **structuredActivityNode** : StructuredActivityNode [1..1]

Package UML::Activities::ExtraStructuredActivities

Nesting Package:

Activities

Imported Packages:

BasicActivities

Merged Packages:

StructuredActivities

Class Summary
ExceptionHandler
ExecutableNode
ExpansionNode
ExpansionRegion

Enumeration Summary
ExpansionKind

Association Summary
A_exceptionInput_exceptionHandler
A_exceptionType_exceptionHandler
A_handlerBody_exceptionHandler
A_handler_protectedNode
A_inputElement_regionAsInput
A_outputElement_regionAsOutput

Package UML::Activities::ExtraStructuredActivities

Class ExceptionHandler

An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node.

Generalizations:

Element

Owned Association Ends

+ **exceptionInput** : ObjectNode [1..1]

An object node within the handler body. When the handler catches an exception, the exception token is placed in this node, causing the body to execute.

+ **exceptionType** : Classifier [1..*]

The kind of instances that the handler catches. If an exception occurs whose type is any of the classifiers in the set, the handler catches the exception and executes its body.

+ **handlerBody** : ExecutableNode [1..1]

A node that is executed if the handler satisfies an uncaught exception.

+ **protectedNode** : ExecutableNode [1..1] {subsets owner}

The node protected by the handler. The handler is examined if an exception propagates to the outside of the node.

Constraints

edge_source_target

An edge that has a source in an exception handler structured node must have its target in the handler also, and vice versa.

expression (OCL): true

exception_body

The exception handler and its input object node are not the source or target of any edge.

expression (OCL): true

one_input

The handler body has one input, and that input is the same as the exception input.

expression (OCL): true

Package UML::Activities::ExtraStructuredActivities

Class ExceptionHandler

result_pins

If the protected node is a StructuredActivityNode with output pins, then the exception handler body must also be a StructuredActivityNode with output pins that correspond in number and types to those of the protected node.

expression (OCL): true

Package UML::Activities::ExtraStructuredActivities

Class ExecutableNode

An executable node is an abstract class for activity nodes that may be executed. It is used as an attachment point for exception handlers.

Generalizations:

ActivityNode

Owned Association Ends

+ **handler** : ExceptionHandler [0..*] {subsets ownedElement}

A set of exception handlers that are examined if an uncaught exception propagates to the outer level of the executable node.

Constraints

region_as_input_or_putput

One of regionAsInput or regionAsOutput must be non-empty, but not both.

expression (OCL): true

Package UML::Activities::ExtraStructuredActivities

Class ExpansionNode

An expansion node is an object node used to indicate a flow across the boundary of an expansion region. A flow into a region contains a collection that is broken into its individual elements inside the region, which is executed once per element. A flow out of a region combines individual elements into a collection for use outside the region.

Generalizations:

ObjectNode

Owned Association Ends

+ **regionAsInput** : ExpansionRegion [0..1]

The expansion region for which the node is an input.

+ **regionAsOutput** : ExpansionRegion [0..1]

The expansion region for which the node is an output.

Package UML::Activities::ExtraStructuredActivities

Class ExpansionRegion

An expansion region is a structured activity region that executes multiple times corresponding to elements of an input collection.

Generalizations:

StructuredActivityNode

Attributes

+ **mode** : ExpansionKind [1..1] = iterative

The way in which the executions interact:
parallel: all interactions are independent
iterative: the interactions occur in order of the elements
stream: a stream of values flows into a single execution

Owned Association Ends

+ **inputElement** : ExpansionNode [1..*]

An object node that holds a separate element of the input collection during each of the multiple executions of the region.

+ **outputElement** : ExpansionNode [0..*]

An object node that accepts a separate element of the output collection during each of the multiple executions of the region. The values are formed into a collection that is available when the execution of the region is complete.

Constraints

expansion_nodes

An ExpansionRegion must have one or more argument ExpansionNodes and zero or more result ExpansionNodes.

expression (OCL): true

Package UML::Activities::ExtraStructuredActivities

Enumeration ExpansionKind

ExpansionKind is an enumeration type used to specify how multiple executions of an expansion region interact.

Enumeration Literals

iterative

The executions are dependent and must be executed one at a time, in order of the collection elements.

parallel

The executions are independent. They may be executed concurrently.

stream

A stream of collection elements flows into a single execution, in order of the collection elements.

Package UML::Activities::ExtraStructuredActivities

Association A_exceptionInput_exceptionHandler

Member Ends:

exceptionInput, exceptionHandler

Owned Association Ends

+ **exceptionHandler** : ExceptionHandler [0..*]

Package UML::Activities::ExtraStructuredActivities

Association A_exceptionType_exceptionHandler

Member Ends:

exceptionType, exceptionHandler

Owned Association Ends

+ **exceptionHandler** : ExceptionHandler [0..*]

Package UML::Activities::ExtraStructuredActivities

Association A_handlerBody_exceptionHandler

Member Ends:

handlerBody, exceptionHandler

Owned Association Ends

+ **exceptionHandler** : ExceptionHandler [0..*]

Package UML::Activities::ExtraStructuredActivities

Association A_handler_protectedNode

Member Ends:

handler, protectedNode

Package UML::Activities::ExtraStructuredActivities

Association A_inputElement_regionAsInput

Member Ends:

inputElement, regionAsInput

Package UML::Activities::ExtraStructuredActivities

Association A_outputElement_regionAsOutput

Member Ends:

outputElement, regionAsOutput

Package UML::Activities::FundamentalActivities

Nesting Package:

Activities

Imported Packages:

Kernel

Merged Packages:

BasicActions, BasicBehaviors

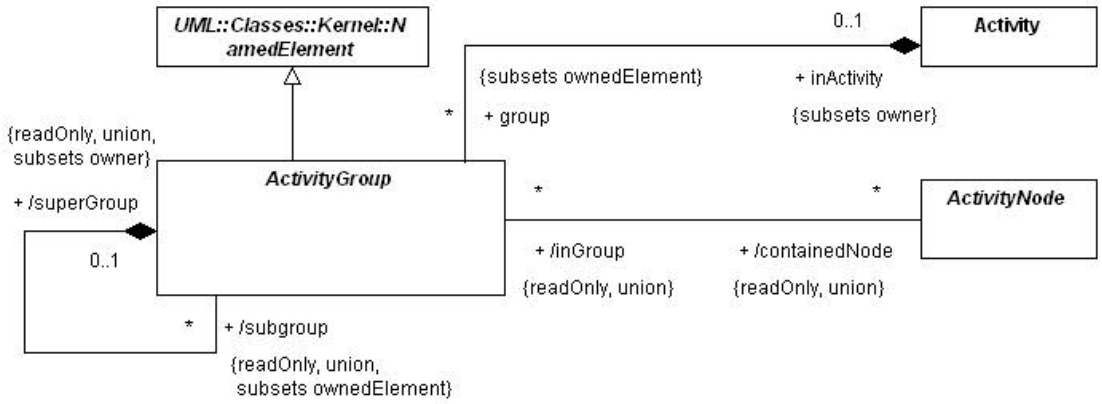
Diagram Summary
Fundamental Groups
Fundamental Nodes

Class Summary
Action
Activity
ActivityGroup
ActivityNode

Association Summary
A_containedNode_inGroup
A_group_inActivity
A_node_activity
A_subgroup_superGroup

Package UML::Activities::FundamentalActivities

Diagram Fundamental Groups



Classifiers Local to Package:

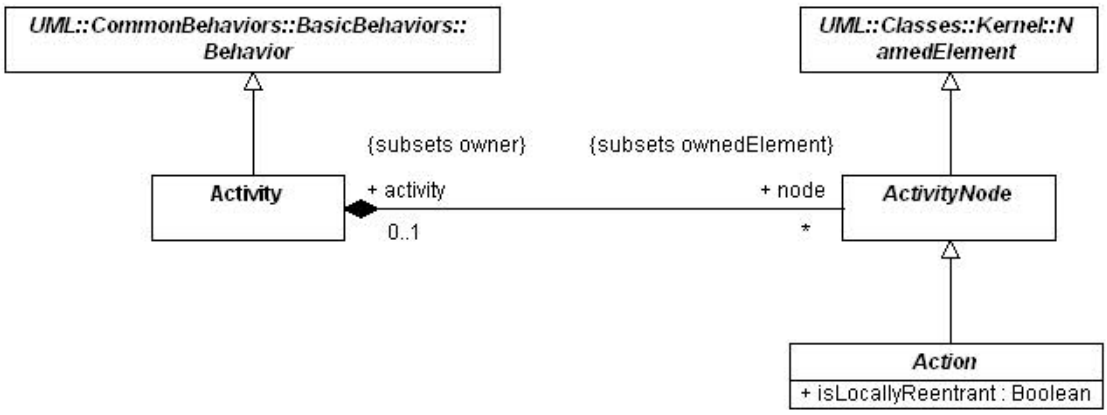
Activity, ActivityGroup, ActivityNode

Classifiers External to Package:

NamedElement

Package UML::Activities::FundamentalActivities

Diagram Fundamental Nodes



Classifiers Local to Package:

Action, Activity, ActivityNode

Classifiers External to Package:

Behavior, NamedElement

Package UML::Activities::FundamentalActivities

Class Action

An action represents a single step within an activity, that is, one that is not further decomposed within the activity.

Generalizations:

ActivityNode

Found in Diagrams:

Fundamental Nodes

Attributes

+ **isLocallyReentrant** : Boolean [1..1] = false

If true, the action can begin a new, concurrent execution, even if there is already another execution of the action ongoing. If false, the action cannot begin a new execution until any previous execution has completed.

Package UML::Activities::FundamentalActivities

Class Activity

An activity is the specification of parameterized behavior as the coordinated sequencing of subordinate units whose individual elements are actions.

Generalizations:

Behavior

Found in Diagrams:

Fundamental Groups, Fundamental Nodes

Owned Association Ends

+ **group** : ActivityGroup [0..*] {subsets ownedElement}

Top-level groups in the activity.

+ **node** : ActivityNode [0..*] {subsets ownedElement}

Nodes coordinated by the activity.

Constraints

no_supergroups

The groups of an activity have no supergroups.

expression (OCL): true

Package UML::Activities::FundamentalActivities

Class ActivityGroup

ActivityGroup is an abstract class for defining sets of nodes and edges in an activity.

Generalizations:

NamedElement

Specializations:

StructuredActivityNode

Found in Diagrams:

Fundamental Groups, Structured Activities

Owned Association Ends

+ /**containedNode** : ActivityNode [0..*] {readOnly, union}

Nodes immediately contained in the group.

+ **inActivity** : Activity [0..1] {subsets owner}

Activity containing the group.

+ /**subgroup** : ActivityGroup [0..*] {readOnly, union, subsets ownedElement}

Groups immediately contained in the group.

+ /**superGroup** : ActivityGroup [0..1] {readOnly, union, subsets owner}

Group immediately containing the group.

Package UML::Activities::FundamentalActivities

Class ActivityNode

ActivityNode is an abstract class for points in the flow of an activity connected by edges.

Generalizations:

NamedElement

Specializations:

Action

Found in Diagrams:

Fundamental Groups, Fundamental Nodes

Owned Association Ends

+ **activity** : Activity [0..1] {subsets owner}

Activity containing the node.

+ /**inGroup** : ActivityGroup [0..*] {readOnly, union}

Groups containing the node.

Package UML::Activities::FundamentalActivities

Association A_containedNode_inGroup

Member Ends:

containedNode, inGroup

Found in Diagrams:

Fundamental Groups

Package UML::Activities::FundamentalActivities

Association A_group_inActivity

Member Ends:

group, inActivity

Found in Diagrams:

Fundamental Groups

Package UML::Activities::FundamentalActivities

Association A_node_activity

Member Ends:

node, activity

Found in Diagrams:

Fundamental Nodes

Package UML::Activities::FundamentalActivities

Association A_subgroup_superGroup

Member Ends:

subgroup, superGroup

Found in Diagrams:

Fundamental Groups

Package UML::Activities::IntermediateActivities

Nesting Package:

Activities

Imported Packages:

BasicBehaviors, Kernel

Merged Packages:

BasicActivities

Diagram Summary
Activity Partitions

Class Summary
Activity
ActivityEdge
ActivityFinalNode
ActivityGroup
ActivityNode
ActivityPartition
CentralBufferNode
DecisionNode
FinalNode
FlowFinalNode
ForkNode
JoinNode
MergeNode

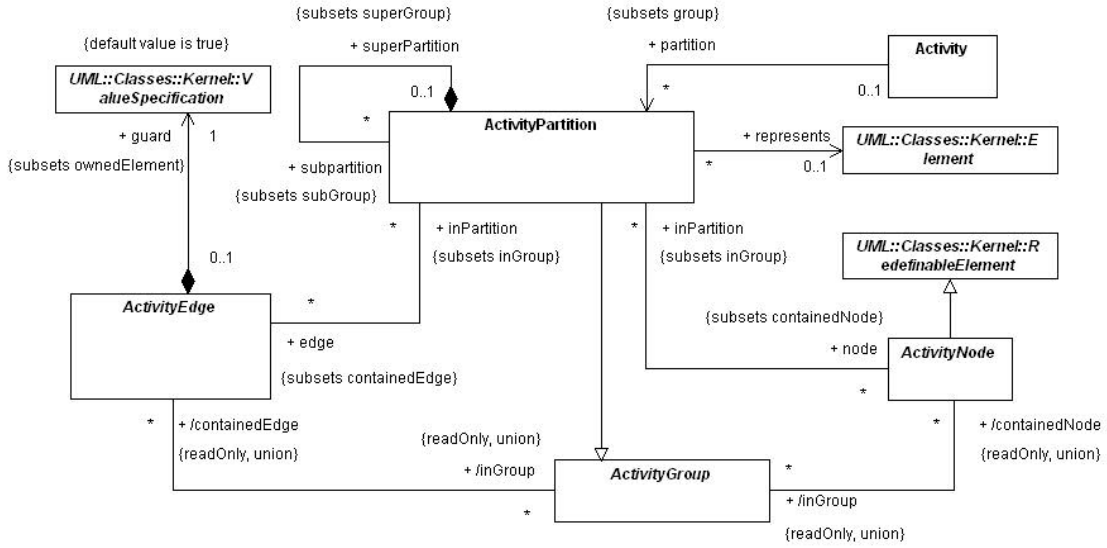
Association Summary
A_containedEdge_inGroup
A_containedNode_inGroup
A_decisionInputFlow_decisionNode
A_decisionInput_decisionNode
A_edge_inPartition
A_group_inActivity
A_guard_activityEdge
A_inPartition_node
A_partition_activity

Package UML::Activities::IntermediateActivities

A_represents_activityPartition
A_subpartition_superPartition

Package UML::Activities::IntermediateActivities

Diagram Activity Partitions



Classifiers Local to Package:

Activity, ActivityEdge, ActivityGroup, ActivityNode, ActivityPartition

Classifiers External to Package:

Element, RedefinableElement, ValueSpecification

Package UML::Activities::IntermediateActivities

Class Activity

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ **group** : ActivityGroup [0..*] {subsets ownedElement}

Top-level groups in the activity.

+ **partition** : ActivityPartition [0..*] {subsets group}

Top-level partitions in the activity.

Package UML::Activities::IntermediateActivities

Class ActivityEdge

Generalizations:

RedefinableElement

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ **guard** : ValueSpecification [1..1] {subsets ownedElement}

Specification evaluated at runtime to determine if the edge can be traversed.

+ **/inGroup** : ActivityGroup [0..*] {readOnly, union}

Groups containing the edge.

+ **inPartition** : ActivityPartition [0..*] {subsets inGroup}

Partitions containing the edge.

Package UML::Activities::IntermediateActivities

Class ActivityFinalNode

Generalizations:

FinalNode

Package UML::Activities::IntermediateActivities

Class ActivityGroup

Specializations:

ActivityPartition

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ /**containedEdge** : ActivityEdge [0..*] {readOnly, union}

Edges immediately contained in the group.

+ /**containedNode** : ActivityNode [0..*] {readOnly, union}

Nodes immediately contained in the group.

+ **inActivity** : Activity [0..1] {subsets owner}

Activity containing the group.

Package UML::Activities::IntermediateActivities

Class ActivityNode

Generalizations:

RedefinableElement

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ /**inGroup** : ActivityGroup [0..*] {readOnly, union}

Groups containing the node.

+ **inPartition** : ActivityPartition [0..*] {subsets inGroup}

Partitions containing the node.

Package UML::Activities::IntermediateActivities

Class ActivityPartition

An activity partition is a kind of activity group for identifying actions that have some characteristic in common.

Generalizations:

ActivityGroup

Found in Diagrams:

Activity Partitions

Attributes

+ **isDimension** : Boolean [1..1] = false

Tells whether the partition groups other partitions along a dimension.

+ **isExternal** : Boolean [1..1] = false

Tells whether the partition represents an entity to which the partitioning structure does not apply.

Owned Association Ends

+ **edge** : ActivityEdge [0..*] {subsets containedEdge}

Edges immediately contained in the group.

+ **node** : ActivityNode [0..*] {subsets containedNode}

Nodes immediately contained in the group.

+ **represents** : Element [0..1]

An element constraining behaviors invoked by nodes in the partition.

+ **subpartition** : ActivityPartition [0..*] {subsets subgroup}

Partitions immediately contained in the partition.

+ **superPartition** : ActivityPartition [0..1] {subsets superGroup}

Partition immediately containing the partition.

Constraints

dimension_not_contained

Package UML::Activities::IntermediateActivities

Class ActivityPartition

A partition with isDimension = true may not be contained by another partition.

expression (OCL): true

represents_classifier

If a non-external partition represents a classifier and is contained in another partition, then the containing partition must represent a classifier, and the classifier of the subpartition must be nested in the classifier represented by the containing partition, or be at the contained end of a strong composition association with the classifier represented by the containing partition.

expression (OCL): true

represents_part

If a partition represents a part, then all the non-external partitions in the same dimension and at the same level of nesting in that dimension must represent parts directly contained in the internal structure of the same classifier.

expression (OCL): true

represents_part_and_is_contained

If a partition represents a part and is contained by another partition, then the part must be of a classifier represented by the containing partition, or of a classifier that is the type of a part representing the containing partition.

expression (OCL): true

Package UML::Activities::IntermediateActivities

Class CentralBufferNode

A central buffer node is an object node for managing flows from multiple sources and destinations.

Generalizations:

ObjectNode

Specializations:

DataStoreNode

Package UML::Activities::IntermediateActivities

Class DecisionNode

A decision node is a control node that chooses between outgoing flows.

Generalizations:

ControlNode

Owned Association Ends

+ **decisionInput** : Behavior [0..1]

Provides input to guard specifications on edges outgoing from the decision node.

+ **decisionInputFlow** : ObjectFlow [0..1]

An additional edge incoming to the decision node that provides a decision input value.

Constraints

decision_input_flow_incoming

The decisionInputFlow of a decision node must be an incoming edge of the decision node.

expression (OCL): true

edges

The edges coming into and out of a decision node, other than the decision input flow (if any), must be either all object flows or all control flows.

expression (OCL): true

incoming_control_one_input_parameter

If the decision node has a decision input flow and an incoming control flow, then a decision input behavior has one input parameter whose type is the same as or a supertype of the type of object tokens offered on the decision input flow.

expression (OCL): true

incoming_object_one_input_parameter

If the decision node has no decision input flow and an incoming object flow, then a decision input behavior has one input parameter whose type is the same as or a supertype of the type of object tokens offered on the incoming edge.

expression (OCL): true

incoming_outgoing_edges

A decision node has one or two incoming edges and at least one outgoing edge.

Package UML::Activities::IntermediateActivities

Class DecisionNode

expression (OCL): true

parameters

A decision input behavior has no output parameters, no in-out parameters and one return parameter.

expression (OCL): true

two_input_parameters

If the decision node has a decision input flow and an second incoming object flow, then a decision input behavior has two input parameters, the first of which has a type that is the same as or a supertype of the type of the type of object tokens offered on the nondecision input flow and the second of which has a type that is the same as or a supertype of the type of object tokens offered on the decision input flow.

expression (OCL): true

zero_input_parameters

If the decision node has no decision input flow and an incoming control flow, then a decision input behavior has zero input parameters.

expression (OCL): true

Package UML::Activities::IntermediateActivities

Class FinalNode

A final node is an abstract control node at which a flow in an activity stops.

Generalizations:

ControlNode

Specializations:

ActivityFinalNode, FlowFinalNode

Constraints

no_outgoing_edges

A final node has no outgoing edges.

expression (OCL): true

Package UML::Activities::IntermediateActivities

Class FlowFinalNode

A flow final node is a final node that terminates a flow.

Generalizations:

FinalNode

Package UML::Activities::IntermediateActivities

Class ForkNode

A fork node is a control node that splits a flow into multiple concurrent flows.

Generalizations:

ControlNode

Constraints

edges

The edges coming into and out of a fork node must be either all object flows or all control flows.

expression (OCL): true

one_incoming_edge

A fork node has one incoming edge.

expression (OCL): true

Package UML::Activities::IntermediateActivities

Class JoinNode

A join node is a control node that synchronizes multiple flows.

Generalizations:

ControlNode

Package UML::Activities::IntermediateActivities

Class MergeNode

A merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows.

Generalizations:

ControlNode

Constraints

edges

The edges coming into and out of a merge node must be either all object flows or all control flows.

expression (OCL): true

one_outgoing_edge

A merge node has one outgoing edge.

expression (OCL): true

Package UML::Activities::IntermediateActivities

Association A_containedEdge_inGroup

Member Ends:

containedEdge, inGroup

Found in Diagrams:

Activity Partitions

Package UML::Activities::IntermediateActivities

Association A_containedNode_inGroup

Member Ends:

containedNode, inGroup

Found in Diagrams:

Activity Partitions

Package UML::Activities::IntermediateActivities

Association A_decisionInputFlow_decisionNode

Member Ends:

decisionInputFlow, decisionNode

Owned Association Ends

+ **decisionNode** : DecisionNode [0..1]

Package UML::Activities::IntermediateActivities

Association A_decisionInput_decisionNode

Member Ends:

decisionInput, decisionNode

Owned Association Ends

+ **decisionNode** : DecisionNode [0..*]

Package UML::Activities::IntermediateActivities

Association A_edge_inPartition

Member Ends:

edge, inPartition

Found in Diagrams:

Activity Partitions

Package UML::Activities::IntermediateActivities

Association A_group_inActivity

Member Ends:

group, inActivity

Package UML::Activities::IntermediateActivities

Association A_guard_activityEdge

Member Ends:

guard, activityEdge

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ **activityEdge** : ActivityEdge [0..1]

Package UML::Activities::IntermediateActivities

Association A_inPartition_node

Member Ends:

inPartition, node

Found in Diagrams:

Activity Partitions

Package UML::Activities::IntermediateActivities

Association A_partition_activity

Member Ends:

partition, activity

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ **activity** : Activity [0..1]

Package UML::Activities::IntermediateActivities

Association A_represents_activityPartition

Member Ends:

represents, activityPartition

Found in Diagrams:

Activity Partitions

Owned Association Ends

+ **activityPartition** : ActivityPartition [0..*]

Package UML::Activities::IntermediateActivities

Association A_subpartition_superPartition

Member Ends:

subpartition, superPartition

Found in Diagrams:

Activity Partitions

Package UML::Activities::StructuredActivities

Nesting Package:

Activities

Merged Packages:

FundamentalActivities

Diagram Summary
Structured Activities

Class Summary
Action
Activity
ActivityGroup
ActivityNode
Clause
ConditionalNode
ExecutableNode
LoopNode
OutputPin
SequenceNode
StructuredActivityNode
Variable

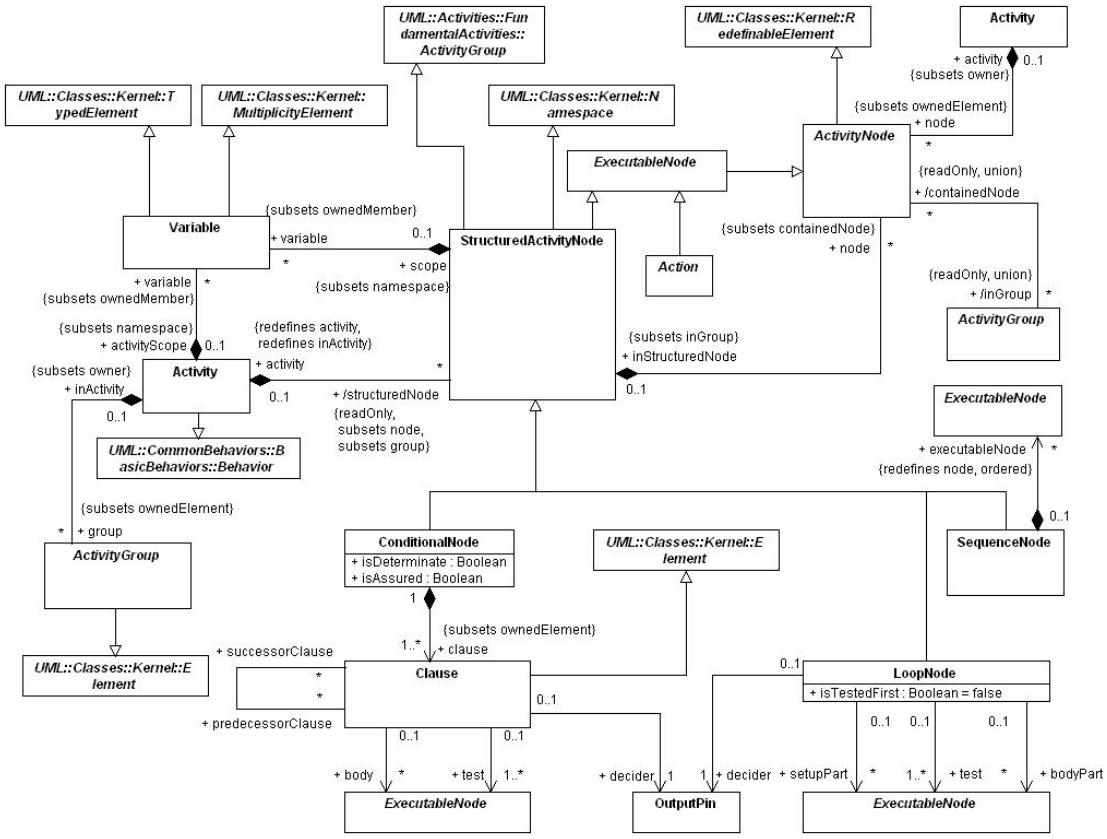
Association Summary
A_bodyPart_loopNode
A_body_clause
A_clause_conditionalNode
A_containedNode_inGroup
A_decider_clause
A_decider_loopNode
A_executableNode_sequenceNode
A_group_inActivity
A_node_activity
A_node_inStructuredNode
A_predecessorClause_successorClause
A_setupPart_loopNode
A_structuredNode_activity

Package UML::Activities::StructuredActivities

A_test_clause
A_test_loopNode
A_variable_activityScope
A_variable_scope

Package UML::Activities::StructuredActivities

Diagram Structured Activities



Classifiers Local to Package:

Action, Activity, ActivityGroup, ActivityNode, Clause, ConditionalNode, ExecutableNode, LoopNode, OutputPin, SequenceNode, StructuredActivityNode, Variable

Classifiers External to Package:

ActivityGroup, Behavior, Element, MultiplicityElement, Namespace, RedefinableElement, TypedElement

Package UML::Activities::StructuredActivities

Class Action

Generalizations:

ExecutableNode

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Class Activity

Generalizations:

Behavior

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **group** : ActivityGroup [0..*] {subsets ownedElement}

Top-level groups in the activity.

+ **node** : ActivityNode [0..*] {subsets ownedElement}

Nodes coordinated by the activity.

+ /**structuredNode** : StructuredActivityNode [0..*] {readOnly, subsets node, subsets group}

Top-level structured nodes in the activity.

+ **variable** : Variable [0..*] {subsets ownedMember}

Top-level variables in the activity.

Package UML::Activities::StructuredActivities

Class ActivityGroup

Generalizations:

Element

Found in Diagrams:

Structured Activities

Owned Association Ends

+ /**containedNode** : ActivityNode [0..*] {readOnly, union}

Nodes immediately contained in the group.

+ **inActivity** : Activity [0..1] {subsets owner}

Activity containing the group.

Package UML::Activities::StructuredActivities

Class ActivityNode

Generalizations:

RedefinableElement

Specializations:

ExecutableNode, ExecutableNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **activity** : Activity [0..1] {subsets owner}

Activity containing the node.

+ **/inGroup** : ActivityGroup [0..*] {readOnly, union}

Groups containing the node.

+ **inStructuredNode** : StructuredActivityNode [0..1] {subsets inGroup}

Structured activity node containing the node.

Constraints

owned_structured_node

Activity nodes may be owned by at most one structured node.

expression (OCL): true

Package UML::Activities::StructuredActivities

Class Clause

A clause is an element that represents a single branch of a conditional construct, including a test and a body section. The body section is executed only if (but not necessarily if) the test section evaluates true.

Generalizations:

Element

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **body** : ExecutableNode [0..*]

A nested activity fragment that is executed if the test evaluates to true and the clause is chosen over any concurrent clauses that also evaluate to true.

+ **decider** : OutputPin [1..1]

An output pin within the test fragment the value of which is examined after execution of the test to determine whether the body should be executed.

+ **predecessorClause** : Clause [0..*]

A set of clauses whose tests must all evaluate false before the current clause can be tested.

+ **successorClause** : Clause [0..*]

A set of clauses which may not be tested unless the current clause tests false.

+ **test** : ExecutableNode [1..*]

A nested activity fragment with a designated output pin that specifies the result of the test.

Constraints

decider_output

The decider output pin must be for the test body or a node contained by the test body as a structured node.

expression (OCL): true

test_and_body

The test and body parts must be disjoint.

Package UML::Activities::StructuredActivities

Class Clause

expression (OCL): true

Package UML::Activities::StructuredActivities

Class ConditionalNode

A conditional node is a structured activity node that represents an exclusive choice among some number of alternatives.

Generalizations:

StructuredActivityNode

Found in Diagrams:

Structured Activities

Attributes

+ **isAssured** : Boolean [1..1] = false

If true, the modeler asserts that at least one test will succeed.

+ **isDeterminate** : Boolean [1..1] = false

If true, the modeler asserts that at most one test will succeed.

Owned Association Ends

+ **clause** : Clause [1..*] {subsets ownedElement}

Set of clauses composing the conditional.

Constraints

clause_no_predecessor

No two clauses within a ConditionalNode maybe predecessor clauses of each other, either directly or indirectly.

expression (OCL): true

executable_nodes

The union of the ExecutabledNodes in the test and body parts of all clauses must be the same as the subset of nodes contained in the ConditionalNode (considered as a StructuredActivityNode) that are ExecutableNodes.

expression (OCL): true

one_clause_with_executable_node

No ExecutableNode may appear in the test or body part of more than one clause of a conditional node.

Package UML::Activities::StructuredActivities

Class ConditionalNode

expression (OCL): true

Package UML::Activities::StructuredActivities

Class ExecutableNode

An executable node is an abstract class for activity nodes that may be executed. It is used as an attachment point for exception handlers.

Generalizations:

ActivityNode

Specializations:

Action, StructuredActivityNode

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Class LoopNode

A loop node is a structured activity node that represents a loop with setup, test, and body sections.

Generalizations:

StructuredActivityNode

Found in Diagrams:

Structured Activities

Attributes

+ **isTestedFirst** : Boolean [1..1] = false

If true, the test is performed before the first execution of the body.
If false, the body is executed once before the test is performed.

Owned Association Ends

+ **bodyPart** : ExecutableNode [0..*]

The set of nodes and edges that perform the repetitive computations of the loop. The body section is executed as long as the test section produces a true value.

+ **decider** : OutputPin [1..1]

An output pin within the test fragment the value of which is examined after execution of the test to determine whether to execute the loop body.

+ **setupPart** : ExecutableNode [0..*]

The set of nodes and edges that initialize values or perform other setup computations for the loop.

+ **test** : ExecutableNode [1..*]

The set of nodes, edges, and designated value that compute a Boolean value to determine if another execution of the body will be performed.

Constraints

executable_nodes

The union of the ExecutableNodes in the setupPart, test and bodyPart of a LoopNode must be the same as the subset of nodes contained in the LoopNode (considered as a StructuredActivityNode) that are ExecutableNodes.

Package UML::Activities::StructuredActivities

Class LoopNode

expression (OCL): true

Package UML::Activities::StructuredActivities

Class OutputPin

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Class SequenceNode

A sequence node is a structured activity node that executes its actions in order.

Generalizations:

StructuredActivityNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **executableNode** : ExecutableNode [0..*] {ordered, redefines node}

An ordered set of executable nodes.

Package UML::Activities::StructuredActivities

Class StructuredActivityNode

A structured activity node is an executable activity node that may have an expansion into subordinate nodes as an activity group. The subordinate nodes must belong to only one structured activity node, although they may be nested.

Generalizations:

ActivityGroup, ExecutableNode, Namespace

Specializations:

ConditionalNode, ExpansionRegion, LoopNode, SequenceNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **activity** : Activity [0..1] {redefines activity, redefines inActivity}

Activity immediately containing the node.

+ **node** : ActivityNode [0..*] {subsets containedNode}

Nodes immediately contained in the group.

+ **variable** : Variable [0..*] {subsets ownedMember}

A variable defined in the scope of the structured activity node. It has no value and may not be accessed

Package UML::Activities::StructuredActivities

Class Variable

Variables are elements for passing data between actions indirectly. A local variable stores values shared by the actions within a structured activity group but not accessible outside it. The output of an action may be written to a variable and read for the input to a subsequent action, which is effectively an indirect data flow path. Because there is no predefined relationship between actions that read and write variables, these actions must be sequenced by control flows to prevent race conditions that may occur between actions that read or write the same variable.

Generalizations:

MultiplicityElement, TypedElement

Found in Diagrams:

Structured Activities, Variable Actions

Owned Association Ends

+ **activityScope** : Activity [0..1] {subsets namespace}

An activity that owns the variable.

+ **scope** : StructuredActivityNode [0..1] {subsets namespace}

A structured activity node that owns the variable.

Operations

+ **isAccessibleBy** (a : Action) : Boolean [1..1] {query}

The isAccessibleBy() operation is not defined in standard UML. Implementations should define it to specify which actions can access a variable.

body (OCL): result = true

Constraints

owned

A variable is owned by a StructuredNode or Activity, but not both.

expression (OCL): true

Package UML::Activities::StructuredActivities

Association A_bodyPart_loopNode

Member Ends:

bodyPart, loopNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::StructuredActivities

Association A_body_clause

Member Ends:

body, clause

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **clause** : Clause [0..1]

Package UML::Activities::StructuredActivities

Association A_clause_conditionalNode

Member Ends:

clause, conditionalNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **conditionalNode** : ConditionalNode [1..1]

Package UML::Activities::StructuredActivities

Association A_containedNode_inGroup

Member Ends:

containedNode, inGroup

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_decider_clause

Member Ends:

decider, clause

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **clause** : Clause [0..1]

Package UML::Activities::StructuredActivities

Association A_decider_loopNode

Member Ends:

decider, loopNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::StructuredActivities

Association A_executableNode_sequenceNode

Member Ends:

executableNode, sequenceNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **sequenceNode** : SequenceNode [0..1]

Package UML::Activities::StructuredActivities

Association A_group_inActivity

Member Ends:

group, inActivity

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_node_activity

Member Ends:

node, activity

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_node_inStructuredNode

Member Ends:

node, inStructuredNode

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_predecessorClause_successorClause

Member Ends:

predecessorClause, successorClause

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_setupPart_loopNode

Member Ends:

setupPart, loopNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::StructuredActivities

Association A_structuredNode_activity

Member Ends:

structuredNode, activity

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_test_clause

Member Ends:

test, clause

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **clause** : Clause [0..1]

Package UML::Activities::StructuredActivities

Association A_test_loopNode

Member Ends:

test, loopNode

Found in Diagrams:

Structured Activities

Owned Association Ends

+ **loopNode** : LoopNode [0..1]

Package UML::Activities::StructuredActivities

Association A_variable_activityScope

Member Ends:

variable, activityScope

Found in Diagrams:

Structured Activities

Package UML::Activities::StructuredActivities

Association A_variable_scope

Member Ends:

variable, scope

Found in Diagrams:

Structured Activities

Package UML::AuxiliaryConstructs

Nesting Package:

UML

Imported Packages:

Classes, Dependencies, InternalStructures, Kernel

Nested Package Summary
InformationFlows
Models
Profiles
Templates

Package UML::AuxiliaryConstructs::InformationFlows

Nesting Package:

AuxiliaryConstructs

Imported Packages:

BasicActivities, BasicInteractions, InternalStructures

Merged Packages:

Kernel

Class Summary
InformationFlow
InformationItem

Association Summary
A_conveyed_informationFlow
A_informationSource_informationFlow
A_informationTarget_informationFlow
A_realization_abstraction
A_realizingActivityEdge_informationFlow
A_realizingConnector_informationFlow
A_realizingMessage_informationFlow
A_represented_representation

Package UML::AuxiliaryConstructs::InformationFlows

Class InformationFlow

An information flow specifies that one or more information items circulates from its sources to its targets. Information flows require some kind of information channel for transmitting information items from the source to the destination. An information channel is represented in various ways depending on the nature of its sources and targets. It may be represented by connectors, links, associations, or even dependencies. For example, if the source and destination are parts in some composite structure such as a collaboration, then the information channel is likely to be represented by a connector between them. Or, if the source and target are objects (which are a kind of instance specification), they may be represented by a link that joins the two, and so on.

Generalizations:

DirectedRelationship, PackageableElement

Owned Association Ends

+ **conveyed** : Classifier [1..*]

Specifies the information items that may circulate on this information flow.

+ **informationSource** : NamedElement [1..*] {subsets source}

Defines from which source the conveyed InformationItems are initiated.

+ **informationTarget** : NamedElement [1..*] {subsets target}

Defines to which target the conveyed InformationItems are directed.

+ **realization** : Relationship [0..*]

Determines which Relationship will realize the specified flow

+ **realizingActivityEdge** : ActivityEdge [0..*]

Determines which ActivityEdges will realize the specified flow.

+ **realizingConnector** : Connector [0..*]

Determines which Connectors will realize the specified flow.

+ **realizingMessage** : Message [0..*]

Determines which Messages will realize the specified flow.

Constraints

Package UML::AuxiliaryConstructs::InformationFlows

Class InformationFlow

convey_classifiers

An information flow can only convey classifiers that are allowed to represent an information item.

expression (OCL): self.conveyed.represented->forAll(p | p->oclIsKindOf(Class) or oclIsKindOf (Interface) or oclIsKindOf(InformationItem) or oclIsKindOf(Signal) or oclIsKindOf(Component))

must_conform

The sources and targets of the information flow must conform with the sources and targets or conversely the targets and sources of the realization relationships.

expression (OCL): true

sources_and_targets_kind

The sources and targets of the information flow can only be one of the following kind: Actor, Node, UseCase, Artifact, Class, Component, Port, Property, Interface, Package, ActivityNode, ActivityPartition and InstanceSpecification except when its classifier is a relationship (i.e. it represents a link).

expression (OCL): (self.informationSource->forAll(p | p->oclIsKindOf(Actor) or oclIsKindOf (Node) or oclIsKindOf(UseCase) or oclIsKindOf(Artifact) or oclIsKindOf(Class) or oclIsKindOf (Component) or oclIsKindOf(Port) or oclIsKindOf(Property) or oclIsKindOf(Interface) or oclIsKindOf(Package) or oclIsKindOf(ActivityNode) or oclIsKindOf(ActivityPartition) or oclIsKindOf(InstanceSpecification))) and (self.informationTarget->forAll(p | p->oclIsKindOf (Actor) or oclIsKindOf(Node) or oclIsKindOf(UseCase) or oclIsKindOf(Artifact) or oclIsKindOf (Class) or oclIsKindOf(Component) or oclIsKindOf(Port) or oclIsKindOf(Property) or oclIsKindOf (Interface) or oclIsKindOf(Package) or oclIsKindOf(ActivityNode) or oclIsKindOf (ActivityPartition) or oclIsKindOf(InstanceSpecification)))

Package UML::AuxiliaryConstructs::InformationFlows

Class InformationItem

An information item is an abstraction of all kinds of information that can be exchanged between objects. It is a kind of classifier intended for representing information in a very abstract way, one which cannot be instantiated.

Generalizations:

Classifier

Owned Association Ends

+ **represented** : Classifier [0..*]

Determines the classifiers that will specify the structure and nature of the information. An information item represents all its represented classifiers.

Constraints

has_no

An informationItem has no feature, no generalization, and no associations.

expression (OCL): self.generalization->isEmpty() and self.feature->isEmpty()

not_instantiable

It is not instantiable.

expression (OCL): isAbstract

sources_and_targets

The sources and targets of an information item (its related information flows) must designate subsets of the sources and targets of the representation information item, if any.The Classifiers that can realize an information item can only be of the following kind: Class, Interface, InformationItem, Signal, Component.

expression (OCL): (self.represented->select(p | p->oclIsKindOf(InformationItem))->forAll(p | p.informationFlow.source->forAll(q | self.informationFlow.source->include(q)) and p.informationFlow.target->forAll(q | self.informationFlow.target->include(q)))) and (self.represented->forAll(p | p->oclIsKindOf(Class) or oclIsKindOf(Interface) or oclIsKindOf(InformationItem) or oclIsKindOf(Signal) or oclIsKindOf(Component)))

Package UML::AuxiliaryConstructs::InformationFlows

Association A_conveyed_informationFlow

Member Ends:

conveyed, informationFlow

Owned Association Ends

+ **informationFlow** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_informationSource_informationFlow

Member Ends:

informationSource, informationFlow

Owned Association Ends

+ **informationFlow** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_informationTarget_informationFlow

Member Ends:

informationTarget, informationFlow

Owned Association Ends

+ **informationFlow** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_realization_abstraction

Member Ends:

realization, abstraction

Owned Association Ends

+ **abstraction** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_realizingActivityEdge_informationFlow

Member Ends:

realizingActivityEdge, informationFlow

Owned Association Ends

+ **informationFlow** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_realizingConnector_informationFlow

Member Ends:

realizingConnector, informationFlow

Owned Association Ends

+ **informationFlow** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_realizingMessage_informationFlow

Member Ends:

realizingMessage, informationFlow

Owned Association Ends

+ **informationFlow** : InformationFlow [0..*]

Package UML::AuxiliaryConstructs::InformationFlows

Association A_represented_representation

Member Ends:

represented, representation

Owned Association Ends

+ **representation** : InformationItem [0..*]

Package UML::AuxiliaryConstructs::Models

Nesting Package:

AuxiliaryConstructs

Merged Packages:

Kernel

Class Summary
Model

Package UML::AuxiliaryConstructs::Models

Class Model

A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose. This purpose determines what is to be included in the model and what is irrelevant. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail.

Generalizations:

Package

Attributes

+ **viewpoint** : String [0..1]

The name of the viewpoint that is expressed by a model (This name may refer to a profile definition).

Package UML::AuxiliaryConstructs::Profiles

Nesting Package:

AuxiliaryConstructs

Merged Packages:

Profiles

Class Summary
ExtensionEnd

Package UML::AuxiliaryConstructs::Profiles

Class ExtensionEnd

The default multiplicity of an extension end is 0..1.

Attributes

+ /**lower** : Integer [0..1] = 0

Package UML::AuxiliaryConstructs::Templates

Nesting Package:

AuxiliaryConstructs

Merged Packages:

InternalStructures

Diagram Summary
Classifier Templates

Class Summary
Classifier
ClassifierTemplateParameter
ConnectableElement
ConnectableElementTemplateParameter
NamedElement
Operation
OperationTemplateParameter
Package
PackageableElement
ParameterableElement
Property
RedefinableTemplateSignature
StringExpression
TemplateBinding
TemplateParameter
TemplateParameterSubstitution
TemplateSignature
TemplateableElement
ValueSpecification

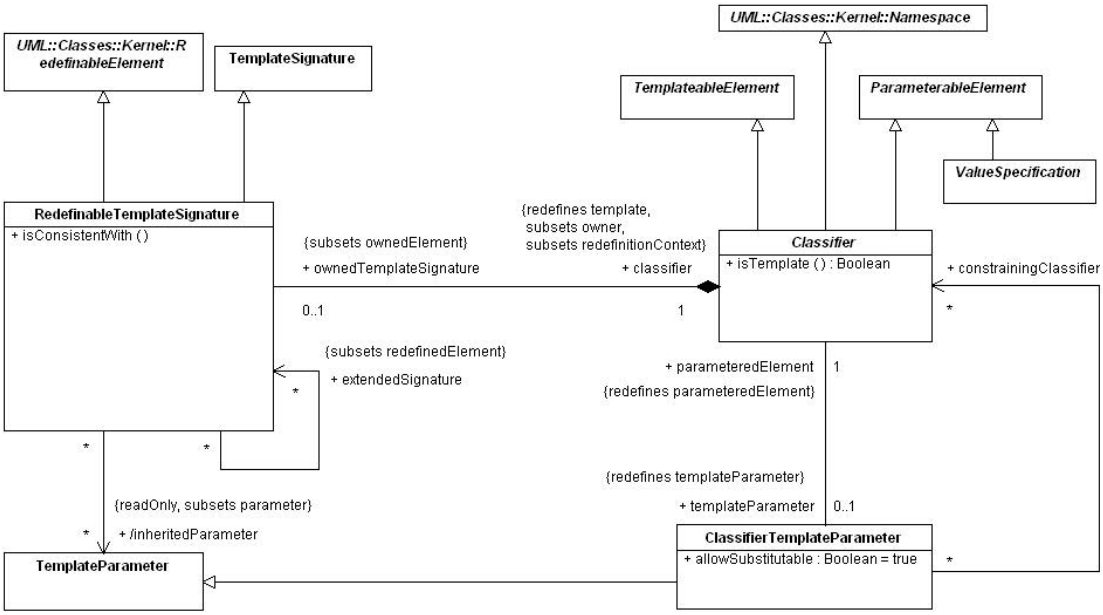
Association Summary
A_actual_templateParameterSubstitution
A_classifier_templateParameter_parameteredElement
A_connectableElement_templateParameter_parameteredElement
A_constrainingClassifier_classifierTemplateParameter
A_default_templateParameter
A_extendedSignature_redefinableTemplateSignature

Package UML::AuxiliaryConstructs::Templates

A_formal_templateParameterSubstitution
A_inheritedParameter_redefinableTemplateSignature
A_nameExpression_namedElement
A_operation_templateParameter_parameteredElement
A_ownedActual_templateParameterSubstitution
A_ownedDefault_templateParameter
A_ownedParameter_signature
A_ownedParameteredElement_owningTemplateParameter
A_ownedTemplateSignature_classifier
A_ownedTemplateSignature_template
A_parameterSubstitution_templateBinding
A_parameter_templateSignature
A_parameteredElement_templateParameter
A_signature_templateBinding
A_subExpression_owningExpression
A_templateBinding_boundElement

Package UML::AuxiliaryConstructs::Templates

Diagram Classifier Templates



Classifiers Local to Package:

Classifier, ClassifierTemplateParameter, ParameterableElement, RedefinableTemplateSignature, TemplateParameter, TemplateSignature, TemplateableElement, ValueSpecification

Classifiers External to Package:

Namespace, RedefinableElement

Package UML::AuxiliaryConstructs::Templates

Class Classifier

Classifier is defined to be a kind of templateable element so that a classifier can be parameterized. It is also defined to be a kind of parameterable element so that a classifier can be a formal template parameter.

Generalizations:

Namespace, ParameterableElement, TemplateableElement

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **ownedTemplateSignature** : RedefinableTemplateSignature [0..1] {subsets ownedElement, redefines ownedTemplateSignature}

The optional template signature specifying the formal template parameters.

+ **templateParameter** : ClassifierTemplateParameter [0..1] {redefines templateParameter}

The template parameter that exposes this element as a formal parameter.

Operations

+ **isTemplate** () : Boolean [1..1] {query}

The query isTemplate() returns whether this templateable element is actually a template.

body (OCL): result = oclAsType(TemplatableElement).isTemplate() or general->exists(g | g.isTemplate())

Package UML::AuxiliaryConstructs::Templates

Class ClassifierTemplateParameter

A classifier template parameter exposes a classifier as a formal template parameter.

Generalizations:

TemplateParameter

Found in Diagrams:

Classifier Templates

Attributes

+ **allowSubstitutable** : Boolean [1..1] = true

Constrains the required relationship between an actual parameter and the parameteredElement for this formal parameter.

Owned Association Ends

+ **constrainingClassifier** : Classifier [0..*]

The classifiers that constrain the argument that can be used for the parameter. If the allowSubstitutable attribute is true, then any classifier that is compatible with this constraining classifier can be substituted; otherwise, it must be either this classifier or one of its subclasses. If this property is empty, there are no constraints on the classifier that can be used as an argument.

+ **parameteredElement** : Classifier [1..1] {redefines parameteredElement}

The parameterable classifier for this template parameter.

Constraints

has_constraining_classifier

If "allowSubstitutable" is true, then there must be a constrainingClassifier.

expression (OCL): allowSubstitutable implies constrainingClassifier->notEmpty()

Package UML::AuxiliaryConstructs::Templates

Class ConnectableElement

A connectable element may be exposed as a connectable element template parameter.

Generalizations:

ParameterableElement

Owned Association Ends

+ **templateParameter** : ConnectableElementTemplateParameter [0..1] {redefines templateParameter}

The ConnectableElementTemplateParameter for this ConnectableElement parameter.

Package UML::AuxiliaryConstructs::Templates

Class ConnectableElementTemplateParameter

A connectable element template parameter exposes a connectable element as a formal parameter for a template.

Generalizations:

TemplateParameter

Owned Association Ends

+ **parameteredElement** : ConnectableElement [1..1] {redefines parameteredElement}

The ConnectableElement for this template parameter.

Package UML::AuxiliaryConstructs::Templates

Class NamedElement

A named element supports using a string expression to specify its name. This allows names of model elements to involve template parameters. The actual name is evaluated from the string expression only when it is sensible to do so (e.g., when a template is bound).

Generalizations:

Element

Owned Association Ends

+ **nameExpression** : StringExpression [0..1] {subsets ownedElement}

The string expression used to define the name of this named element.

Package UML::AuxiliaryConstructs::Templates

Class Operation

Operation specializes TemplateableElement in order to support specification of template operations and bound operations. Operation specializes ParameterableElement to specify that an operation can be exposed as a formal template parameter, and provided as an actual parameter in a binding of a template.

Generalizations:

ParameterableElement, TemplateableElement

Owned Association Ends

+ **templateParameter** : OperationTemplateParameter [0..1] {redefines templateParameter}

The template parameter that exposes this element as a formal parameter.

Package UML::AuxiliaryConstructs::Templates

Class OperationTemplateParameter

An operation template parameter exposes an operation as a formal parameter for a template.

Generalizations:

TemplateParameter

Owned Association Ends

+ **parameteredElement** : Operation [1..1] {redefines parameteredElement}

The operation for this template parameter.

Package UML::AuxiliaryConstructs::Templates

Class Package

Package specializes TemplateableElement and PackageableElement specializes ParameterableElement to specify that a package can be used as a template and a PackageableElement as a template parameter.

Generalizations:

TemplateableElement

Package UML::AuxiliaryConstructs::Templates

Class PackageableElement

Packageable elements are able to serve as a template parameter.

Generalizations:

ParameterableElement

Package UML::AuxiliaryConstructs::Templates

Class ParameterableElement

A parameterable element is an element that can be exposed as a formal template parameter for a template, or specified as an actual parameter in a binding of a template.

Generalizations:

Element

Specializations:

Classifier, ConnectableElement, Operation, PackageableElement, Property, ValueSpecification

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **owningTemplateParameter** : TemplateParameter [0..1] {subsets owner, subsets templateParameter}

The formal template parameter that owns this element.

+ **templateParameter** : TemplateParameter [0..1]

The template parameter that exposes this element as a formal parameter.

Operations

+ **isCompatibleWith** (p : ParameterableElement) : Boolean [1..1] {query}

The query isCompatibleWith() determines if this parameterable element is compatible with the specified parameterable element. By default parameterable element P is compatible with parameterable element Q if the kind of P is the same or a subtype as the kind of Q. Subclasses should override this operation to specify different compatibility constraints.

body (OCL): result = p->oclIsKindOf(self.oclType)

+ **isTemplateParameter** () : Boolean [1..1] {query}

The query isTemplateParameter() determines if this parameterable element is exposed as a formal template parameter.

body (OCL): result = templateParameter->notEmpty()

Package UML::AuxiliaryConstructs::Templates

Class Property

Property specializes ParameterableElement to specify that a property can be exposed as a formal template parameter, and provided as an actual parameter in a binding of a template.

Generalizations:

ParameterableElement

Operations

+ **isCompatibleWith** (p : ParameterableElement) : Boolean [1..1] {query}

The query isCompatibleWith() determines if this parameterable element is compatible with the specified parameterable element. By default parameterable element P is compatible with parameterable element Q if the kind of P is the same or a subtype as the kind of Q. In addition, for properties, the type must be conformant with the type of the specified parameterable element.

body (OCL): result = p->oclIsKindOf(self.oclType) and self.type.conformsTo(p.oclAsType (TypedElement).type)

Constraints

binding_to_attribute

A binding of a property template parameter representing an attribute must be to an attribute.

expression (OCL): (isAttribute(self) and (templateParameterSubstitution->notEmpty()) implies (templateParameterSubstitution->forAll(ts | isAttribute(ts.formal)))

Package UML::AuxiliaryConstructs::Templates

Class RedefinableTemplateSignature

A redefinable template signature supports the addition of formal template parameters in a specialization of a template classifier.

Generalizations:

RedefinableElement, TemplateSignature

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **classifier** : Classifier [1..1] {subsets redefinitionContext, subsets owner, redefines template}

The classifier that owns this template signature.

+ **extendedSignature** : RedefinableTemplateSignature [0..*] {subsets redefinedElement}

The template signature that is extended by this template signature.

+ /**inheritedParameter** : TemplateParameter [0..*] {readOnly, subsets parameter}

The formal template parameters of the extendedSignature.

Operations

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two RedefinableTemplateSignatures in a context in which redefinition is possible, whether redefinition would be logically consistent. A redefining template signature is always consistent with a redefined template signature, since redefinition only adds new formal parameters.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = redefinee.oclIsKindOf(RedefineableTemplateSignature)

Constraints

inherited_parameters

The inherited parameters are the parameters of the extended template signature.

expression (OCL): if extendedSignature->isEmpty() then Set{} else extendedSignature.parameter endif

Package UML::AuxiliaryConstructs::Templates

Class StringExpression

An expression that specifies a string value that is derived by concatenating a set of sub string expressions, some of which might be template parameters.

Generalizations:

Expression, TemplateableElement

Owned Association Ends

+ **owningExpression** : StringExpression [0..1] {subsets owner}

The string expression of which this expression is a substring.

+ **subExpression** : StringExpression [0..*] {subsets ownedElement}

The StringExpressions that constitute this StringExpression.

Operations

+ **stringValue** () : String [1..1] {query}

The query stringValue() returns the string that concatenates, in order, all the component string literals of all the subexpressions that are part of the StringExpression.

body (OCL): result = if subExpression->notEmpty() then subExpression->iterate(se; stringValue = '| stringValue.concat(se.stringValue())) else operand->iterate()(op; stringValue = ' | stringValue. concat(op.value))

Constraints

operands

All the operands of a StringExpression must be LiteralStrings

expression (OCL): operand->forAll (op | op.oclIsKindOf (LiteralString))

subexpressions

If a StringExpression has sub-expressions, it cannot have operands and vice versa (this avoids the problem of having to
define a collating sequence between operands and subexpressions).

expression (OCL): if subExpression->notEmpty() then operand->isEmpty() else operand-> notEmpty()

Package UML::AuxiliaryConstructs::Templates

Class TemplateBinding

A template binding represents a relationship between a templateable element and a template. A template binding specifies the substitutions of actual parameters for the formal parameters of the template.

Generalizations:

DirectedRelationship

Owned Association Ends

+ **boundElement** : TemplateableElement [1..1] {subsets owner, subsets source}

The element that is bound by this binding.

+ **parameterSubstitution** : TemplateParameterSubstitution [0..*] {subsets ownedElement}

The parameter substitutions owned by this template binding.

+ **signature** : TemplateSignature [1..1] {subsets target}

The template signature for the template that is the target of the binding.

Constraints

one_parameter_substitution

A binding contains at most one parameter substitution for each formal template parameter of the target template signature.

expression (OCL): template.parameter->forAll(p | parameterSubstitution->select(b | b.formal = p) ->size() <= 1)

parameter_substitution_formal

Each parameter substitution must refer to a formal template parameter of the target template signature.

expression (OCL): parameterSubstitution->forAll(b | template.parameter->includes(b.formal))

Package UML::AuxiliaryConstructs::Templates

Class TemplateParameter

A template parameter exposes a parameterable element as a formal template parameter of a template.

Generalizations:

Element

Specializations:

ClassifierTemplateParameter, ConnectableElementTemplateParameter, OperationTemplateParameter

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **default** : ParameterableElement [0..1]

The element that is the default for this formal template parameter.

+ **ownedDefault** : ParameterableElement [0..1] {subsets default, subsets ownedElement}

The element that is owned by this template parameter for the purpose of providing a default.

+ **ownedParameteredElement** : ParameterableElement [0..1] {subsets ownedElement, subsets parameteredElement}

The element that is owned by this template parameter.

+ **parameteredElement** : ParameterableElement [1..1]

The element exposed by this template parameter.

+ **signature** : TemplateSignature [1..1] {subsets owner}

The template signature that owns this template parameter.

Constraints

must_be_compatible

The default must be compatible with the formal template parameter.

expression (OCL): default->notEmpty() implies default->isCompatibleWith(parameteredElement)

Package UML::AuxiliaryConstructs::Templates

Class TemplateParameterSubstitution

A template parameter substitution relates the actual parameter to a formal template parameter as part of a template binding.

Generalizations:

Element

Owned Association Ends

+ **actual** : ParameterableElement [1..1]

The element that is the actual parameter for this substitution.

+ **formal** : TemplateParameter [1..1]

The formal template parameter that is associated with this substitution.

+ **ownedActual** : ParameterableElement [0..1] {subsets actual, subsets ownedElement}

The actual parameter that is owned by this substitution.

+ **templateBinding** : TemplateBinding [1..1] {subsets owner}

The optional bindings from this element to templates.

Constraints

must_be_compatible

The actual parameter must be compatible with the formal template parameter, e.g. the actual parameter for a class template parameter must be a class.

expression (OCL): actual->forAll(a | a.isCompatibleWith(formal.parameteredElement))

Package UML::AuxiliaryConstructs::Templates

Class TemplateSignature

A template signature bundles the set of formal template parameters for a templated element.

Generalizations:

Element

Specializations:

RedefinableTemplateSignature

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **ownedParameter** : TemplateParameter [0..*] {ordered, subsets ownedElement, subsets parameter }

The formal template parameters that are owned by this template signature.

+ **parameter** : TemplateParameter [1..*] {ordered}

The ordered set of all formal template parameters for this template signature.

+ **template** : TemplateableElement [1..1] {subsets owner}

The element that owns this template signature.

Constraints

own_elements

Parameters must own the elements they parameter or those elements must be owned by the element being templated.

expression (OCL): templatedElement.ownedElement->includesAll(parameter.parameteredElement - parameter.ownedParameteredElement)

Package UML::AuxiliaryConstructs::Templates

Class TemplateableElement

A templateable element is an element that can optionally be defined as a template and bound to other templates.

Generalizations:

Element

Specializations:

Classifier, Operation, Package, StringExpression

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **ownedTemplateSignature** : TemplateSignature [0..1] {subsets ownedElement}

The optional template signature specifying the formal template parameters.

+ **templateBinding** : TemplateBinding [0..*] {subsets ownedElement}

The optional bindings from this element to templates.

Operations

+ **isTemplate** () : Boolean [1..1] {query}

The query isTemplate() returns whether this templateable element is actually a template.

body (OCL): result = ownedTemplateSignature->notEmpty()

+ **parameterableElements** () : ParameterableElement [0..*] {query}

The query parameterableElements() returns the set of elements that may be used as the parametered elements for a template parameter of this templateable element. By default, this set includes all the owned elements. Subclasses may override this operation if they choose to restrict the set of parameterable elements.

body (OCL): result = allOwnedElements->select(oclIsKindOf(ParameterableElement))

Package UML::AuxiliaryConstructs::Templates

Class ValueSpecification

ValueSpecification specializes ParameterableElement to specify that a value specification can be exposed as a formal template parameter, and provided as an actual parameter in a binding of a template.

Generalizations:

ParameterableElement

Found in Diagrams:

Classifier Templates

Operations

+ **isCompatibleWith** (p : ParameterableElement) : Boolean [1..1] {query}

The query isCompatibleWith() determines if this parameterable element is compatible with the specified parameterable element. By default parameterable element P is compatible with parameterable element Q if the kind of P is the same or a subtype as the kind of Q. In addition, for ValueSpecification, the type must be conformant with the type of the specified parameterable element.

body (OCL): result = p->oclIsKindOf(self.oclType) and self.type.conformsTo(p.oclAsType (TypedElement).type)

Package UML::AuxiliaryConstructs::Templates

Association A_actual_templateParameterSubstitution

Member Ends:

actual, templateParameterSubstitution

Owned Association Ends

+ **templateParameterSubstitution** : TemplateParameterSubstitution [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_classifier_templateParameter_parameteredElement

Member Ends:

templateParameter, parameteredElement

Found in Diagrams:

Classifier Templates

Package UML::AuxiliaryConstructs::Templates

Association A_connectableElement_templateParameter_parameteredElement

Member Ends:

templateParameter, parameteredElement

Package UML::AuxiliaryConstructs::Templates

Association A_constrainingClassifier_classifierTemplateParameter

Member Ends:

constrainingClassifier, classifierTemplateParameter

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **classifierTemplateParameter** : ClassifierTemplateParameter [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_default_templateParameter

Member Ends:

default, templateParameter

Owned Association Ends

+ **templateParameter** : TemplateParameter [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_extendedSignature_redefinableTemplateSignature

Member Ends:

extendedSignature, redefinableTemplateSignature

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **redefinableTemplateSignature** : RedefinableTemplateSignature [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_formal_templateParameterSubstitution

Member Ends:

formal, templateParameterSubstitution

Owned Association Ends

+ **templateParameterSubstitution** : TemplateParameterSubstitution [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_inheritedParameter_redefinableTemplateSignature

Member Ends:

inheritedParameter, redefinableTemplateSignature

Found in Diagrams:

Classifier Templates

Owned Association Ends

+ **redefinableTemplateSignature** : RedefinableTemplateSignature [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_nameExpression_namedElement

Member Ends:

nameExpression, namedElement

Owned Association Ends

+ **namedElement** : NamedElement [0..1]

Package UML::AuxiliaryConstructs::Templates

Association A_operation_templateParameter_parameteredElement

Member Ends:

templateParameter, parameteredElement

Package UML::AuxiliaryConstructs::Templates

Association A_ownedActual_templateParameterSubstitution

Member Ends:

ownedActual, templateParameterSubstitution

Owned Association Ends

+ **templateParameterSubstitution** : TemplateParameterSubstitution [0..1]

Package UML::AuxiliaryConstructs::Templates

Association A_ownedDefault_templateParameter

Member Ends:

ownedDefault, templateParameter

Owned Association Ends

+ **templateParameter** : TemplateParameter [0..1]

Package UML::AuxiliaryConstructs::Templates

Association A_ownedParameter_signature

Member Ends:

ownedParameter, signature

Package UML::AuxiliaryConstructs::Templates

Association A_ownedParameteredElement_owningTemplateParameter

Member Ends:

ownedParameteredElement, owningTemplateParameter

Package UML::AuxiliaryConstructs::Templates

Association A_ownedTemplateSignature_classifier

Member Ends:

ownedTemplateSignature, classifier

Found in Diagrams:

Classifier Templates

Package UML::AuxiliaryConstructs::Templates

Association A_ownedTemplateSignature_template

Member Ends:

ownedTemplateSignature, template

Package UML::AuxiliaryConstructs::Templates

Association A_parameterSubstitution_templateBinding

Member Ends:

parameterSubstitution, templateBinding

Package UML::AuxiliaryConstructs::Templates

Association A_parameter_templateSignature

Member Ends:

parameter, templateSignature

Owned Association Ends

+ **templateSignature** : TemplateSignature [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_parameteredElement_templateParameter

Member Ends:

parameteredElement, templateParameter

Package UML::AuxiliaryConstructs::Templates

Association A_signature_templateBinding

Member Ends:

signature, templateBinding

Owned Association Ends

+ **templateBinding** : TemplateBinding [0..*]

Package UML::AuxiliaryConstructs::Templates

Association A_subExpression_owningExpression

Member Ends:

subExpression, owningExpression

Package UML::AuxiliaryConstructs::Templates

Association A_templateBinding_boundElement

Member Ends:

templateBinding, boundElement

Package UML::Classes

Nesting Package:

UML

Imported Packages:

AuxiliaryConstructs, CommonBehaviors

Nested Package Summary
AssociationClasses
Dependencies
Interfaces
Kernel
PowerTypes

Package UML::Classes::AssociationClasses

Nesting Package:

Classes

Merged Packages:

Kernel

Class Summary
AssociationClass
Property

Association Summary
A_qualifier_associationEnd

Package UML::Classes::AssociationClasses

Class AssociationClass

A model element that has both association and class properties. An AssociationClass can be seen as an association that also has class properties, or as a class that also has association properties. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not to any of the classifiers.

Generalizations:

Association, Class

Constraints

cannot_be_defined

An AssociationClass cannot be defined between itself and something else.

expression (OCL): self.endType->excludes(self) and self.endType>collect(et|et.allparents()->excludes(self))

disjoint_attributes_ends

The owned attributes and owned ends of an AssociationClass are disjoint

expression (OCL): ownedAttribute->intersection(ownedEnd)->isEmpty()

Package UML::Classes::AssociationClasses

Class Property

Property represents a declared state of one or more instances in terms of a named relationship to a value or values. When a property is an attribute of a classifier, the value or values are related to the instance of the classifier by being held in slots of the instance. When a property is an association end, the value or values are related to the instance or instances at the other end(s) of the association. The range of valid values represented by the property can be controlled by setting the property's type.

Generalizations:

StructuralFeature

Owned Association Ends

+ **associationEnd** : Property [0..1] {subsets owner}

Designates the optional association end that owns a qualifier attribute.

+ **qualifier** : Property [0..*] {ordered, subsets ownedElement}

An optional list of ordered qualifier attributes for the end. If the list is empty, then the Association is not qualified.

Package UML::Classes::AssociationClasses

Association A_qualifier_associationEnd

Member Ends:

qualifier, associationEnd

Package UML::Classes::Dependencies

Nesting Package:

Classes

Imported Packages:

Kernel

Merged Packages:

Kernel

Class Summary
Abstraction
Classifier
Dependency
NamedElement
Namespace
PackageableElement
Realization
Substitution
Usage

Association Summary
A_clientDependency_client
A_contract_substitution
A_mapping_abstraction
A_ownedMember_namespace
A_substitution_substitutingClassifier
A_supplier_supplierDependency

Package UML::Classes::Dependencies

Class Abstraction

An abstraction is a relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints.

Generalizations:

Dependency

Specializations:

Manifestation, Realization

Owned Association Ends

+ **mapping** : OpaqueExpression [0..1] {subsets ownedElement}

An composition of an Expression that states the abstraction relationship between the supplier and the client. In some cases, such as Derivation, it is usually formal and unidirectional; in other cases, such as Trace, it is usually informal and bidirectional. The mapping expression is optional and may be omitted if the precise relationship between the elements is not specified.

Package UML::Classes::Dependencies

Class Classifier

Generalizations:

NamedElement, Namespace

Owned Association Ends

+ **substitution** : Substitution [0..*] {subsets ownedElement, subsets clientDependency}

References the substitutions that are owned by this Classifier.

Package UML::Classes::Dependencies

Class Dependency

A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).

Generalizations:

DirectedRelationship, PackageableElement

Specializations:

Abstraction, Deployment, Deployment, Usage

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **client** : NamedElement [1..*] {subsets source}

The element(s) dependent on the supplier element(s). In some cases (such as a Trace Abstraction) the assignment of direction (that is, the designation of the client element) is at the discretion of the modeler, and is a stipulation.

+ **supplier** : NamedElement [1..*] {subsets target}

The element(s) independent of the client element(s), in the same respect and the same dependency relationship. In some directed dependency relationships (such as Refinement Abstractions), a common convention in the domain of class-based OO software is to put the more abstract element in this role. Despite this convention, users of UML may stipulate a sense of dependency suitable for their domain, which makes a more abstract element dependent on that which is more specific.

Package UML::Classes::Dependencies

Class NamedElement

Generalizations:

Element

Specializations:

Artifact, BehavioredClassifier, Classifier, Component, DeployedArtifact, DeploymentTarget, Namespace, PackageableElement

Found in Diagrams:

Component Construct, Messages

Owned Association Ends

+ **clientDependency** : Dependency [0..*]

Indicates the dependencies that reference the client.

+ /**namespace** : Namespace [0..1] {readOnly}

Specifies the namespace that owns the NamedElement.

Package UML::Classes::Dependencies

Class Namespace

Generalizations:

NamedElement

Specializations:

Classifier

Owned Association Ends

+ /**ownedMember** : NamedElement [0..*] {readOnly}

A collection of NamedElements owned by the Namespace.

Package UML::Classes::Dependencies

Class PackageableElement

Generalizations:

NamedElement

Specializations:

Dependency

Package UML::Classes::Dependencies

Class Realization

Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client). Realization can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, etc.

Generalizations:

Abstraction

Specializations:

ComponentRealization, InterfaceRealization, Substitution

Found in Diagrams:

Component Construct, Interfaces

Package UML::Classes::Dependencies

Class Substitution

A substitution is a relationship between two classifiers signifies that the substituting classifier complies with the contract specified by the contract classifier. This implies that instances of the substituting classifier are runtime substitutable where instances of the contract classifier are expected.

Generalizations:

Realization

Owned Association Ends

+ **contract** : Classifier [1..1] {subsets supplier}

The contract with which the substituting classifier complies.

+ **substitutingClassifier** : Classifier [1..1] {subsets client}

Instances of the substituting classifier are runtime substitutable where instances of the contract classifier are expected.

Package UML::Classes::Dependencies

Class Usage

A usage is a relationship in which one element requires another element (or set of elements) for its full implementation or operation. A usage is a dependency in which the client requires the presence of the supplier.

Generalizations:

Dependency

Package UML::Classes::Dependencies

Association A_clientDependency_client

Member Ends:

clientDependency, client

Package UML::Classes::Dependencies

Association A_contract_substitution

Member Ends:

contract, substitution

Owned Association Ends

+ **substitution** : Substitution [0..*]

Package UML::Classes::Dependencies

Association A_mapping_abstraction

Member Ends:

mapping, abstraction

Owned Association Ends

+ **abstraction** : Abstraction [0..1]

Package UML::Classes::Dependencies

Association A_ownedMember_namespace

Member Ends:

ownedMember, namespace

Package UML::Classes::Dependencies

Association A_substitution_substitutingClassifier

Member Ends:

substitution, substitutingClassifier

Package UML::Classes::Dependencies

Association A_supplier_supplierDependency

Member Ends:

supplier, supplierDependency

Owned Association Ends

+ **supplierDependency** : Dependency [0..*]

Indicates the dependencies that reference the supplier.

Package UML::Classes::Interfaces

Nesting Package:

Classes

Merged Packages:

BasicBehaviors, Dependencies

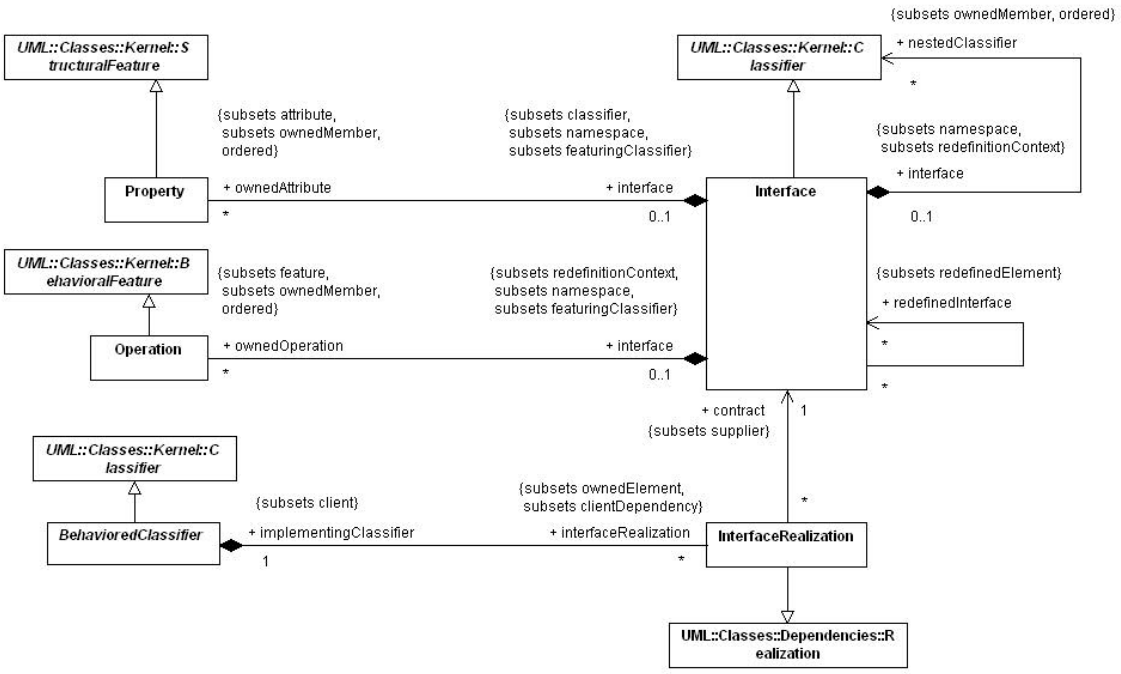
Diagram Summary
Interfaces

Class Summary
BehavioredClassifier
Interface
InterfaceRealization
Operation
Property

Association Summary
A_contract_interfaceRealization
A_interfaceRealization_implementingClassifier
A_nestedClassifier_interface
A_ownedAttribute_interface
A_ownedOperation_interface
A_redefinedInterface_interface

Package UML::Classes::Interfaces

Diagram Interfaces



Classifiers Local to Package:

BehavioredClassifier, Interface, InterfaceRealization, Operation, Property

Classifiers External to Package:

BehavioralFeature, Classifier, Realization, StructuralFeature

Package UML::Classes::Interfaces

Class BehavioredClassifier

A behaviored classifier may have an interface realization.

Generalizations:

Classifier, NamedElement

Found in Diagrams:

Interfaces

Owned Association Ends

+ **interfaceRealization** : InterfaceRealization [0..*] {subsets ownedElement, subsets clientDependency}

The set of InterfaceRealizations owned by the BehavioredClassifier. Interface realizations reference the Interfaces of which the BehavioredClassifier is an implementation.

Package UML::Classes::Interfaces

Class Interface

An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract. The obligations that may be associated with an interface are in the form of various kinds of constraints (such as pre- and post-conditions) or protocol specifications, which may impose ordering restrictions on interactions through the interface.

Generalizations:

Classifier

Found in Diagrams:

Component Construct, Interfaces, The port metaclass

Owned Association Ends

+ **nestedClassifier** : Classifier [0..*] {ordered, subsets ownedMember}

References all the Classifiers that are defined (nested) within the Class.

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The attributes (i.e. the properties) owned by the class.

+ **ownedOperation** : Operation [0..*] {ordered, subsets feature, subsets ownedMember}

The operations owned by the class.

+ **redefinedInterface** : Interface [0..*] {subsets redefinedElement}

References all the Interfaces redefined by this Interface.

Constraints

visibility

The visibility of all features owned by an interface must be public.

expression (OCL): self.feature->forAll(f | f.visibility = #public)

Package UML::Classes::Interfaces

Class InterfaceRealization

An interface realization is a specialized realization relationship between a classifier and an interface. This relationship signifies that the realizing classifier conforms to the contract specified by the interface.

Generalizations:

Realization

Found in Diagrams:

Interfaces

Owned Association Ends

+ **contract** : Interface [1..1] {subsets supplier}

References the Interface specifying the conformance contract.

+ **implementingClassifier** : BehavioredClassifier [1..1] {subsets client}

References the BehavioredClassifier that owns this Interfacerealization (i.e., the classifier that realizes the Interface to which it points).

Package UML::Classes::Interfaces

Class Operation

Generalizations:

BehavioralFeature

Found in Diagrams:

Interfaces

Owned Association Ends

+ **interface** : Interface [0..1] {subsets redefinitionContext, subsets featuringClassifier, subsets namespace}

The Interface that owns this Operation.

Package UML::Classes::Interfaces

Class Property

Generalizations:

StructuralFeature

Found in Diagrams:

Interfaces

Owned Association Ends

+ **interface** : Interface [0..1] {subsets classifier, subsets namespace, subsets featuringClassifier}

Package UML::Classes::Interfaces

Association A_contract_interfaceRealization

Member Ends:

contract, interfaceRealization

Found in Diagrams:

Interfaces

Owned Association Ends

+ **interfaceRealization** : InterfaceRealization [0..*]

Package UML::Classes::Interfaces

Association A_interfaceRealization_implementingClassifier

Member Ends:

interfaceRealization, implementingClassifier

Found in Diagrams:

Interfaces

Package UML::Classes::Interfaces

Association A_nestedClassifier_interface

Member Ends:

nestedClassifier, interface

Found in Diagrams:

Interfaces

Owned Association Ends

+ **interface** : Interface [0..1] {subsets namespace, subsets redefinitionContext}

Package UML::Classes::Interfaces

Association A_ownedAttribute_interface

Member Ends:

ownedAttribute, interface

Found in Diagrams:

Interfaces

Package UML::Classes::Interfaces

Association A_ownedOperation_interface

Member Ends:

ownedOperation, interface

Found in Diagrams:

Interfaces

Package UML::Classes::Interfaces

Association A_redefinedInterface_interface

Member Ends:

redefinedInterface, interface

Found in Diagrams:

Interfaces

Owned Association Ends

+ **interface** : Interface [0..*]

Package UML::Classes::Kernel

Nesting Package:

Classes

Merged Packages:

Constructs, PrimitiveTypes

Diagram Summary
Classifiers
Expression

Class Summary
Association
BehavioralFeature
Class
Classifier
Comment
Constraint
DataType
DirectedRelationship
Element
ElementImport
Enumeration
EnumerationLiteral
Expression
Feature
Generalization
InstanceSpecification
InstanceValue
LiteralBoolean
LiteralInteger
LiteralNull
LiteralSpecification
LiteralString
LiteralUnlimitedNatural
MultiplicityElement
NamedElement
Namespace

Package UML::Classes::Kernel

OpaqueExpression
Operation
Package
PackageImport
PackageMerge
PackageableElement
Parameter
PrimitiveType
Property
RedefinableElement
Relationship
Slot
StructuralFeature
Type
TypedElement
ValueSpecification

Enumeration Summary
AggregationKind
ParameterDirectionKind
VisibilityKind

Association Summary
A_annotatedElement_comment
A_attribute_classifier
A_bodyCondition_bodyContext
A_classifier_instanceSpecification
A_constrainedElement_constraint
A_defaultValue_owningParameter
A_defaultValue_owningProperty
A_definingFeature_slot
A_elementImport_importingNamespace
A_endType_association
A_feature_featuringClassifier
A_general_classifier
A_general_generalization
A_generalization_specific

Package UML::Classes::Kernel

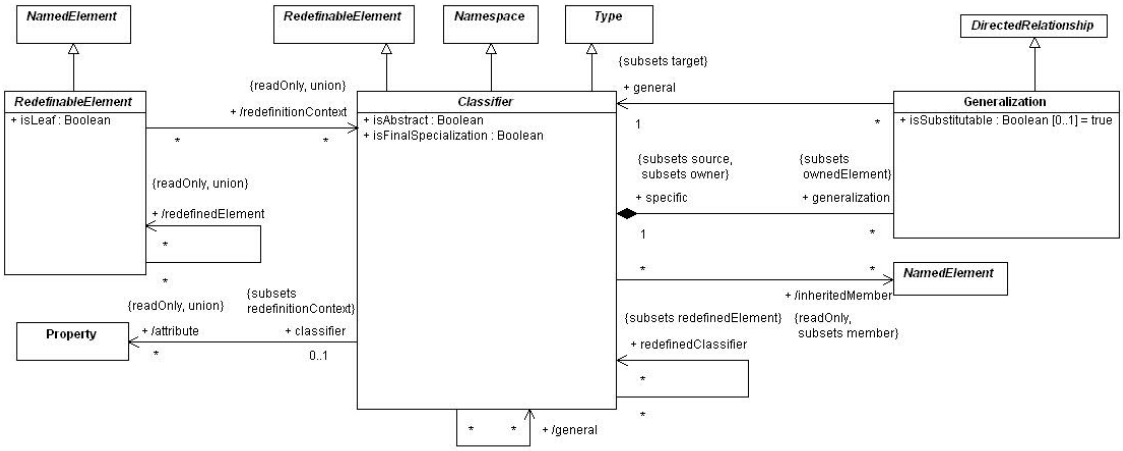
A_importedElement_elementImport
A_importedMember_namespace
A_importedPackage_packageImport
A_inheritedMember_classifier
A_instance_instanceValue
A_lowerValue_owningLower
A_memberEnd_association
A_member_namespace
A_mergedPackage_packageMerge
A_navigableOwnedEnd_association
A_nestedClassifier_class
A_nestedPackage_nestingPackage
A_operand_expression
A_opposite_property
A_ownedAttribute_class
A_ownedAttribute_datatype
A_ownedComment_owningElement
A_ownedElement_owner
A_ownedEnd_owningAssociation
A_ownedLiteral_enumeration
A_ownedMember_namespace
A_ownedOperation_class
A_ownedOperation_datatype
A_ownedParameter_operation
A_ownedParameter_ownerFormalParam
A_ownedRule_context
A_ownedType_package
A_packageImport_importingNamespace
A_packageMerge_receivingPackage
A_packagedElement_owningPackage
A_postcondition_postContext
A_precondition_preContext
A_raisedException_behavioralFeature
A_raisedException_operation
A_redefinedClassifier_classifier
A_redefinedElement_redefinableElement
A_redefinedOperation_operation

Package UML::Classes::Kernel

A_redefinedProperty_property
A_redefinitionContext_redefinableElement
A_relatedElement_relationship
A_slot_owningInstance
A_source_directedRelationship
A_specification_owningConstraint
A_specification_owningInstanceSpec
A_subsettedProperty_property
A_superClass_class
A_target_directedRelationship
A_type_operation
A_type_typedElement
A_upperValue_owningUpper
A_value_owningSlot

Package UML::Classes::Kernel

Diagram Classifiers

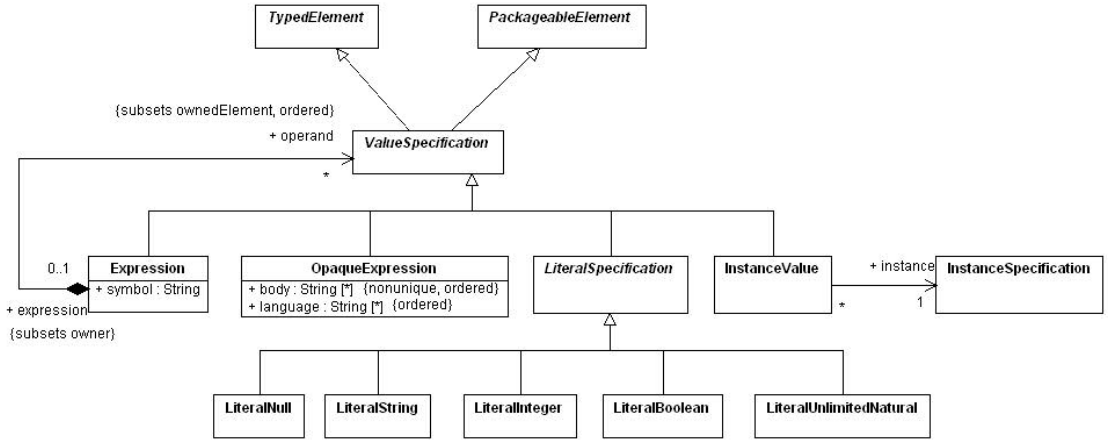


Classifiers Local to Package:

Classifier, DirectedRelationship, Generalization, NamedElement, Namespace, Property, RedefinableElement, Type

Package UML::Classes::Kernel

Diagram Expression



Classifiers Local to Package:

Expression, InstanceSpecification, InstanceValue, LiteralBoolean, LiteralInteger, LiteralNull, LiteralSpecification, LiteralString, LiteralUnlimitedNatural, OpaqueExpression, PackageableElement, TypedElement, ValueSpecification

Package UML::Classes::Kernel

Class Association

An association describes a set of tuples whose values refer to typed instances. An instance of an association is called a link. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.

Generalizations:

Classifier, Relationship

Specializations:

AssociationClass, CommunicationPath

Attributes

+ **isDerived** : Boolean [1..1] = false

Specifies whether the association is derived from other model elements such as other associations or constraints.

Owned Association Ends

+ /**endType** : Type [1..*] {ordered, readOnly, subsets relatedElement}

References the classifiers that are used as types of the ends of the association.

+ **memberEnd** : Property [2..*] {ordered, subsets member}

Each end represents participation of instances of the classifier connected to the end in links of the association.

+ **navigableOwnedEnd** : Property [0..*] {subsets ownedEnd}

The navigable ends that are owned by the association itself.

+ **ownedEnd** : Property [0..*] {ordered, subsets memberEnd, subsets feature, subsets ownedMember }

The ends that are owned by the association itself.

Operations

+ **endType** () : Type [0..*] {ordered, query}

endType is derived from the types of the member ends.

body (OCL): result = self.memberEnd->collect(e | e.type)

Package UML::Classes::Kernel

Class Association

Constraints

association_ends

Association ends of associations with more than two ends must be owned by the association.

expression (OCL): if memberEnd->size() > 2 then ownedEnd->includesAll(memberEnd)

binary_associations

Only binary associations can be aggregations.

expression (OCL): self.memberEnd->exists(aggregation <> Aggregation::none) implies self.memberEnd->size() = 2

specialized_end_number

An association specializing another association has the same number of ends as the other association.

expression (OCL): parents()->select(oclIsKindOf(Association)).oclAsType(Association)->forAll(p | p.memberEnd->size() = self.memberEnd->size())

specialized_end_types

When an association specializes another association, every end of the specific association corresponds to an end of the general association, and the specific end reaches the same type or a subtype of the more general end.

expression (OCL): Sequence{1..self.memberEnd->size()}-> forAll(i | self.general->select (oclIsKindOf(Association)).oclAsType(Association)-> forAll(ga |self.memberEnd->at(i).type. conformsTo(ga.memberEnd->at(i).type)))

Package UML::Classes::Kernel

Class BehavioralFeature

A behavioral feature is a feature of a classifier that specifies an aspect of the behavior of its instances.

Generalizations:

Feature, Namespace

Specializations:

Operation, Operation

Found in Diagrams:

Interfaces

Owned Association Ends

+ **ownedParameter** : Parameter [0..*] {ordered, subsets ownedMember}

Specifies the ordered set of formal parameters of this BehavioralFeature.

+ **raisedException** : Type [0..*]

References the Types representing exceptions that may be raised during an invocation of this feature.

Operations

+ **isDistinguishableFrom** (n : NamedElement, ns : Namespace) : Boolean [1..1] {query}

The query isDistinguishableFrom() determines whether two BehavioralFeatures may coexist in the same Namespace. It specifies that they have to have different signatures.

body (OCL): result = if n.oclIsKindOf(BehavioralFeature) then if ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->notEmpty() then Set{}->including(self)->including(n)->isUnique(bf | bf.ownedParameter->collect(type)) else true endif else true endif

Package UML::Classes::Kernel

Class Class

A class describes a set of objects that share the same specifications of features, constraints, and semantics.

Generalizations:

Classifier

Specializations:

AssociationClass, Behavior, Behavior, Component

Found in Diagrams:

Common Behavior, Component Construct

Attributes

+ **isAbstract** : Boolean [1..1] = false {redefines isAbstract}

If true, the Classifier does not provide a complete declaration and can typically not be instantiated. An abstract classifier is intended to be used by other classifiers e.g. as the target of general metarelationships or generalization relationships.

Owned Association Ends

+ **nestedClassifier** : Classifier [0..*] {ordered, subsets ownedMember}

References all the Classifiers that are defined (nested) within the Class.

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The attributes (i.e. the properties) owned by the class.

+ **ownedOperation** : Operation [0..*] {ordered, subsets feature, subsets ownedMember}

The operations owned by the class.

+ /**superClass** : Class [0..*] {redefines general}

This gives the superclasses of a class.

Operations

+ **inherit** (inhs : NamedElement [0..*]) : NamedElement [0..*] {query}

The inherit operation is overridden to exclude redefined properties.

Package UML::Classes::Kernel

Class Class

body (OCL): result = inhs->excluding(inh | ownedMember->select(oclIsKindOf (RedefinableElement))->select(redefinedElement->includes(inh)))

Package UML::Classes::Kernel

Class Classifier

A classifier is a classification of instances - it describes a set of instances that have features in common. A classifier can specify a generalization hierarchy by referencing its general classifiers.

Generalizations:

Namespace, RedefinableElement, Type

Specializations:

Artifact, Association, BehavioredClassifier, BehavioredClassifier, BehavioredClassifier, Class, DataType, InformationItem, Interface, Interface, Interface, Signal

Found in Diagrams:

Basic Actions, Classifiers, Common Behavior, Component Construct, Interfaces, Reception

Attributes

+ **isAbstract** : Boolean [1..1] = false

If true, the Classifier does not provide a complete declaration and can typically not be instantiated. An abstract classifier is intended to be used by other classifiers e.g. as the target of general metarelationships or generalization relationships.

+ **isFinalSpecialization** : Boolean [1..1] = false

If true, the Classifier cannot be specialized by generalization. Note that this property is preserved through package merge operations; that is, the capability to specialize a Classifier (i.e., isFinalSpecialization =false) must be preserved in the resulting Classifier of a package merge operation where a Classifier with isFinalSpecialization =false is merged with a matching Classifier with isFinalSpecialization =true: the resulting Classifier will have isFinalSpecialization =false.

Owned Association Ends

+ /**attribute** : Property [0..*] {readOnly, union, subsets feature}

Refers to all of the Properties that are direct (i.e. not inherited or imported) attributes of the classifier.

+ /**feature** : Feature [0..*] {readOnly, union}

Specifies each feature defined in the classifier.

+ /**general** : Classifier [0..*]

Specifies the general Classifiers for this Classifier.

Package UML::Classes::Kernel

Class Classifier

+ **generalization** : Generalization [0..*] {subsets ownedElement}

Specifies the Generalization relationships for this Classifier. These Generalizations navigaten to more general classifiers in the generalization hierarchy.

+ /**inheritedMember** : NamedElement [0..*] {readOnly, subsets member}

Specifies all elements inherited by this classifier from the general classifiers.

+ **redefinedClassifier** : Classifier [0..*] {subsets redefinedElement}

References the Classifiers that are redefined by this Classifier.

Operations

+ **allFeatures** () : Feature [0..*] {query}

The query allFeatures() gives all of the features in the namespace of the classifier. In general, through mechanisms such as inheritance, this will be a larger set than feature.

body (OCL): result = member->select(oclIsKindOf(Feature))

+ **allParents** () : Classifier [0..*] {query}

The query allParents() gives all of the direct and indirect ancestors of a generalized Classifier.

body (OCL): result = self.parents()->union(self.parents()->collect(p | p.allParents())

+ **conformsTo** (other : Classifier) : Boolean [1..1] {query}

The query conformsTo() gives true for a classifier that defines a type that conforms to another. This is used, for example, in the specification of signature conformance for operations.

body (OCL): result = (self=other) or (self.allParents()->includes(other))

+ **general** () : Classifier [0..*] {query}

The general classifiers are the classifiers referenced by the generalization relationships.

body (OCL): result = self.parents()

+ **hasVisibilityOf** (n : NamedElement) : Boolean [1..1] {query}

The query hasVisibilityOf() determines whether a named element is visible in the classifier. By default all are visible. It is only called when the argument is something owned by a parent.

precondition (): self.allParents()->collect(c | c.member)->includes(n)

body (OCL): result = if (self.inheritedMember->includes(n)) then (n.visibility <> #private) else true

Package UML::Classes::Kernel

Class Classifier

+ **inherit** (inhs : NamedElement [0..*]) : NamedElement [0..*] {query}

The query inherit() defines how to inherit a set of elements. Here the operation is defined to inherit them all. It is intended to be redefined in circumstances where inheritance is affected by redefinition.

body (OCL): result = inhs

+ **inheritableMembers** (c : Classifier) : NamedElement [0..*] {query}

The query inheritableMembers() gives all of the members of a classifier that may be inherited in one of its descendants, subject to whatever visibility restrictions apply.

precondition (OCL): c.allParents()->includes(self)

body (OCL): result = member->select(m | c.hasVisibilityOf(m))

postcondition (OCL): c.allParents()->includes(self)

+ **inheritedMember** () : NamedElement [0..*] {query}

The inheritedMember association is derived by inheriting the inheritable members of the parents.

body (OCL): result = self.inherit(self.parents()->collect(p | p.inheritableMembers(self))

+ **maySpecializeType** (c : Classifier) : Boolean [1..1] {query}

The query maySpecializeType() determines whether this classifier may have a generalization relationship to classifiers of the specified type. By default a classifier may specialize classifiers of the same or a more general type. It is intended to be redefined by classifiers that have different specialization constraints.

body (OCL): result = self.oclIsKindOf(c.oclType)

+ **parents** () : Classifier [0..*] {query}

The query parents() gives all of the immediate ancestors of a generalized Classifier.

body (OCL): result = generalization.general

Constraints

generalization_hierarchies

Generalization hierarchies must be directed and acyclical. A classifier can not be both a transitively general and transitively specific classifier of the same classifier.

expression (OCL): not self.allParents()->includes(self)

non_final_parents

The parents of a classifier must be non-final.

Package UML::Classes::Kernel

Class Classifier

expression (OCL): self.parents()->forAll(not isFinalSpecialization)

specialize_type

A classifier may only specialize classifiers of a valid type.

expression (OCL): self.parents()->forAll(c | self.maySpecializeType(c))

Package UML::Classes::Kernel

Class Comment

A comment is a textual annotation that can be attached to a set of elements.

Generalizations:

Element

Attributes

+ **body** : String [0..1]

Specifies a string that is the comment.

Owned Association Ends

+ **annotatedElement** : Element [0..*]

References the Element(s) being commented.

Package UML::Classes::Kernel

Class Constraint

A constraint is a condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.

Generalizations:

PackageableElement

Specializations:

InteractionConstraint, IntervalConstraint

Found in Diagrams:

Interactions, Simple Time

Owned Association Ends

+ **constrainedElement** : Element [0..*] {ordered}

The ordered set of Elements referenced by this Constraint.

+ **context** : Namespace [0..1] {subsets namespace}

Specifies the namespace that owns the NamedElement.

+ **specification** : ValueSpecification [1..1] {subsets ownedElement}

A condition that must be true when evaluated in order for the constraint to be satisfied.

Constraints

boolean_value

The value specification for a constraint must evaluate to a Boolean value.

expression (OCL): true

no_side_effects

Evaluating the value specification for a constraint must not have side effects.

expression (OCL): true

not_applied_to_self

A constraint cannot be applied to itself.

expression (OCL): not constrainedElement->includes(self)

Package UML::Classes::Kernel

Class DataType

A data type is a type whose instances are identified only by their value. A data type may contain attributes to support the modeling of structured data types.

Generalizations:

Classifier

Specializations:

Enumeration, PrimitiveType

Owned Association Ends

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The Attributes owned by the DataType.

+ **ownedOperation** : Operation [0..*] {ordered, subsets feature, subsets ownedMember}

The Operations owned by the DataType.

Package UML::Classes::Kernel

Class DirectedRelationship

A directed relationship represents a relationship between a collection of source model elements and a collection of target model elements.

Generalizations:

Relationship

Specializations:

Dependency, ElementImport, Extend, Generalization, Include, InformationFlow, PackageImport, PackageMerge, ProtocolConformance, TemplateBinding

Found in Diagrams:

Classifiers

Owned Association Ends

+ /**source** : Element [1..*] {readOnly, union, subsets relatedElement}

Specifies the sources of the DirectedRelationship.

+ /**target** : Element [1..*] {readOnly, union, subsets relatedElement}

Specifies the targets of the DirectedRelationship.

Package UML::Classes::Kernel

Class Element

An element is a constituent of a model. As such, it has the capability of owning other elements.

Specializations:

ActivityGroup, Clause, Comment, ExceptionHandler, LinkEndData, MultiplicityElement, NamedElement, NamedElement, NamedElement, ParameterableElement, QualifierValue, Relationship, Slot, TemplateParameter, TemplateParameterSubstitution, TemplateSignature, TemplateableElement

Found in Diagrams:

Activity Partitions, Structured Activities

Owned Association Ends

+ **ownedComment** : Comment [0..*] {subsets ownedElement}

The Comments owned by this element.

+ /**ownedElement** : Element [0..*] {readOnly, union}

The Elements owned by this element.

+ /**owner** : Element [0..1] {readOnly, union}

The Element that owns this element.

Operations

+ **allOwnedElements** () : Element [0..*] {query}

The query allOwnedElements() gives all of the direct and indirect owned elements of an element.

body (OCL): result = ownedElement->union(ownedElement->collect(e | e.allOwnedElements()))

+ **mustBeOwned** () : Boolean [1..1] {query}

The query mustBeOwned() indicates whether elements of this type must have an owner. Subclasses of Element that do not require an owner must override this operation.

body (OCL): result = true

Constraints

has_owner

Elements that must be owned must have an owner.

expression (OCL): self.mustBeOwned() implies owner->notEmpty()

Package UML::Classes::Kernel

Class Element

not_own_self

An element may not directly or indirectly own itself.

expression (OCL): not self.allOwnedElements()->includes(self)

Package UML::Classes::Kernel

Class ElementImport

An element import identifies an element in another package, and allows the element to be referenced using its name without a qualifier.

Generalizations:

DirectedRelationship

Attributes

+ **alias** : String [0..1]

Specifies the name that should be added to the namespace of the importing package in lieu of the name of the imported packagable element. The aliased name must not clash with any other member name in the importing package. By default, no alias is used.

+ **visibility** : VisibilityKind [1..1] = public

Specifies the visibility of the imported PackageableElement within the importing Package. The default visibility is the same as that of the imported element. If the imported element does not have a visibility, it is possible to add visibility to the element import.

Owned Association Ends

+ **importedElement** : PackageableElement [1..1] {subsets target}

Specifies the PackageableElement whose name is to be added to a Namespace.

+ **importingNamespace** : Namespace [1..1] {subsets source, subsets owner}

Specifies the Namespace that imports a PackageableElement from another Package.

Operations

+ **getName** () : String [1..1] {query}

The query getName() returns the name under which the imported PackageableElement will be known in the importing namespace.

body (OCL): result = if self.alias->notEmpty() then self.alias else self.importedElement.name endif

Constraints

imported_element_is_public

An importedElement has either public visibility or no visibility at all.

Package UML::Classes::Kernel

Class ElementImport

expression (OCL): self.importedElement.visibility.notEmpty() implies self.importedElement.visibility = #public

visibility_public_or_private

The visibility of an ElementImport is either public or private.

expression (OCL): self.visibility = #public or self.visibility = #private

Package UML::Classes::Kernel

Class Enumeration

An enumeration is a data type whose values are enumerated in the model as enumeration literals.

Generalizations:

DataType

Owned Association Ends

+ **ownedLiteral** : EnumerationLiteral [0..*] {ordered, subsets ownedMember}

The ordered set of literals for this Enumeration.

Package UML::Classes::Kernel

Class EnumerationLiteral

An enumeration literal is a user-defined data value for an enumeration.

Generalizations:

InstanceSpecification

Owned Association Ends

+ **enumeration** : Enumeration [0..1] {subsets namespace}

The Enumeration that this EnumerationLiteral is a member of.

Package UML::Classes::Kernel

Class Expression

An expression represents a node in an expression tree, which may be non-terminal or terminal. It defines a symbol, and has a possibly empty sequence of operands which are value specifications.

Generalizations:

ValueSpecification

Specializations:

StringExpression

Found in Diagrams:

Expression

Attributes

+ **symbol** : String [0..1]

The symbol associated with the node in the expression tree.

Owned Association Ends

+ **operand** : ValueSpecification [0..*] {ordered, subsets ownedElement}

Specifies a sequence of operands.

Package UML::Classes::Kernel

Class Feature

A feature declares a behavioral or structural characteristic of instances of classifiers.

Generalizations:

RedefinableElement

Specializations:

BehavioralFeature, BehavioralFeature, Connector, StructuralFeature

Found in Diagrams:

Structured Classifier

Attributes

+ **isStatic** : Boolean [1..1] = false

Specifies whether this feature characterizes individual instances classified by the classifier (false) or the classifier itself (true).

Owned Association Ends

+ /**featuringClassifier** : Classifier [0..*] {readOnly, union}

The Classifiers that have this Feature as a feature.

Package UML::Classes::Kernel

Class Generalization

A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.

Generalizations:

DirectedRelationship

Found in Diagrams:

Classifiers

Attributes

+ **isSubstitutable** : Boolean [0..1] = true

Indicates whether the specific classifier can be used wherever the general classifier can be used. If true, the execution traces of the specific classifier will be a superset of the execution traces of the general classifier.

Owned Association Ends

+ **general** : Classifier [1..1] {subsets target}

References the general classifier in the Generalization relationship.

+ **specific** : Classifier [1..1] {subsets source, subsets owner}

References the specializing classifier in the Generalization relationship.

Package UML::Classes::Kernel

Class InstanceSpecification

An instance specification is a model element that represents an instance in a modeled system.

Generalizations:

PackageableElement

Specializations:

EnumerationLiteral

Found in Diagrams:

Expression

Owned Association Ends

+ **classifier** : Classifier [0..*]

The classifier or classifiers of the represented instance. If multiple classifiers are specified, the instance is classified by all of them.

+ **slot** : Slot [0..*] {subsets ownedElement}

A slot giving the value or values of a structural feature of the instance. An instance specification can have one slot per structural feature of its classifiers, including inherited features. It is not necessary to model a slot for each structural feature, in which case the instance specification is a partial description.

+ **specification** : ValueSpecification [0..1] {subsets ownedElement}

A specification of how to compute, derive, or construct the instance.

Constraints

defining_feature

The defining feature of each slot is a structural feature (directly or inherited) of a classifier of the instance specification.

expression (OCL): slot->forAll(s | classifier->exists (c | c.allFeatures()->includes (s. definingFeature)))

structural_feature

One structural feature (including the same feature inherited from multiple classifiers) is the defining feature of at most one slot in an instance specification.

expression (OCL): classifier->forAll(c | (c.allFeatures()->forAll(f | slot->select(s | s. definingFeature = f)->size() <= 1)))

Package UML::Classes::Kernel

Class InstanceValue

An instance value is a value specification that identifies an instance.

Generalizations:

ValueSpecification

Found in Diagrams:

Expression

Owned Association Ends

+ **instance** : InstanceSpecification [1..1]

The instance that is the specified value.

Package UML::Classes::Kernel

Class LiteralBoolean

A literal Boolean is a specification of a Boolean value.

Generalizations:

LiteralSpecification

Found in Diagrams:

Expression

Attributes

+ **value** : Boolean [1..1] = false

The specified Boolean value.

Operations

+ **booleanValue** () : Boolean [1..1] {query}

The query booleanValue() gives the value.

body (OCL): result = value

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

Package UML::Classes::Kernel

Class LiteralInteger

A literal integer is a specification of an integer value.

Generalizations:

LiteralSpecification

Found in Diagrams:

Expression

Attributes

+ **value** : Integer [1..1] = 0

The specified Integer value.

Operations

+ **integerValue** () : Integer [1..1] {query}

The query integerValue() gives the value.

body (OCL): result = value

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

Package UML::Classes::Kernel

Class LiteralNull

A literal null specifies the lack of a value.

Generalizations:

LiteralSpecification

Found in Diagrams:

Expression

Operations

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

+ **isNull** () : Boolean [1..1] {query}

The query isNull() returns true.

body (OCL): result = true

Package UML::Classes::Kernel

Class LiteralSpecification

A literal specification identifies a literal constant being modeled.

Generalizations:

ValueSpecification

Specializations:

LiteralBoolean, LiteralInteger, LiteralNull, LiteralString, LiteralUnlimitedNatural

Found in Diagrams:

Expression

Package UML::Classes::Kernel

Class LiteralString

A literal string is a specification of a string value.

Generalizations:

LiteralSpecification

Found in Diagrams:

Expression

Attributes

+ **value** : String [0..1]

The specified String value.

Operations

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

+ **stringValue** () : String [1..1] {query}

The query stringValue() gives the value.

body (OCL): result = value

Package UML::Classes::Kernel

Class LiteralUnlimitedNatural

A literal unlimited natural is a specification of an unlimited natural number.

Generalizations:

LiteralSpecification

Found in Diagrams:

Expression

Attributes

+ **value** : UnlimitedNatural [1..1] = 0

The specified UnlimitedNatural value.

Operations

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() is redefined to be true.

body (OCL): result = true

+ **unlimitedValue** () : UnlimitedNatural [1..1] {query}

The query unlimitedValue() gives the value.

body (OCL): result = value

Package UML::Classes::Kernel

Class MultiplicityElement

A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. A multiplicity element embeds this information to specify the allowable cardinalities for an instantiation of this element.

Generalizations:

Element

Specializations:

ConnectorEnd, Parameter, Pin, StructuralFeature, Variable

Found in Diagrams:

Structured Activities

Attributes

+ **isOrdered** : Boolean [1..1] = false

For a multivalued multiplicity, this attribute specifies whether the values in an instantiation of this element are sequentially ordered.

+ **isUnique** : Boolean [1..1] = true

For a multivalued multiplicity, this attributes specifies whether the values in an instantiation of this element are unique.

+ /**lower** : Integer [0..1] = 1

Specifies the lower bound of the multiplicity interval.

+ /**upper** : UnlimitedNatural [0..1] = 1

Specifies the upper bound of the multiplicity interval.

Owned Association Ends

+ **lowerValue** : ValueSpecification [0..1] {subsets ownedElement}

The specification of the lower bound for this multiplicity.

+ **upperValue** : ValueSpecification [0..1] {subsets ownedElement}

The specification of the upper bound for this multiplicity.

Operations

Package UML::Classes::Kernel

Class MultiplicityElement

+ **includesCardinality** (C : Integer) : Boolean [1..1] {query}

The query includesCardinality() checks whether the specified cardinality is valid for this multiplicity.

precondition (): upperBound()->notEmpty() and lowerBound()->notEmpty()

body (OCL): result = (lowerBound() <= C) and (upperBound() >= C)

+ **includesMultiplicity** (M : MultiplicityElement) : Boolean [1..1] {query}

The query includesMultiplicity() checks whether this multiplicity includes all the cardinalities allowed by the specified multiplicity.

precondition (): self.upperBound()->notEmpty() and self.lowerBound()->notEmpty() and M.upperBound()->notEmpty() and M.lowerBound()->notEmpty()

body (OCL): result = (self.lowerBound() <= M.lowerBound()) and (self.upperBound() >= M.upperBound())

+ **isMultivalued** () : Boolean [1..1] {query}

The query isMultivalued() checks whether this multiplicity has an upper bound greater than one.

precondition (): upperBound()->notEmpty()

body (OCL): result = upperBound() > 1

+ **lower** () : Integer [1..1] {query}

The derived lower attribute must equal the lowerBound.

body (OCL): result = lowerBound()

+ **lowerBound** () : Integer [1..1] {query}

The query lowerBound() returns the lower bound of the multiplicity as an integer.

body (OCL): result = if lowerValue->isEmpty() then 1 else lowerValue.integerValue() endif

+ **upper** () : UnlimitedNatural [1..1] {query}

The derived upper attribute must equal the upperBound.

body (OCL): result = upperBound()

+ **upperBound** () : UnlimitedNatural [1..1] {query}

The query upperBound() returns the upper bound of the multiplicity for a bounded multiplicity as an unlimited natural.

body (OCL): result = if upperValue->isEmpty() then 1 else upperValue.unlimitedValue() endif

Constraints

Package UML::Classes::Kernel

Class MultiplicityElement

lower_ge_0

The lower bound must be a non-negative integer literal.

expression (OCL): lowerBound()->notEmpty() implies lowerBound() >= 0

upper_ge_lower

The upper bound must be greater than or equal to the lower bound.

expression (OCL): (upperBound()->notEmpty() and lowerBound()->notEmpty()) implies upperBound() >= lowerBound()

value_specification_constant

If a non-literal ValueSpecification is used for the lower or upper bound, then that specification must be a constant expression.

expression (OCL): true

value_specification_no_side_effects

If a non-literal ValueSpecification is used for the lower or upper bound, then evaluating that specification must not have side effects.

expression (OCL): true

Package UML::Classes::Kernel

Class NamedElement

A named element is an element in a model that may have a name.

Generalizations:

Element

Specializations:

Action, Action, ActivityGroup, ActivityNode, CollaborationUse, Extend, GeneralOrdering, Include , InteractionFragment, InteractionFragment, Lifeline, Message, MessageEnd, Namespace, PackageableElement, ParameterSet, RedefinableElement, Trigger, TypedElement, Vertex

Found in Diagrams:

Basic Actions, Classifiers, Collaboration Use and Role Binding, Fundamental Groups, Fundamental Nodes, Interactions, Lifelines, Messages, Simple Time

Attributes

+ **name** : String [0..1]

The name of the NamedElement.

+ /**qualifiedName** : String [0..1] {readOnly}

A name which allows the NamedElement to be identified within a hierarchy of nested Namespaces. It is constructed from the names of the containing namespaces starting at the root of the hierarchy and ending with the name of the NamedElement itself.

+ **visibility** : VisibilityKind [0..1]

Determines where the NamedElement appears within different Namespaces within the overall model, and its accessibility.

Owned Association Ends

+ /**namespace** : Namespace [0..1] {readOnly, union, subsets owner}

Specifies the namespace that owns the NamedElement.

Operations

+ **allNamespaces** () : Namespace [0..*] {ordered, query}

The query allNamespaces() gives the sequence of namespaces in which the NamedElement is nested, working outwards.

body (OCL): result = if self.namespace->isEmpty() then Sequence{} else self.namespace.

Package UML::Classes::Kernel

Class NamedElement

allNamespaces()->prepend(self.namespace) endif

+ **isDistinguishableFrom** (n : NamedElement, ns : Namespace) : Boolean [1..1] {query}

The query isDistinguishableFrom() determines whether two NamedElements may logically co-exist within a Namespace. By default, two named elements are distinguishable if (a) they have unrelated types or (b) they have related types but different names.

body (OCL): result = if self.oclIsKindOf(n.oclType) or n.oclIsKindOf(self.oclType) then ns. getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->isEmpty() else true endif

+ **separator** () : String [1..1] {query}

The query separator() gives the string that is used to separate names when constructing a qualified name.

body (OCL): result = '::'

Constraints

has_no_qualified_name

If there is no name, or one of the containing namespaces has no name, there is no qualified name.

expression (OCL): (self.name->isEmpty() or self.allNamespaces()->select(ns | ns.name->isEmpty ())->notEmpty()) implies self.qualifiedName->isEmpty()

has_qualified_name

When there is a name, and all of the containing namespaces have a name, the qualified name is constructed from the names of the containing namespaces.

expression (OCL): (self.name->notEmpty() and self.allNamespaces()->select(ns | ns.name-> isEmpty())->isEmpty()) implies self.qualifiedName = self.allNamespaces()->iterate(ns : Namespace; result: String = self.name | ns.name->union(self.separator())->union(result))

visibility_needs_ownership

If a NamedElement is not owned by a Namespace, it does not have a visibility.

expression (OCL): namespace->isEmpty() implies visibility->isEmpty()

Package UML::Classes::Kernel

Class Namespace

A namespace is an element in a model that contains a set of named elements that can be identified by name.

Generalizations:

NamedElement

Specializations:

BehavioralFeature, BehavioralFeature, Classifier, Classifier, Classifier, Classifier, Classifier, InteractionOperand, Package, Region, State, State, StructuredActivityNode, Transition

Found in Diagrams:

Classifier Templates, Classifiers, Collaboration Use and Role Binding, Structured Activities, Structured Classifier

Owned Association Ends

+ **elementImport** : ElementImport [0..*] {subsets ownedElement}

References the ElementImports owned by the Namespace.

+ /**importedMember** : PackageableElement [0..*] {readOnly, subsets member}

References the PackageableElements that are members of this Namespace as a result of either PackageImports or ElementImports.

+ /**member** : NamedElement [0..*] {readOnly, union}

A collection of NamedElements identifiable within the Namespace, either by being owned or by being introduced by importing or inheritance.

+ /**ownedMember** : NamedElement [0..*] {readOnly, union, subsets member, subsets ownedElement}

A collection of NamedElements owned by the Namespace.

+ **ownedRule** : Constraint [0..*] {subsets ownedMember}

Specifies a set of Constraints owned by this Namespace.

+ **packageImport** : PackageImport [0..*] {subsets ownedElement}

References the PackageImports owned by the Namespace.

Operations

Package UML::Classes::Kernel

Class Namespace

+ **excludeCollisions** (imps : PackageableElement) : PackageableElement [0..*] {query}

The query excludeCollisions() excludes from a set of PackageableElements any that would not be distinguishable from each other in this namespace.

body (OCL): result = imps->reject(imp1 | imps.exists(imp2 | not imp1.isDistinguishableFrom (imp2, self)))

+ **getNamesOfMember** (element : NamedElement) : String [0..*] {query}

The query getNamesOfMember() gives a set of all of the names that a member would have in a Namespace. In general a member can have multiple names in a Namespace if it is imported more than once with different aliases. The query takes account of importing. It gives back the set of names that an element would have in an importing namespace, either because it is owned, or if not owned then imported individually, or if not individually then from a package.

body (OCL): result = if self.ownedMember ->includes(element) then Set{}->include(element. name) else let elementImports: ElementImport = self.elementImport->select(ei | ei. importedElement = element) in if elementImports->notEmpty() then elementImports->collect(el | el. getName()) else self.packageImport->select(pi | pi.importedPackage.visibleMembers()->includes (element))-> collect(pi | pi.importedPackage.getNamesOfMember(element)) endif endif

+ **importMembers** (imps : PackageableElement) : PackageableElement [0..*] {query}

The query importMembers() defines which of a set of PackageableElements are actually imported into the namespace. This excludes hidden ones, i.e., those which have names that conflict with names of owned members, and also excludes elements which would have the same name when imported.

body (OCL): result = self.excludeCollisions(imps)->select(imp | self.ownedMember->forAll(mem | mem.imp.isDistinguishableFrom(mem, self)))

+ **importedMember** () : PackageableElement [0..*] {query}

The importedMember property is derived from the ElementImports and the PackageImports. References the PackageableElements that are members of this Namespace as a result of either PackageImports or ElementImports.

body (OCL): result = self.importMembers(self.elementImport.importedElement.asSet()- >union (self.packageImport.importedPackage->collect(p | p.visibleMembers())))

+ **membersAreDistinguishable** () : Boolean [1..1] {query}

The Boolean query membersAreDistinguishable() determines whether all of the namespace's members are distinguishable within it.

body (OCL): result = self.member->forAll(memb | self.member->excluding(memb)->forAll(other | memb.isDistinguishableFrom(other, self)))

Package UML::Classes::Kernel

Class Namespace

Constraints

members_distinguishable

All the members of a Namespace are distinguishable within it.

expression (OCL): membersAreDistinguishable()

Package UML::Classes::Kernel

Class OpaqueExpression

An opaque expression is an uninterpreted textual statement that denotes a (possibly empty) set of values when evaluated in a context.

Generalizations:

ValueSpecification

Found in Diagrams:

Expression

Attributes

+ **body** : String [0..*] {ordered, nonunique}

The text of the expression, possibly in multiple languages.

+ **language** : String [0..*] {ordered}

Specifies the languages in which the expression is stated. The interpretation of the expression body depends on the languages. If the languages are unspecified, they might be implicit from the expression body or the context. Languages are matched to body strings by order.

Operations

+ **isIntegral** () : Boolean [1..1] {query}

The query isIntegral() tells whether an expression is intended to produce an integer.

body (OCL): result = false

+ **isNonNegative** () : Boolean [1..1] {query}

The query isNonNegative() tells whether an integer expression has a non-negative value.

precondition (): self.isIntegral()

body (OCL): result = false

+ **isPositive** () : Boolean [1..1] {query}

The query isPositive() tells whether an integer expression has a positive value.

precondition (): self.isIntegral()

body (OCL): result = false

+ **value** () : Integer [1..1] {query}

The query value() gives an integer value for an expression intended to produce one.

precondition (): self.isIntegral()

Package UML::Classes::Kernel

Class OpaqueExpression

body (OCL): true

Constraints

language_body_size

If the language attribute is not empty, then the size of the body and language arrays must be the same.

expression (OCL): language->notEmpty() implies (body->size() = language->size())

Package UML::Classes::Kernel

Class Operation

An operation is a behavioral feature of a classifier that specifies the name, type, parameters, and constraints for invoking an associated behavior.

Generalizations:

BehavioralFeature

Attributes

+ /**isOrdered** : Boolean [1..1] = false

  Specifies whether the return parameter is ordered or not, if present.

+ **isQuery** : Boolean [1..1] = false

  Specifies whether an execution of the BehavioralFeature leaves the state of the system unchanged (isQuery=true) or whether side effects may occur (isQuery=false).

+ /**isUnique** : Boolean [1..1] = true

  Specifies whether the return parameter is unique or not, if present.

+ /**lower** : Integer [0..1] = 1

  Specifies the lower multiplicity of the return parameter, if present.

+ /**upper** : UnlimitedNatural [0..1] = 1

  Specifies the upper multiplicity of the return parameter, if present.

Owned Association Ends

+ **bodyCondition** : Constraint [0..1] {subsets ownedRule}

  An optional Constraint on the result values of an invocation of this Operation.

+ **class** : Class [0..1] {subsets redefinitionContext, subsets namespace, subsets featuringClassifier}

  The class that owns the operation.

+ **datatype** : DataType [0..1] {subsets namespace, subsets redefinitionContext, subsets featuringClassifier}

  The DataType that owns this Operation.

Package UML::Classes::Kernel

Class Operation

+ **ownedParameter** : Parameter [0..*] {ordered, redefines ownedParameter}

Specifies the parameters owned by this Operation.

+ **postcondition** : Constraint [0..*] {subsets ownedRule}

An optional set of Constraints specifying the state of the system when the Operation is completed.

+ **precondition** : Constraint [0..*] {subsets ownedRule}

An optional set of Constraints on the state of the system when the Operation is invoked.

+ **raisedException** : Type [0..*] {redefines raisedException}

References the Types representing exceptions that may be raised during an invocation of this operation.

+ **redefinedOperation** : Operation [0..*] {subsets redefinedElement}

References the Operations that are redefined by this Operation.

+ /**type** : Type [0..1]

Specifies the return result of the operation, if present.

Operations

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

A redefining operation is consistent with a redefined operation if it has the same number of owned parameters, and the type of each owned parameter conforms to the type of the corresponding redefined parameter.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = redefinee.oclIsKindOf(Operation) and let op : Operation = redefinee.oclAsType(Operation) in self.ownedParameter->size() = op.ownedParameter->size() and Sequence{1..self.ownedParameter->size()}-> forAll(i |op.ownedParameter->at(1).type.conformsTo(self.ownedParameter->at(i).type))

+ **isOrdered** () : Boolean [1..1] {query}

If this operation has a return parameter, isOrdered equals the value of isOrdered for that parameter. Otherwise isOrdered is false.

body (OCL): result = if returnResult()->notEmpty() then returnResult()->any().isOrdered else false

Package UML::Classes::Kernel

Class Operation

endif

+ **isUnique** () : Boolean [1..1] {query}

If this operation has a return parameter, isUnique equals the value of isUnique for that parameter. Otherwise isUnique is true.

body (OCL): result = if returnResult()->notEmpty() then returnResult()->any().isUnique else true endif

+ **lower** () : Integer [1..1] {query}

If this operation has a return parameter, lower equals the value of lower for that parameter. Otherwise lower is not defined.

body (OCL): result = if returnResult()->notEmpty() then returnResult()->any().lower else Set{} endif

+ **returnResult** () : Parameter [0..*] {query}

The query returnResult() returns the set containing the return parameter of the Operation if one exists, otherwise, it returns an empty set

body (OCL): result = ownedParameter->select (par | par.direction = #return)

+ **type** () : Type [1..1] {query}

If this operation has a return parameter, type equals the value of type for that parameter. Otherwise type is not defined.

body (OCL): result = if returnResult()->notEmpty() then returnResult()->any().type else Set{} endif

+ **upper** () : UnlimitedNatural [1..1] {query}

If this operation has a return parameter, upper equals the value of upper for that parameter. Otherwise upper is not defined.

body (OCL): result = if returnResult()->notEmpty() then returnResult()->any().upper else Set{} endif

Constraints

at_most_one_return

An operation can have at most one return parameter; i.e., an owned parameter with the direction set to 'return'

expression (OCL): self.ownedParameter->select(par | par.direction = #return)->size() <= 1

only_body_for_query

Package UML::Classes::Kernel

Class Operation

A bodyCondition can only be specified for a query operation.

expression (OCL): bodyCondition->notEmpty() implies isQuery

Package UML::Classes::Kernel

Class Package

A package is used to group elements, and provides a namespace for the grouped elements.

Generalizations:

Namespace, PackageableElement

Specializations:

Model

Owned Association Ends

+ /**nestedPackage** : Package [0..*] {subsets packagedElement}

References the packaged elements that are Packages.

+ **nestingPackage** : Package [0..1] {subsets namespace}

References the Package that owns this Package.

+ /**ownedType** : Type [0..*] {subsets packagedElement}

References the packaged elements that are Types.

+ **packageMerge** : PackageMerge [0..*] {subsets ownedElement}

References the PackageMerges that are owned by this Package.

+ **packagedElement** : PackageableElement [0..*] {subsets ownedMember}

Specifies the packageable elements that are owned by this Package.

Operations

+ **makesVisible** (el : NamedElement) : Boolean [1..1] {query}

The query makesVisible() defines whether a Package makes an element visible outside itself. Elements with no visibility and elements with public visibility are made visible.

precondition (): self.member->includes(el)

body (OCL): result = (ownedMember->includes(el)) or (elementImport->select(ei|ei.importedElement = #public)->collect(ei|ei.importedElement)->includes(el)) or (packageImport->select(pi|pi.visibility = #public)->collect(pi|pi.importedPackage.member->includes(el))->notEmpty())

+ **mustBeOwned** () : Boolean [1..1] {query}

Package UML::Classes::Kernel

Class Package

The query mustBeOwned() indicates whether elements of this type must have an owner.

body (OCL): result = false

\+ **visibleMembers** () : PackageableElement [0..*] {query}

The query visibleMembers() defines which members of a Package can be accessed outside it.

body (OCL): result = member->select(m | self.makesVisible(m))

Constraints

elements_public_or_private

If an element that is owned by a package has visibility, it is public or private.

expression (OCL): self.ownedElements->forAll(e | e.visibility->notEmpty() implies e.visbility = #public or e.visibility = #private)

Package UML::Classes::Kernel

Class PackageImport

A package import is a relationship that allows the use of unqualified names to refer to package members from other namespaces.

Generalizations:

DirectedRelationship

Attributes

+ **visibility** : VisibilityKind [1..1] = public

Specifies the visibility of the imported PackageableElements within the importing Namespace, i.e., whether imported elements will in turn be visible to other packages that use that importingPackage as an importedPackage. If the PackageImport is public, the imported elements will be visible outside the package, while if it is private they will not.

Owned Association Ends

+ **importedPackage** : Package [1..1] {subsets target}

Specifies the Package whose members are imported into a Namespace.

+ **importingNamespace** : Namespace [1..1] {subsets source, subsets owner}

Specifies the Namespace that imports the members from a Package.

Constraints

public_or_private

The visibility of a PackageImport is either public or private.

expression (OCL): self.visibility = #public or self.visibility = #private

Package UML::Classes::Kernel

Class PackageMerge

A package merge defines how the contents of one package are extended by the contents of another package.

Generalizations:

DirectedRelationship

Owned Association Ends

+ **mergedPackage** : Package [1..1] {subsets target}

References the Package that is to be merged with the receiving package of the PackageMerge.

+ **receivingPackage** : Package [1..1] {subsets source, subsets owner}

References the Package that is being extended with the contents of the merged package of the PackageMerge.

Package UML::Classes::Kernel

Class PackageableElement

A packageable element indicates a named element that may be owned directly by a package.

Generalizations:

NamedElement

Specializations:

Constraint, Event, GeneralizationSet, InformationFlow, InstanceSpecification, Observation, Package, Type, ValueSpecification

Found in Diagrams:

Expression, Packaging Components, Simple Time

Attributes

+ **visibility** : VisibilityKind [1..1] = public {redefines visibility}

Indicates that packageable elements must always have a visibility, i.e., visibility is not optional.

Package UML::Classes::Kernel

Class Parameter

A parameter is a specification of an argument used to pass information into or out of an invocation of a behavioral feature.

Generalizations:

MultiplicityElement, TypedElement

Found in Diagrams:

Common Behavior

Attributes

+ /**default** : String [0..1]

Specifies a String that represents a value to be used when no argument is supplied for the Parameter.

+ **direction** : ParameterDirectionKind [1..1] = in

Indicates whether a parameter is being sent into or out of a behavioral element.

Owned Association Ends

+ **defaultValue** : ValueSpecification [0..1] {subsets ownedElement}

Specifies a ValueSpecification that represents a value to be used when no argument is supplied for the Parameter.

+ **operation** : Operation [0..1] {subsets namespace}

References the Operation owning this parameter.

Package UML::Classes::Kernel

Class PrimitiveType

A primitive type defines a predefined data type, without any relevant substructure (i.e., it has no parts in the context of UML). A primitive datatype may have an algebra and operations defined outside of UML, for example, mathematically.

Generalizations:

DataType

Package UML::Classes::Kernel

Class Property

A property is a structural feature of a classifier that characterizes instances of the classifier. A property related by ownedAttribute to a classifier (other than an association) represents an attribute and might also represent an association end. It relates an instance of the class to a value or set of values of the type of the attribute. A property related by memberEnd or its specializations to an association represents an end of the association. The type of the property is the type of the end of the association.

Generalizations:

StructuralFeature

Specializations:

Port

Found in Diagrams:

Classifiers, Reception, The port metaclass

Attributes

+ **aggregation** : AggregationKind [1..1] = none

Specifies the kind of aggregation that applies to the Property.

+ /**default** : String [0..1]

A String that is evaluated to give a default value for the Property when an object of the owning Classifier is instantiated.

+ /**isComposite** : Boolean [1..1]

This is a derived value, indicating whether the aggregation of the Property is composite or not.

+ **isDerived** : Boolean [1..1] = false

Specifies whether the Property is derived, i.e., whether its value or values can be computed from other information.

+ **isDerivedUnion** : Boolean [1..1] = false

Specifies whether the property is derived as the union of all of the properties that are constrained to subset it.

+ **isReadOnly** : Boolean [1..1] = false {redefines isReadOnly}

If true, the attribute may only be read, and not written.

Owned Association Ends

Package UML::Classes::Kernel

Class Property

+ **association** : Association [0..1]

References the association of which this property is a member, if any.

+ **class** : Class [0..1] {subsets classifier, subsets namespace, subsets featuringClassifier}

References the Class that owns the Property.

+ **datatype** : DataType [0..1] {subsets namespace, subsets featuringClassifier, subsets classifier}

The DataType that owns this Property.

+ **defaultValue** : ValueSpecification [0..1] {subsets ownedElement}

A ValueSpecification that is evaluated to give a default value for the Property when an object of the owning Classifier is instantiated.

+ /**opposite** : Property [0..1]

In the case where the property is one navigable end of a binary association with both ends navigable, this gives the other end.

+ **owningAssociation** : Association [0..1] {subsets association, subsets namespace, subsets featuringClassifier}

References the owning association of this property, if any.

+ **redefinedProperty** : Property [0..*] {subsets redefinedElement}

References the properties that are redefined by this property.

+ **subsettedProperty** : Property [0..*]

References the properties of which this property is constrained to be a subset.

Operations

+ **isAttribute** (p : Property) : Boolean [1..1] {query}

The query isAttribute() is true if the Property is defined as an attribute of some classifier.

body (OCL): result = Classifier.allInstances->exists(c | c.attribute->includes(p))

+ **isComposite** () : Boolean [1..1] {query}

Package UML::Classes::Kernel

Class Property

The value of isComposite is true only if aggregation is composite.

body (OCL): result = (self.aggregation = #composite)

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two Properties in a context in which redefinition is possible, whether redefinition would be logically consistent. A redefining property is consistent with a redefined property if the type of the redefining property conforms to the type of the redefined property, the multiplicity of the redefining property (if specified) is contained in the multiplicity of the redefined property, and the redefining property is derived if the redefined property is derived.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = redefinee.oclIsKindOf(Property) and let prop : Property = redefinee.oclAsType(Property) in (prop.type.conformsTo(self.type) and ((prop.lowerBound()->notEmpty() and self.lowerBound()->notEmpty()) implies prop.lowerBound() >= self.lowerBound()) and ((prop.upperBound()->notEmpty() and self.upperBound()->notEmpty()) implies prop.lowerBound() <= self.lowerBound()) and (self.isDerived implies prop.isDerived) and (self.isComposite implies prop.isComposite))

+ **isNavigable** () : Boolean [1..1] {query}

The query isNavigable() indicates whether it is possible to navigate across the property.

body (OCL): result = not classifier->isEmpty() or association.owningAssociation.navigableOwnedEnd->includes(self)

+ **opposite** () : Property [1..1] {query}

If this property is owned by a class, associated with a binary association, and the other end of the association is also owned by a class, then opposite gives the other end.

body (OCL): result = if owningAssociation->isEmpty() and association.memberEnd->size() = 2 then let otherEnd = (association.memberEnd - self)->any() in if otherEnd.owningAssociation->isEmpty() then otherEnd else Set{} endif else Set {} endif

+ **subsettingContext** () : Type [0..*] {query}

The query subsettingContext() gives the context for subsetting a property. It consists, in the case of an attribute, of the corresponding classifier, and in the case of an association end, all of the classifiers at the other ends.

body (OCL): result = if association->notEmpty() then association.endType-type else if classifier->notEmpty() then Set{classifier} else Set{} endif endif

Constraints

Package UML::Classes::Kernel

Class Property

derived_union_is_derived

A derived union is derived.

expression (OCL): isDerivedUnion implies isDerived

derived_union_is_read_only

A derived union is read only.

expression (OCL): isDerivedUnion implies isReadOnly

multiplicity_of_composite

A multiplicity on an aggregate end of a composite aggregation must not have an upper bound greater than 1.

expression (OCL): isComposite implies (upperBound()->isEmpty() or upperBound() <= 1)

navigable_readonly

Only a navigable property can be marked as readOnly.

expression (OCL): isReadOnly implies isNavigable()

redefined_property_inherited

A redefined property must be inherited from a more general classifier containing the redefining property.

expression (OCL): if (redefinedProperty->notEmpty()) then (redefinitionContext->notEmpty() and redefinedProperty->forAll(rp| ((redefinitionContext->collect(fc| fc.allParents()))->asSet())->collect (c| c.allFeatures())->asSet()->includes(rp))

subsetted_property_names

A property may not subset a property with the same name.

expression (OCL): true

subsetting_context_conforms

Subsetting may only occur when the context of the subsetting property conforms to the context of the subsetted property.

expression (OCL): self.subsettedProperty->notEmpty() implies (self.subsettingContext()-> notEmpty() and self.subsettingContext()->forAll (sc | self.subsettedProperty->forAll(sp | sp. subsettingContext()->exists(c | sc.conformsTo(c)))))

subsetting_rules

A subsetting property may strengthen the type of the subsetted property, and its upper bound may be less.

Package UML::Classes::Kernel

Class Property

expression (OCL): self.subsettedProperty->forAll(sp | self.type.conformsTo(sp.type) and ((self.upperBound()->notEmpty() and sp.upperBound()->notEmpty()) implies self.upperBound()<=sp.upperBound()))

Package UML::Classes::Kernel

Class RedefinableElement

A redefinable element is an element that, when defined in the context of a classifier, can be redefined more specifically or differently in the context of another classifier that specializes (directly or indirectly) the context classifier.

Generalizations:

NamedElement

Specializations:

ActivityEdge, ActivityEdge, ActivityEdge, ActivityNode, ActivityNode, ActivityNode, Classifier, ExtensionPoint, Feature, RedefinableTemplateSignature, Region, State, Transition

Found in Diagrams:

Activity Partitions, Classifier Templates, Classifiers, Structured Activities

Attributes

+ **isLeaf** : Boolean [1..1] = false

Indicates whether it is possible to further redefine a RedefinableElement. If the value is true, then it is not possible to further redefine the RedefinableElement. Note that this property is preserved through package merge operations; that is, the capability to redefine a RedefinableElement (i.e., isLeaf=false) must be preserved in the resulting RedefinableElement of a package merge operation where a RedefinableElement with isLeaf=false is merged with a matching RedefinableElement with isLeaf=true: the resulting RedefinableElement will have isLeaf=false. Default value is false.

Owned Association Ends

+ /**redefinedElement** : RedefinableElement [0..*] {readOnly, union}

The redefinable element that is being redefined by this element.

+ /**redefinitionContext** : Classifier [0..*] {readOnly, union}

References the contexts that this element may be redefined from.

Operations

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies, for any two RedefinableElements in a context in which redefinition is possible, whether redefinition would be logically consistent. By default, this is false; this operation must be overridden for subclasses of RedefinableElement to define the consistency conditions.

Package UML::Classes::Kernel

Class RedefinableElement

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = false

\+ **isRedefinitionContextValid** (redefined : RedefinableElement) : Boolean [1..1] {query}

The query isRedefinitionContextValid() specifies whether the redefinition contexts of this RedefinableElement are properly related to the redefinition contexts of the specified RedefinableElement to allow this element to redefine the other. By default at least one of the redefinition contexts of this element must be a specialization of at least one of the redefinition contexts of the specified element.

body (OCL): result = redefinitionContext->exists(c | c.allParents()->includes(redefined.redefinitionContext)))

Constraints

non_leaf_redefinition

A redefinable element can only redefine non-leaf redefinable elements

expression (OCL): self.redefinedElement->forAll(not isLeaf)

redefinition_consistent

A redefining element must be consistent with each redefined element.

expression (OCL): self.redefinedElement->forAll(re | re.isConsistentWith(self))

redefinition_context_valid

At least one of the redefinition contexts of the redefining element must be a specialization of at least one of the redefinition contexts for each redefined element.

expression (OCL): self.redefinedElement->forAll(e | self.isRedefinitionContextValid(e))

Package UML::Classes::Kernel

Class Relationship

Relationship is an abstract concept that specifies some kind of relationship between elements.

Generalizations:

Element

Specializations:

Association, DirectedRelationship

Owned Association Ends

+ /**relatedElement** : Element [1..*] {readOnly, union}

Specifies the elements related by the Relationship.

Package UML::Classes::Kernel

Class Slot

A slot specifies that an entity modeled by an instance specification has a value or values for a specific structural feature.

Generalizations:

Element

Owned Association Ends

+ **definingFeature** : StructuralFeature [1..1]

The structural feature that specifies the values that may be held by the slot.

+ **owningInstance** : InstanceSpecification [1..1] {subsets owner}

The instance specification that owns this slot.

+ **value** : ValueSpecification [0..*] {ordered, subsets ownedElement}

The value or values corresponding to the defining feature for the owning instance specification.

Package UML::Classes::Kernel

Class StructuralFeature

By specializing multiplicity element, it supports a multiplicity that specifies valid cardinalities for the collection of values associated with an instantiation of the structural feature.

Generalizations:

Feature, MultiplicityElement, TypedElement

Specializations:

Property, Property, Property, Property

Found in Diagrams:

Interfaces, Structural Feature Actions, Structured Classifier

Attributes

+ **isReadOnly** : Boolean [1..1] = false

States whether the feature's value may be modified by a client.

Package UML::Classes::Kernel

Class Type

A type constrains the values represented by a typed element.

Generalizations:

PackageableElement

Specializations:

Classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **package** : Package [0..1] {subsets namespace}

Specifies the owning package of this classifier, if any.

Operations

+ **conformsTo** (other : Type) : Boolean [1..1] {query}

The query conformsTo() gives true for a type that conforms to another. By default, two types do not conform to each other. This query is intended to be redefined for specific conformance situations.

body (OCL): result = false

Package UML::Classes::Kernel

Class TypedElement

A typed element has a type.

Generalizations:

NamedElement

Specializations:

ConnectableElement, ObjectNode, ObjectNode, Parameter, Pin, StructuralFeature, ValueSpecification, Variable

Found in Diagrams:

Expression, Structured Activities, Structured Classifier

Owned Association Ends

+ **type** : Type [0..1]

The type of the TypedElement.

Package UML::Classes::Kernel

Class ValueSpecification

A value specification is the specification of a (possibly empty) set of instances, including both objects and data values.

Generalizations:

PackageableElement, TypedElement

Specializations:

Duration, Expression, InstanceValue, Interval, LiteralSpecification, OpaqueExpression, TimeExpression

Found in Diagrams:

Activity Partitions, Events, Expression, Lifelines, Messages, Simple Time

Operations

+ **booleanValue** () : Boolean [1..1] {query}

The query booleanValue() gives a single Boolean value when one can be computed.

body (OCL): result = Set{}

+ **integerValue** () : Integer [1..1] {query}

The query integerValue() gives a single Integer value when one can be computed.

body (OCL): result = Set{}

+ **isComputable** () : Boolean [1..1] {query}

The query isComputable() determines whether a value specification can be computed in a model. This operation cannot be fully defined in OCL. A conforming implementation is expected to deliver true for this operation for all value specifications that it can compute, and to compute all of those for which the operation is true. A conforming implementation is expected to be able to compute the value of all literals.

body (OCL): result = false

+ **isNull** () : Boolean [1..1] {query}

The query isNull() returns true when it can be computed that the value is null.

body (OCL): result = false

+ **stringValue** () : String [1..1] {query}

The query stringValue() gives a single String value when one can be computed.

body (OCL): result = Set{}

+ **unlimitedValue** () : UnlimitedNatural [1..1] {query}

The query unlimitedValue() gives a single UnlimitedNatural value when one can be computed.

Package UML::Classes::Kernel

Class ValueSpecification

body (OCL): result = Set{}

Package UML::Classes::Kernel

Enumeration AggregationKind

AggregationKind is an enumeration type that specifies the literals for defining the kind of aggregation of a property.

Enumeration Literals

composite

Indicates that the property is aggregated compositely, i.e., the composite object has responsibility for the existence and storage of the composed objects (parts).

none

Indicates that the property has no aggregation.

shared

Indicates that the property has a shared aggregation.

Package UML::Classes::Kernel

Enumeration ParameterDirectionKind

Parameter direction kind is an enumeration type that defines literals used to specify direction of parameters.

Enumeration Literals

in

Indicates that parameter values are passed into the behavioral element by the caller.

inout

Indicates that parameter values are passed into a behavioral element by the caller and then back out to the caller from the behavioral element.

out

return

Indicates that parameter values are passed as return values from a behavioral element back to the caller.

Package UML::Classes::Kernel

Enumeration VisibilityKind

VisibilityKind is an enumeration type that defines literals to determine the visibility of elements in a model.

Enumeration Literals

package

private

protected

public

Operations

+ **bestVisibility** (vis : VisibilityKind [0..*]) : VisibilityKind [1..1] {query}

The query bestVisibility() examines a set of VisibilityKinds, and returns public as the preferred visibility.

precondition (): pre: not vis->includes(#protected) and not vis->includes(#package)

body (OCL): result = if vis->includes(#public) then #public else #private endif

Package UML::Classes::Kernel

Association A_annotatedElement_comment

Member Ends:

annotatedElement, comment

Owned Association Ends

+ **comment** : Comment [0..*]

Package UML::Classes::Kernel

Association A_attribute_classifier

Member Ends:

attribute, classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **classifier** : Classifier [0..1] {subsets redefinitionContext}

Package UML::Classes::Kernel

Association A_bodyCondition_bodyContext

Member Ends:

bodyCondition, bodyContext

Owned Association Ends

+ **bodyContext** : Operation [0..1] {subsets context}

Package UML::Classes::Kernel

Association A_classifier_instanceSpecification

Member Ends:

classifier, instanceSpecification

Owned Association Ends

+ **instanceSpecification** : InstanceSpecification [0..*]

Package UML::Classes::Kernel

Association A_constrainedElement_constraint

Member Ends:

constrainedElement, constraint

Owned Association Ends

+ **constraint** : Constraint [0..*]

Package UML::Classes::Kernel

Association A_defaultValue_owningParameter

Member Ends:

defaultValue, owningParameter

Owned Association Ends

+ **owningParameter** : Parameter [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_defaultValue_owningProperty

Member Ends:

defaultValue, owningProperty

Owned Association Ends

+ **owningProperty** : Property [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_definingFeature_slot

Member Ends:

definingFeature, slot

Owned Association Ends

+ **slot** : Slot [0..*]

Package UML::Classes::Kernel

Association A_elementImport_importingNamespace

Member Ends:

elementImport, importingNamespace

Package UML::Classes::Kernel

Association A_endType_association

Member Ends:

endType, association

Owned Association Ends

+ **association** : Association [0..*]

Package UML::Classes::Kernel

Association A_feature_featuringClassifier

Member Ends:

feature, featuringClassifier

Package UML::Classes::Kernel

Association A_general_classifier

Member Ends:

general, classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package UML::Classes::Kernel

Association A_general_generalization

Member Ends:

general, generalization

Found in Diagrams:

Classifiers

Owned Association Ends

+ **generalization** : Generalization [0..*]

Package UML::Classes::Kernel

Association A_generalization_specific

Member Ends:

generalization, specific

Found in Diagrams:

Classifiers

Package UML::Classes::Kernel

Association A_importedElement_elementImport

Member Ends:

importedElement, elementImport

Owned Association Ends

+ **elementImport** : ElementImport [1..1]

Package UML::Classes::Kernel

Association A_importedMember_namespace

Member Ends:

importedMember, namespace

Owned Association Ends

+ **namespace** : Namespace [0..*]

Package UML::Classes::Kernel

Association A_importedPackage_packageImport

Member Ends:

importedPackage, packageImport

Owned Association Ends

+ **packageImport** : PackageImport [0..*]

Package UML::Classes::Kernel

Association A_inheritedMember_classifier

Member Ends:

inheritedMember, classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package UML::Classes::Kernel

Association A_instance_instanceValue

Member Ends:

instance, instanceValue

Found in Diagrams:

Expression

Owned Association Ends

+ **instanceValue** : InstanceValue [0..*]

Package UML::Classes::Kernel

Association A_lowerValue_owningLower

Member Ends:

lowerValue, owningLower

Owned Association Ends

+ **owningLower** : MultiplicityElement [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_memberEnd_association

Member Ends:

memberEnd, association

Package UML::Classes::Kernel

Association A_member_namespace

Member Ends:

member, namespace

Owned Association Ends

+ **namespace** : Namespace [0..*]

Package UML::Classes::Kernel

Association A_mergedPackage_packageMerge

Member Ends:

mergedPackage, packageMerge

Owned Association Ends

+ **packageMerge** : PackageMerge [0..*]

Package UML::Classes::Kernel

Association A_navigableOwnedEnd_association

Member Ends:

navigableOwnedEnd, association

Owned Association Ends

+ **association** : Association [0..1]

Package UML::Classes::Kernel

Association A_nestedClassifier_class

Member Ends:

nestedClassifier, class

Owned Association Ends

+ **class** : Class [0..1] {subsets namespace, subsets redefinitionContext}

Package UML::Classes::Kernel

Association A_nestedPackage_nestingPackage

Member Ends:

nestedPackage, nestingPackage

Package UML::Classes::Kernel

Association A_operand_expression

Member Ends:

operand, expression

Found in Diagrams:

Expression

Owned Association Ends

+ **expression** : Expression [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_opposite_property

Member Ends:

opposite, property

Owned Association Ends

+ **property** : Property [0..1]

Package UML::Classes::Kernel

Association A_ownedAttribute_class

Member Ends:

ownedAttribute, class

Package UML::Classes::Kernel

Association A_ownedAttribute_datatype

Member Ends:

ownedAttribute, datatype

Package UML::Classes::Kernel

Association A_ownedComment_owningElement

Member Ends:

ownedComment, owningElement

Owned Association Ends

+ **owningElement** : Element [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_ownedElement_owner

Member Ends:

ownedElement, owner

Package UML::Classes::Kernel

Association A_ownedEnd_owningAssociation

Member Ends:

ownedEnd, owningAssociation

Package UML::Classes::Kernel

Association A_ownedLiteral_enumeration

Member Ends:

ownedLiteral, enumeration

Package UML::Classes::Kernel

Association A_ownedMember_namespace

Member Ends:

ownedMember, namespace

Package UML::Classes::Kernel

Association A_ownedOperation_class

Member Ends:

ownedOperation, class

Package UML::Classes::Kernel

Association A_ownedOperation_datatype

Member Ends:

ownedOperation, datatype

Package UML::Classes::Kernel

Association A_ownedParameter_operation

Member Ends:

ownedParameter, operation

Package UML::Classes::Kernel

Association A_ownedParameter_ownerFormalParam

Member Ends:

ownedParameter, ownerFormalParam

Owned Association Ends

+ **ownerFormalParam** : BehavioralFeature [0..1] {subsets namespace}

Package UML::Classes::Kernel

Association A_ownedRule_context

Member Ends:

ownedRule, context

Package UML::Classes::Kernel

Association A_ownedType_package

Member Ends:

ownedType, package

Package UML::Classes::Kernel

Association A_packageImport_importingNamespace

Member Ends:

packageImport, importingNamespace

Package UML::Classes::Kernel

Association A_packageMerge_receivingPackage

Member Ends:

packageMerge, receivingPackage

Package UML::Classes::Kernel

Association A_packagedElement_owningPackage

Member Ends:

packagedElement, owningPackage

Owned Association Ends

+ **owningPackage** : Package [0..1] {subsets namespace}

Package UML::Classes::Kernel

Association A_postcondition_postContext

Member Ends:

postcondition, postContext

Owned Association Ends

+ **postContext** : Operation [0..1] {subsets context}

Package UML::Classes::Kernel

Association A_precondition_preContext

Member Ends:

precondition, preContext

Owned Association Ends

+ **preContext** : Operation [0..1] {subsets context}

Package UML::Classes::Kernel

Association A_raisedException_behavioralFeature

Member Ends:

raisedException, behavioralFeature

Owned Association Ends

+ **behavioralFeature** : BehavioralFeature [0..*]

Package UML::Classes::Kernel

Association A_raisedException_operation

Member Ends:

raisedException, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package UML::Classes::Kernel

Association A_redefinedClassifier_classifier

Member Ends:

redefinedClassifier, classifier

Found in Diagrams:

Classifiers

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package UML::Classes::Kernel

Association A_redefinedElement_redefinableElement

Member Ends:

redefinedElement, redefinableElement

Found in Diagrams:

Classifiers

Owned Association Ends

+ **redefinableElement** : RedefinableElement [0..*]

Package UML::Classes::Kernel

Association A_redefinedOperation_operation

Member Ends:

redefinedOperation, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package UML::Classes::Kernel

Association A_redefinedProperty_property

Member Ends:

redefinedProperty, property

Owned Association Ends

+ **property** : Property [0..*]

Package UML::Classes::Kernel

Association A_redefinitionContext_redefinableElement

Member Ends:

redefinitionContext, redefinableElement

Found in Diagrams:

Classifiers

Owned Association Ends

+ **redefinableElement** : RedefinableElement [0..*]

Package UML::Classes::Kernel

Association A_relatedElement_relationship

Member Ends:

relatedElement, relationship

Owned Association Ends

+ **relationship** : Relationship [0..*]

Package UML::Classes::Kernel

Association A_slot_owningInstance

Member Ends:

slot, owningInstance

Package UML::Classes::Kernel

Association A_source_directedRelationship

Member Ends:

source, directedRelationship

Owned Association Ends

+ **directedRelationship** : DirectedRelationship [0..*]

Package UML::Classes::Kernel

Association A_specification_owningConstraint

Member Ends:

specification, owningConstraint

Owned Association Ends

+ **owningConstraint** : Constraint [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_specification_owningInstanceSpec

Member Ends:

specification, owningInstanceSpec

Owned Association Ends

+ **owningInstanceSpec** : InstanceSpecification [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_subsettedProperty_property

Member Ends:

subsettedProperty, property

Owned Association Ends

+ **property** : Property [0..*]

Package UML::Classes::Kernel

Association A_superClass_class

Member Ends:

superClass, class

Owned Association Ends

+ **class** : Class [0..*]

Package UML::Classes::Kernel

Association A_target_directedRelationship

Member Ends:

target, directedRelationship

Owned Association Ends

+ **directedRelationship** : DirectedRelationship [0..*]

Package UML::Classes::Kernel

Association A_type_operation

Member Ends:

type, operation

Owned Association Ends

+ **operation** : Operation [0..*]

Package UML::Classes::Kernel

Association A_type_typedElement

Member Ends:

type, typedElement

Owned Association Ends

+ **typedElement** : TypedElement [0..*]

Package UML::Classes::Kernel

Association A_upperValue_owningUpper

Member Ends:

upperValue, owningUpper

Owned Association Ends

+ **owningUpper** : MultiplicityElement [0..1] {subsets owner}

Package UML::Classes::Kernel

Association A_value_owningSlot

Member Ends:

value, owningSlot

Owned Association Ends

+ **owningSlot** : Slot [0..1] {subsets owner}

Package UML::Classes::PowerTypes

Nesting Package:

Classes

Merged Packages:

Kernel

Class Summary
Classifier
Generalization
GeneralizationSet

Association Summary
A_generalizationSet_generalization
A_powertypeExtent_powertype

Package UML::Classes::PowerTypes

Class Classifier

Owned Association Ends

+ **powertypeExtent** : GeneralizationSet [0..*]

Designates the GeneralizationSet of which the associated Classifier is a power type.

Constraints

maps_to_generalization_set

The Classifier that maps to a GeneralizationSet may neither be a specific nor a general Classifier in any of the Generalization relationships defined for that GeneralizationSet. In other words, a power type may not be an instance of itself nor may its instances also be its subclasses.

expression (OCL): true

Package UML::Classes::PowerTypes

Class Generalization

A generalization relates a specific classifier to a more general classifier, and is owned by the specific classifier.

Owned Association Ends

+ **generalizationSet** : GeneralizationSet [0..*]

Designates a set in which instances of Generalization is considered members.

Constraints

generalization_same_classifier

Every Generalization associated with a given GeneralizationSet must have the same general Classifier. That is, all Generalizations for a particular GeneralizationSet must have the same superclass.

expression (OCL): true

Package UML::Classes::PowerTypes

Class GeneralizationSet

A generalization set is a packageable element whose instances define collections of subsets of generalization relationships.

Generalizations:

PackageableElement

Attributes

+ **isCovering** : Boolean [1..1] = false

Indicates (via the associated Generalizations) whether or not the set of specific Classifiers are covering for a particular general classifier. When isCovering is true, every instance of a particular general Classifier is also an instance of at least one of its specific Classifiers for the GeneralizationSet. When isCovering is false, there are one or more instances of the particular general Classifier that are not instances of at least one of its specific Classifiers defined for the GeneralizationSet.

+ **isDisjoint** : Boolean [1..1] = false

Indicates whether or not the set of specific Classifiers in a Generalization relationship have instance in common. If isDisjoint is true, the specific Classifiers for a particular GeneralizationSet have no members in common; that is, their intersection is empty. If isDisjoint is false, the specific Classifiers in a particular GeneralizationSet have one or more members in common; that is, their intersection is not empty. For example, Person could have two Generalization relationships, each with the different specific Classifier: Manager or Staff. This would be disjoint because every instance of Person must either be a Manager or Staff. In contrast, Person could have two Generalization relationships involving two specific (and non-covering) Classifiers: Sales Person and Manager. This GeneralizationSet would not be disjoint because there are instances of Person which can be a Sales Person and a Manager.

Owned Association Ends

+ **generalization** : Generalization [0..*]

Designates the instances of Generalization which are members of a given GeneralizationSet.

+ **powertype** : Classifier [0..1]

Designates the Classifier that is defined as the power type for the associated GeneralizationSet.

Constraints

generalization_same_classifier

Package UML::Classes::PowerTypes

Class GeneralizationSet

Every Generalization associated with a particular GeneralizationSet must have the same general Classifier.

expression (OCL): generalization->collect(g | g.general)->asSet()->size() <= 1

maps_to_generalization_set

The Classifier that maps to a GeneralizationSet may neither be a specific nor a general Classifier in any of the Generalization relationships defined for that GeneralizationSet. In other words, a power type may not be an instance of itself nor may its instances be its subclasses.

expression (OCL): true

Package UML::Classes::PowerTypes

Association A_generalizationSet_generalization

Member Ends:

generalizationSet, generalization

Package UML::Classes::PowerTypes

Association A_powertypeExtent_powertype

Member Ends:

powertypeExtent, powertype

Package UML::CommonBehaviors

Nesting Package:

UML

Imported Packages:

Actions, Classes

Nested Package Summary
BasicBehaviors
Communications
SimpleTime

Package UML::CommonBehaviors::BasicBehaviors

Nesting Package:

CommonBehaviors

Merged Packages:

Kernel

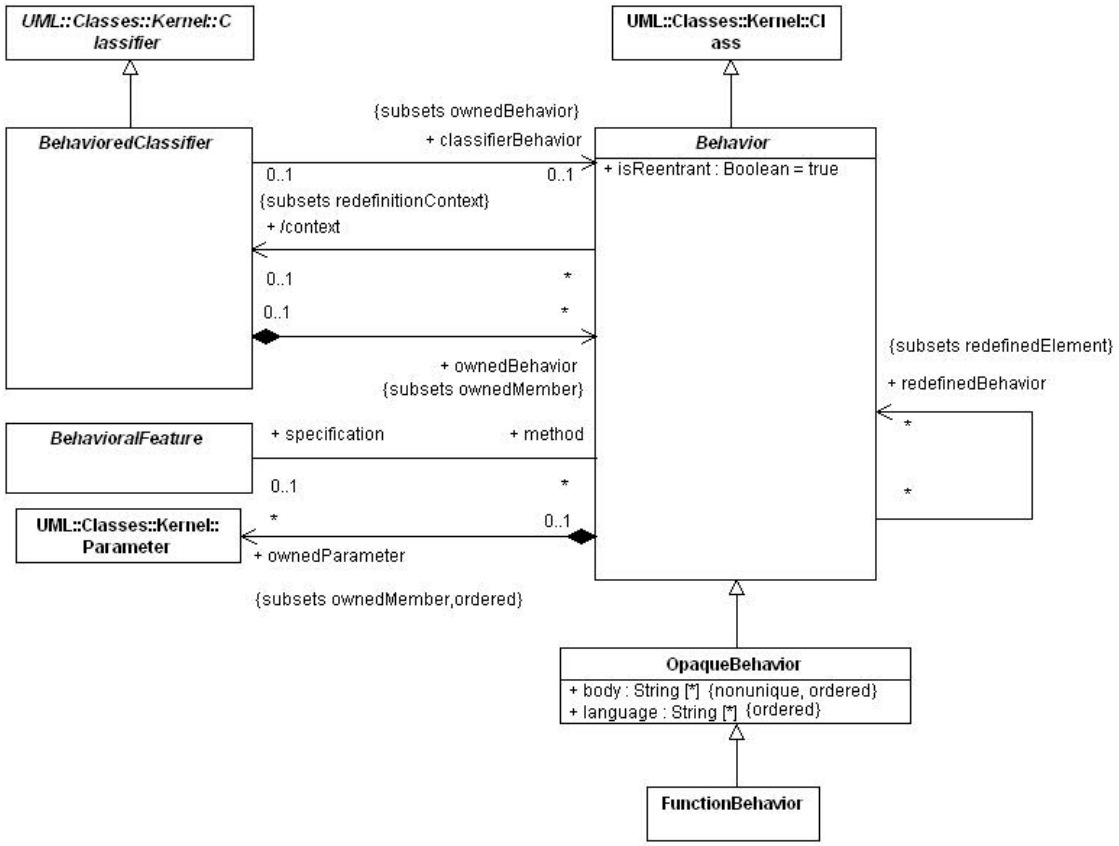
Diagram Summary
Common Behavior

Class Summary
Behavior
BehavioralFeature
BehavioredClassifier
FunctionBehavior
OpaqueBehavior
OpaqueExpression

Association Summary
A_behavior_opaqueExpression
A_classifierBehavior_behavioredClassifier
A_context_behavior
A_method_specification
A_ownedParameter_behavior
A_ownedParameter_behavioredClassifier
A_postcondition_behavior
A_precondition_behavior
A_redefinedBehavior_behavior
A_result_opaqueExpression

Package UML::CommonBehaviors::BasicBehaviors

Diagram Common Behavior



Classifiers Local to Package:

Behavior, BehavioralFeature, BehavioredClassifier, FunctionBehavior, OpaqueBehavior

Classifiers External to Package:

Class, Classifier, Parameter

Package UML::CommonBehaviors::BasicBehaviors

Class Behavior

Behavior is a specification of how its context classifier changes state over time. This specification may be either a definition of possible behavior execution or emergent behavior, or a selective illustration of an interesting subset of possible executions. The latter form is typically used for capturing examples, such as a trace of a particular execution.

Generalizations:

Class

Specializations:

Activity, Activity, Activity, Interaction, Interaction, OpaqueBehavior, StateMachine

Found in Diagrams:

Common Behavior, Component Wiring, Fundamental Nodes, Interactions, Structured Activities

Attributes

+ **isReentrant** : Boolean [1..1] = true

Tells whether the behavior can be invoked while it is still executing from a previous invocation.

Owned Association Ends

+ /**context** : BehavioredClassifier [0..1] {readOnly, subsets redefinitionContext}

The classifier that is the context for the execution of the behavior. If the behavior is owned by a BehavioredClassifier, that classifier is the context. Otherwise, the context is the first BehavioredClassifier reached by following the chain of owner relationships. For example, following this algorithm, the context of an entry action in a state machine is the classifier that owns the state machine. The features of the context classifier as well as the elements visible to the context classifier are visible to the behavior.

+ **ownedParameter** : Parameter [0..*] {ordered, subsets ownedMember}

References a list of parameters to the behavior which describes the order and type of arguments that can be given when the behavior is invoked and of the values which will be returned when the behavior completes its execution.

+ **postcondition** : Constraint [0..*] {subsets ownedRule}

An optional set of Constraints specifying what is fulfilled after the execution of the behavior is completed, if its precondition was fulfilled before its invocation.

+ **precondition** : Constraint [0..*] {subsets ownedRule}

Package UML::CommonBehaviors::BasicBehaviors

Class Behavior

An optional set of Constraints specifying what must be fulfilled when the behavior is invoked.

+ **redefinedBehavior** : Behavior [0..*] {subsets redefinedElement}

References a behavior that this behavior redefines. A subtype of Behavior may redefine any other subtype of Behavior. If the behavior implements a behavioral feature, it replaces the redefined behavior. If the behavior is a classifier behavior, it extends the redefined behavior.

+ **specification** : BehavioralFeature [0..1]

Designates a behavioral feature that the behavior implements. The behavioral feature must be owned by the classifier that owns the behavior or be inherited by it. The parameters of the behavioral feature and the implementing behavior must match. A behavior does not need to have a specification, in which case it either is the classifer behavior of a BehavioredClassifier or it can only be invoked by another behavior of the classifier.

Constraints

feature_of_context_classifier

The implemented behavioral feature must be a feature (possibly inherited) of the context classifier of the behavior.

expression (OCL): true

most_one_behaviour

There may be at most one behavior for a given pairing of classifier (as owner of the behavior) and behavioral feature (as specification of the behavior).

expression (OCL): true

must_realize

If the implemented behavioral feature has been redefined in the ancestors of the owner of the behavior, then the behavior must realize the latest redefining behavioral feature.

expression (OCL): true

parameters_match

The parameters of the behavior must match the parameters of the implemented behavioral feature.

expression (OCL): true

Package UML::CommonBehaviors::BasicBehaviors

Class BehavioralFeature

A behavioral feature is implemented (realized) by a behavior. A behavioral feature specifies that a classifier will respond to a designated request by invoking its implementing method.

Found in Diagrams:

Common Behavior

Attributes

+ **isAbstract** : Boolean [1..1] = false

If true, then the behavioral feature does not have an implementation, and one must be supplied by a more specific element. If false, the behavioral feature must have an implementation in the classifier or one must be inherited from a more general element.

Owned Association Ends

+ **method** : Behavior [0..*]

A behavioral description that implements the behavioral feature. There may be at most one behavior for a particular pairing of a classifier (as owner of the behavior) and a behavioral feature (as specification of the behavior).

Package UML::CommonBehaviors::BasicBehaviors

Class BehavioredClassifier

A classifier can have behavior specifications defined in its namespace. One of these may specify the behavior of the classifier itself.

Generalizations:

Classifier

Specializations:

Actor, Class, Collaboration, UseCase

Found in Diagrams:

Common Behavior, Reception

Owned Association Ends

+ **classifierBehavior** : Behavior [0..1] {subsets ownedBehavior}

A behavior specification that specifies the behavior of the classifier itself.

+ **ownedBehavior** : Behavior [0..*] {subsets ownedMember}

References behavior specifications owned by a classifier.

Constraints

class_behavior

If a behavior is classifier behavior, it does not have a specification.

expression (OCL): self.classifierBehavior->notEmpty() implies self.classifierBehavior.specification->isEmpty()

Package UML::CommonBehaviors::BasicBehaviors

Class FunctionBehavior

A function behavior is an opaque behavior that does not access or modify any objects or other external data.

Generalizations:

OpaqueBehavior

Found in Diagrams:

Common Behavior

Constraints

one_output_parameter

A function behavior has at least one output parameter.

expression (OCL): self.ownedParameters-> select(p | p.direction=#out or p.direction=#inout or p.direction=#return)->size() >= 1

types_of_parameters

The types of parameters are all data types, which may not nest anything but other datatypes.

expression (OCL): def: hasAllDataTypeAttributes(d : DataType) : Boolean = d.ownedAttribute->forAll(a | a.type.oclIsTypeOf(DataType) and hasAllDataTypeAttributes(a.type)) self.ownedParameters->forAll(p | p.type.notEmpty() and p.oclIsTypeOf(DataType) and hasAllDataTypeAttributes(p))

Package UML::CommonBehaviors::BasicBehaviors

Class OpaqueBehavior

An behavior with implementation-specific semantics.

Generalizations:

Behavior

Specializations:

FunctionBehavior

Found in Diagrams:

Common Behavior

Attributes

+ **body** : String [0..*] {ordered, nonunique}

  Specifies the behavior in one or more languages.

+ **language** : String [0..*] {ordered}

  Languages the body strings use in the same order as the body strings.

Package UML::CommonBehaviors::BasicBehaviors

Class OpaqueExpression

Provides a mechanism for precisely defining the behavior of an opaque expression. An opaque expression is defined by a behavior restricted to return one result.

Owned Association Ends

+ **behavior** : Behavior [0..1]

Specifies the behavior of the opaque expression.

+ /**result** : Parameter [0..1] {readOnly}

Restricts an opaque expression to return exactly one return result. When the invocation of the opaque expression completes, a single set of values is returned to its owner. This association is derived from the single return result parameter of the associated behavior.

Constraints

one_return_result_parameter

The behavior must have exactly one return result parameter.

expression (OCL): self.behavior.notEmpty() implies self.behavior.ownedParameter->select(p | p.direction=#return)->size() = 1

only_return_result_parameters

The behavior may only have return result parameters.

expression (OCL): self.behavior.notEmpty() implies self.behavior.ownedParameters->select(p | p.direction<>#return)->isEmpty()

Package UML::CommonBehaviors::BasicBehaviors

Association A_behavior_opaqueExpression

Member Ends:

behavior, opaqueExpression

Owned Association Ends

+ **opaqueExpression** : OpaqueExpression [0..*]

Package UML::CommonBehaviors::BasicBehaviors

Association A_classifierBehavior_behavioredClassifier

Member Ends:

classifierBehavior, behavioredClassifier

Found in Diagrams:

Common Behavior

Owned Association Ends

+ **behavioredClassifier** : BehavioredClassifier [0..1]

Package UML::CommonBehaviors::BasicBehaviors

Association A_context_behavior

Member Ends:

context, behavior

Found in Diagrams:

Common Behavior

Owned Association Ends

+ **behavior** : Behavior [0..*]

Package UML::CommonBehaviors::BasicBehaviors

Association A_method_specification

Member Ends:

method, specification

Found in Diagrams:

Common Behavior

Package UML::CommonBehaviors::BasicBehaviors

Association A_ownedParameter_behavior

Member Ends:

ownedParameter, behavior

Found in Diagrams:

Common Behavior

Owned Association Ends

+ **behavior** : Behavior [0..1]

Package UML::CommonBehaviors::BasicBehaviors

Association A_ownedParameter_behavioredClassifier

Member Ends:

ownedBehavior, behavioredClassifier

Found in Diagrams:

Common Behavior

Owned Association Ends

+ **behavioredClassifier** : BehavioredClassifier [0..1]

Package UML::CommonBehaviors::BasicBehaviors

Association A_postcondition_behavior

Member Ends:

postcondition, behavior

Owned Association Ends

+ **behavior** : Behavior [0..1] {subsets context}

Package UML::CommonBehaviors::BasicBehaviors

Association A_precondition_behavior

Member Ends:

precondition, behavior

Owned Association Ends

+ **behavior** : Behavior [0..1] {subsets context}

Package UML::CommonBehaviors::BasicBehaviors

Association A_redefinedBehavior_behavior

Member Ends:

redefinedBehavior, behavior

Found in Diagrams:

Common Behavior

Owned Association Ends

+ **behavior** : Behavior [0..*]

Package UML::CommonBehaviors::BasicBehaviors

Association A_result_opaqueExpression

Member Ends:

result, opaqueExpression

Owned Association Ends

+ **opaqueExpression** : OpaqueExpression [0..*]

Package UML::CommonBehaviors::Communications

Nesting Package:

CommonBehaviors

Merged Packages:

Interfaces

Diagram Summary
Events
Extensions to behavioral features
Reception

Class Summary
AnyReceiveEvent
BehavioralFeature
BehavioredClassifier
CallEvent
ChangeEvent
Class
Event
Interface
MessageEvent
Operation
Reception
Signal
SignalEvent
Trigger

Enumeration Summary
CallConcurrencyKind

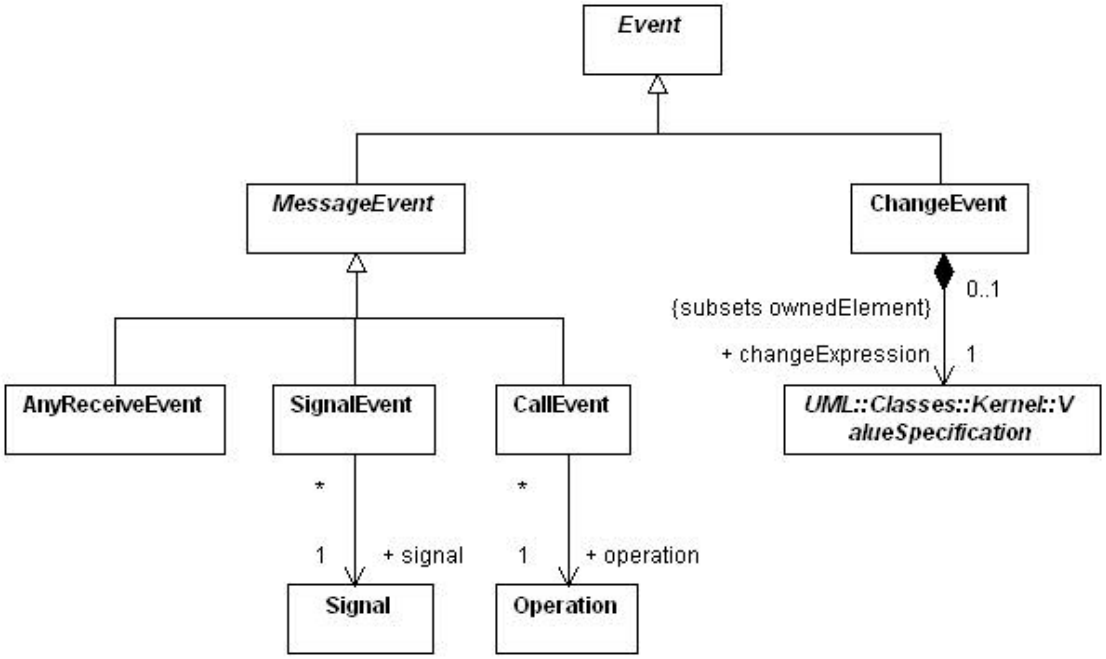
Association Summary
A_changeExpression_changeEvent
A_event_trigger
A_operation_callEvent
A_ownedAttribute_owningSignal
A_ownedReception_class
A_ownedReception_interface

Package UML::CommonBehaviors::Communications

A_ownedTrigger_behavioredClassifier
A_signal_reception
A_signal_signalEvent

Package UML::CommonBehaviors::Communications

Diagram Events



Classifiers Local to Package:

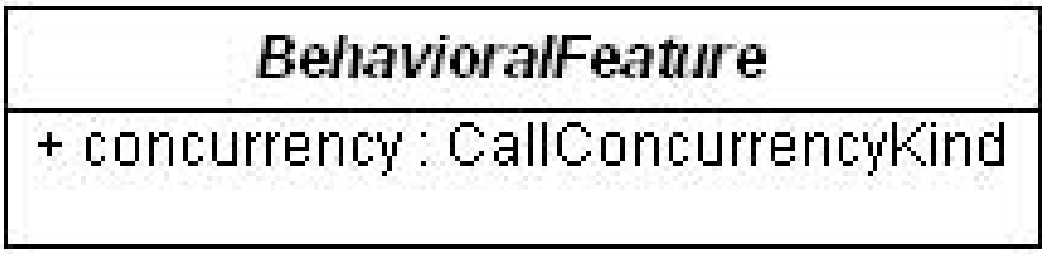
AnyReceiveEvent, CallEvent, ChangeEvent, Event, MessageEvent, Operation, Signal, SignalEvent

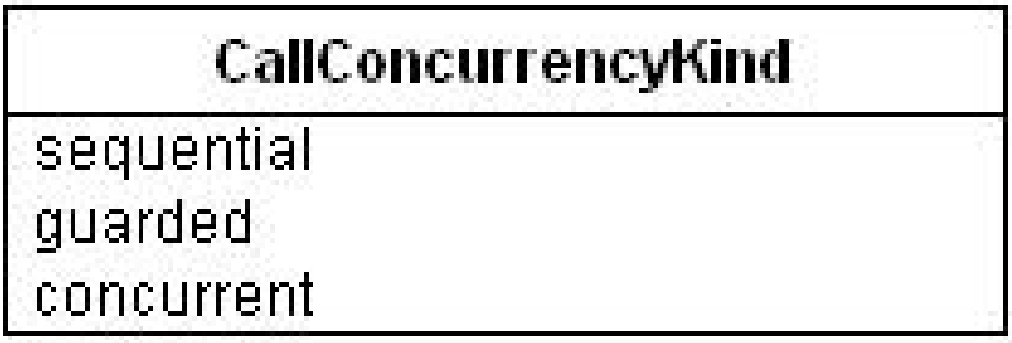
Classifiers External to Package:

ValueSpecification

Package UML::CommonBehaviors::Communications

Diagram Extensions to behavioral features



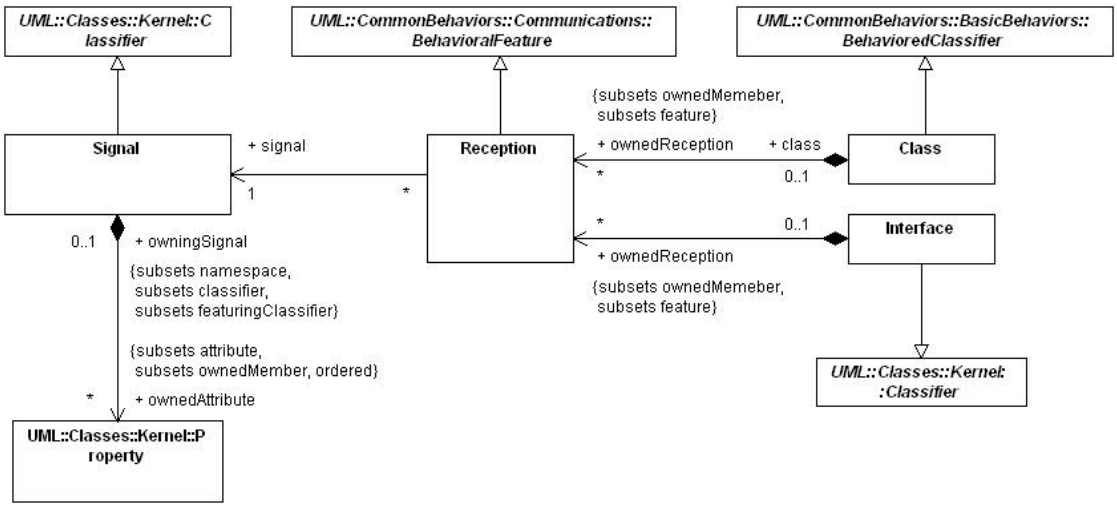


Classifiers Local to Package:

BehavioralFeature, CallConcurrencyKind

Package UML::CommonBehaviors::Communications

Diagram Reception



Classifiers Local to Package:

BehavioralFeature, Class, Interface, Reception, Signal

Classifiers External to Package:

BehavioredClassifier, Classifier, Property

Package UML::CommonBehaviors::Communications

Class AnyReceiveEvent

A trigger for an AnyReceiveEvent is triggered by the receipt of any message that is not explicitly handled by any related trigger.

Generalizations:

MessageEvent

Found in Diagrams:

Events

Package UML::CommonBehaviors::Communications

Class BehavioralFeature

A behavioral feature is implemented (realized) by a behavior. A behavioral feature specifies that a classifier will respond to a designated request by invoking its implementing method.

Specializations:

Reception

Found in Diagrams:

Extensions to behavioral features, Reception

Attributes

+ **concurrency** : CallConcurrencyKind [1..1] = sequential

Specifies the semantics of concurrent calls to the same passive instance (i.e., an instance originating from a class with isActive being false). Active instances control access to their own behavioral features.

Package UML::CommonBehaviors::Communications

Class BehavioredClassifier

A classifier can have behavior specifications defined in its namespace. One of these may specify the behavior of the classifier itself.

Generalizations:

Classifier

Owned Association Ends

+ **ownedTrigger** : Trigger [0..*] {subsets ownedMember}

References Trigger descriptions owned by a Classifier.

Package UML::CommonBehaviors::Communications

Class CallEvent

A call event models the receipt by an object of a message invoking a call of an operation.

Generalizations:

MessageEvent

Found in Diagrams:

Events

Owned Association Ends

+ **operation** : Operation [1..1]

Designates the operation whose invocation raised the call event.

Package UML::CommonBehaviors::Communications

Class ChangeEvent

A change event models a change in the system configuration that makes a condition true.

Generalizations:

Event

Found in Diagrams:

Events

Owned Association Ends

+ **changeExpression** : ValueSpecification [1..1] {subsets ownedElement}

A Boolean-valued expression that will result in a change event whenever its value changes from false to true.

Package UML::CommonBehaviors::Communications

Class Class

A class may be designated as active (i.e., each of its instances having its own thread of control) or passive (i.e., each of its instances executing within the context of some other object). A class may also specify which signals the instances of this class handle.

Generalizations:

BehavioredClassifier

Found in Diagrams:

Reception

Attributes

+ **isActive** : Boolean [1..1] = false

Determines whether an object specified by this class is active or not. If true, then the owning class is referred to as an active class. If false, then such a class is referred to as a passive class.

Owned Association Ends

+ **ownedReception** : Reception [0..*] {subsets ownedMember, subsets feature}

Receptions that objects of this class are willing to accept.

Constraints

passive_class

A passive class may not own receptions.

expression (OCL): not self.isActive implies self.ownedReception.isEmpty()

Package UML::CommonBehaviors::Communications

Class Event

An event is the specification of some occurrence that may potentially trigger effects by an object.

Generalizations:

PackageableElement

Specializations:

ChangeEvent, CreationEvent, DestructionEvent, ExecutionEvent, MessageEvent, TimeEvent

Found in Diagrams:

Events, Messages, Simple Time

Package UML::CommonBehaviors::Communications

Class Interface

Interfaces may include receptions (in addition to operations).

Generalizations:

Classifier

Found in Diagrams:

Reception

Owned Association Ends

+ **ownedReception** : Reception [0..*] {subsets feature, subsets ownedMember}

Receptions that objects providing this interface are willing to accept.

Package UML::CommonBehaviors::Communications

Class MessageEvent

A message event specifies the receipt by an object of either a call or a signal.

Generalizations:

Event

Specializations:

AnyReceiveEvent, CallEvent, ReceiveOperationEvent, ReceiveSignalEvent, SendOperationEvent, SendSignalEvent, SignalEvent

Found in Diagrams:

Events

Package UML::CommonBehaviors::Communications

Class Operation

An operation may invoke both the execution of method behaviors as well as other behavioral responses.

Found in Diagrams:

Events

Package UML::CommonBehaviors::Communications

Class Reception

A reception is a declaration stating that a classifier is prepared to react to the receipt of a signal. A reception designates a signal and specifies the expected behavioral response. The details of handling a signal are specified by the behavior associated with the reception or the classifier itself.

Generalizations:

BehavioralFeature

Found in Diagrams:

Reception

Owned Association Ends

+ **signal** : Signal [1..1]

The signal that this reception handles.

Constraints

not_query

A Reception can not be a query.

expression (OCL): not self.isQuery

Package UML::CommonBehaviors::Communications

Class Signal

A signal is a specification of send request instances communicated between objects. The receiving object handles the received request instances as specified by its receptions. The data carried by a send request (which was passed to it by the send invocation occurrence that caused that request) are represented as attributes of the signal. A signal is defined independently of the classifiers handling the signal occurrence.

Generalizations:

Classifier

Found in Diagrams:

Events, Reception

Owned Association Ends

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The attributes owned by the signal.

Package UML::CommonBehaviors::Communications

Class SignalEvent

A signal event represents the receipt of an asynchronous signal instance. A signal event may, for example, cause a state machine to trigger a transition.

Generalizations:

MessageEvent

Found in Diagrams:

Events

Owned Association Ends

+ **signal** : Signal [1..1]

The specific signal that is associated with this event.

Package UML::CommonBehaviors::Communications

Class Trigger

A trigger relates an event to a behavior that may affect an instance of the classifier.

Generalizations:

NamedElement

Owned Association Ends

+ **event** : Event [1..1]

The event that causes the trigger.

Package UML::CommonBehaviors::Communications

Enumeration CallConcurrencyKind

CallConcurrencyKind is an enumeration type.

Found in Diagrams:

Extensions to behavioral features

Enumeration Literals

concurrent

Multiple invocations of a behavioral feature may occur simultaneously to one instance and all of them may proceed concurrently.

guarded

Multiple invocations of a behavioral feature may occur simultaneously to one instance, but only one is allowed to commence. The others are blocked until the performance of the currently executing behavioral feature is complete. It is the responsibility of the system designer to ensure that deadlocks do not occur due to simultaneous blocks.

sequential

No concurrency management mechanism is associated with the operation and, therefore, concurrency conflicts may occur. Instances that invoke a behavioral feature need to coordinate so that only one invocation to a target on any behavioral feature occurs at once.

Package UML::CommonBehaviors::Communications

Association A_changeExpression_changeEvent

Member Ends:

changeExpression, changeEvent

Found in Diagrams:

Events

Owned Association Ends

+ **changeEvent** : ChangeEvent [0..1]

Package UML::CommonBehaviors::Communications

Association A_event_trigger

Member Ends:

event, trigger

Owned Association Ends

+ **trigger** : Trigger [0..*]

Package UML::CommonBehaviors::Communications

Association A_operation_callEvent

Member Ends:

operation, callEvent

Found in Diagrams:

Events

Owned Association Ends

+ **callEvent** : CallEvent [0..*]

Package UML::CommonBehaviors::Communications

Association A_ownedAttribute_owningSignal

Member Ends:

ownedAttribute, owningSignal

Found in Diagrams:

Reception

Owned Association Ends

+ **owningSignal** : Signal [0..1] {subsets namespace, subsets classifier, subsets featuringClassifier}

Package UML::CommonBehaviors::Communications

Association A_ownedReception_class

Member Ends:

ownedReception, class

Found in Diagrams:

Reception

Owned Association Ends

+ **class** : Class [0..1]

Package UML::CommonBehaviors::Communications

Association A_ownedReception_interface

Member Ends:

ownedReception, interface

Found in Diagrams:

Reception

Owned Association Ends

+ **interface** : Interface [0..1]

Package UML::CommonBehaviors::Communications

Association A_ownedTrigger_behavioredClassifier

Member Ends:

ownedTrigger, behavioredClassifier

Owned Association Ends

+ **behavioredClassifier** : BehavioredClassifier [0..1]

Package UML::CommonBehaviors::Communications

Association A_signal_reception

Member Ends:

signal, reception

Found in Diagrams:

Reception

Owned Association Ends

+ **reception** : Reception [0..*]

Package UML::CommonBehaviors::Communications

Association A_signal_signalEvent

Member Ends:

signal, signalEvent

Found in Diagrams:

Events

Owned Association Ends

+ **signalEvent** : SignalEvent [0..*]

Package UML::CommonBehaviors::SimpleTime

Nesting Package:

CommonBehaviors

Merged Packages:

IntermediateActions

Diagram Summary
Simple Time

Class Summary
Duration
DurationConstraint
DurationInterval
DurationObservation
Interval
IntervalConstraint
Observation
TimeConstraint
TimeEvent
TimeExpression
TimeInterval
TimeObservation

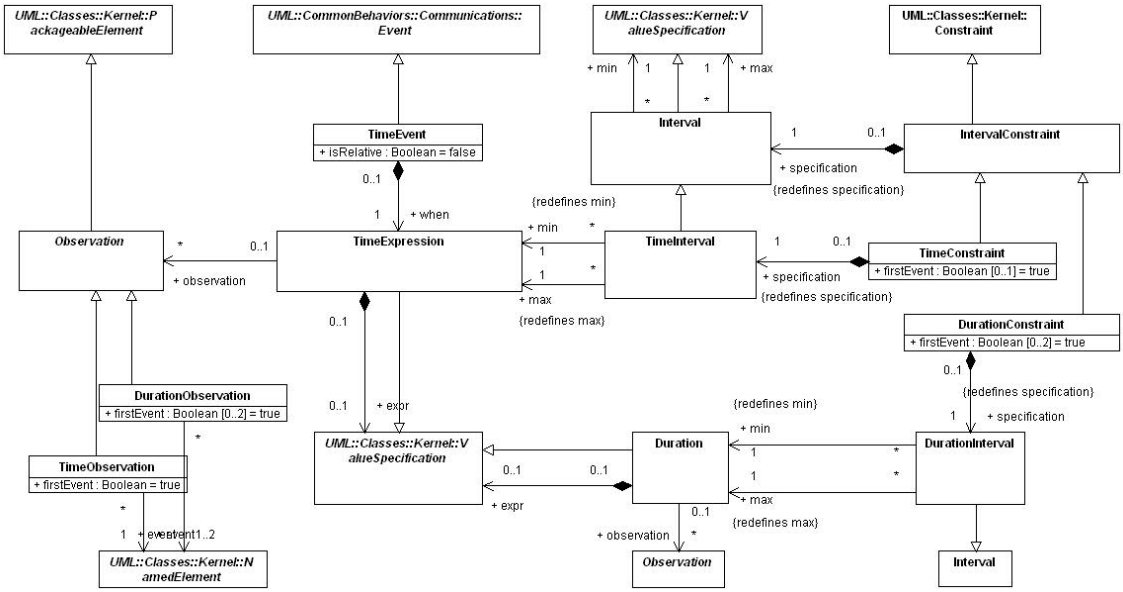
Association Summary
A_event_durationObservation
A_event_timeObservation
A_expr_duration
A_expr_timeExpression
A_max_durationInterval
A_max_interval
A_max_timeInterval
A_min_durationInterval
A_min_interval
A_min_timeInterval
A_observation_duration
A_observation_timeExpression
A_specification_durationConstraint

Package UML::CommonBehaviors::SimpleTime

A_specification_intervalConstraint
A_specification_timeConstraint
A_when_timeEvent

Package UML::CommonBehaviors::SimpleTime

Diagram Simple Time



Classifiers Local to Package:

Duration, DurationConstraint, DurationInterval, DurationObservation, Interval, IntervalConstraint, Observation, TimeConstraint, TimeEvent, TimeExpression, TimeInterval, TimeObservation

Classifiers External to Package:

Constraint, Event, NamedElement, PackageableElement, ValueSpecification

Package UML::CommonBehaviors::SimpleTime

Class Duration

Duration defines a value specification that specifies the temporal distance between two time instants.

Generalizations:

ValueSpecification

Found in Diagrams:

Simple Time

Owned Association Ends

+ **expr** : ValueSpecification [0..1]

The value of the Duration.

+ **observation** : Observation [0..*]

Refers to the time and duration observations that are involved in expr.

Package UML::CommonBehaviors::SimpleTime

Class DurationConstraint

A duration constraint is a constraint that refers to a duration interval.

Generalizations:

IntervalConstraint

Found in Diagrams:

Simple Time

Attributes

+ **firstEvent** : Boolean [0..2] = true

The value of firstEvent[i] is related to constrainedElement[i] (where i is 1 or 2). If firstEvent[i] is true, then the corresponding observation event is the first time instant the execution enters constrainedElement[i]. If firstEvent[i] is false, then the corresponding observation event is the last time instant the execution is within constrainedElement[i]. Default value is true applied when constrainedElement[i] refers an element that represents only one time instant.

Owned Association Ends

+ **specification** : DurationInterval [1..1] {redefines specification}

The interval constraining the duration.

Constraints

first_event_multiplicity

The multiplicity of firstEvent must be 2 if the multiplicity of constrainedElement is 2. Otherwise the multiplicity of firstEvent is 0.

expression (OCL): if (constrainedElement->size() =2) then (firstEvent->size() = 2) else (firstEvent ->size() = 0)

Package UML::CommonBehaviors::SimpleTime

Class DurationInterval

A duration interval defines the range between two durations.

Generalizations:

Interval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **max** : Duration [1..1] {redefines max}

Refers to the Duration denoting the maximum value of the range.

+ **min** : Duration [1..1] {redefines min}

Refers to the Duration denoting the minimum value of the range.

Package UML::CommonBehaviors::SimpleTime

Class DurationObservation

A duration observation is a reference to a duration during an execution. It points out the element(s) in the model to observe and whether the observations are when this model element is entered or when it is exited.

Generalizations:

Observation

Found in Diagrams:

Simple Time

Attributes

+ **firstEvent** : Boolean [0..2] = true

The value of firstEvent[i] is related to event[i] (where i is 1 or 2). If firstEvent[i] is true, then the corresponding observation event is the first time instant the execution enters event[i]. If firstEvent [i] is false, then the corresponding observation event is the time instant the execution exits event[i]. Default value is true applied when event[i] refers an element that represents only one time instant.

Owned Association Ends

+ **event** : NamedElement [1..2]

The observation is determined by the entering or exiting of the event element during execution.

Constraints

first_event_multiplicity

The multiplicity of firstEvent must be 2 if the multiplicity of event is 2. Otherwise the multiplicity of firstEvent is 0.

expression (OCL): if (event->size() = 2) then (firstEvent->size() = 2) else (firstEvent->size() = 0)

Package UML::CommonBehaviors::SimpleTime

Class Interval

An interval defines the range between two value specifications.

Generalizations:

ValueSpecification

Specializations:

DurationInterval, TimeInterval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **max** : ValueSpecification [1..1]

Refers to the ValueSpecification denoting the maximum value of the range.

+ **min** : ValueSpecification [1..1]

Refers to the ValueSpecification denoting the minimum value of the range.

Package UML::CommonBehaviors::SimpleTime

Class IntervalConstraint

An interval constraint is a constraint that refers to an interval.

Generalizations:

Constraint

Specializations:

DurationConstraint, TimeConstraint

Found in Diagrams:

Simple Time

Owned Association Ends

+ **specification** : Interval [1..1] {redefines specification}

A condition that must be true when evaluated in order for the constraint to be satisfied.

Package UML::CommonBehaviors::SimpleTime

Class Observation

Observation is a superclass of TimeObservation and DurationObservation in order for TimeExpression and Duration to refer to either in a simple way.

Generalizations:

PackageableElement

Specializations:

DurationObservation, TimeObservation

Found in Diagrams:

Simple Time

Package UML::CommonBehaviors::SimpleTime

Class TimeConstraint

A time constraint is a constraint that refers to a time interval.

Generalizations:

IntervalConstraint

Found in Diagrams:

Simple Time

Attributes

+ **firstEvent** : Boolean [0..1] = true

The value of firstEvent is related to constrainedElement. If firstEvent is true, then the corresponding observation event is the first time instant the execution enters constrainedElement. If firstEvent is false, then the corresponding observation event is the last time instant the execution is within constrainedElement.

Owned Association Ends

+ **specification** : TimeInterval [1..1] {redefines specification}

A condition that must be true when evaluated in order for the constraint to be satisfied.

Package UML::CommonBehaviors::SimpleTime

Class TimeEvent

A time event specifies a point in time. At the specified time, the event occurs.

Generalizations:

Event

Found in Diagrams:

Simple Time

Attributes

+ **isRelative** : Boolean [1..1] = false

Specifies whether it is relative or absolute time.

Owned Association Ends

+ **when** : TimeExpression [1..1]

Specifies the corresponding time deadline.

Constraints

when_non_negative

The ValueSpecification when must return a non-negative Integer.

expression (OCL): true

Package UML::CommonBehaviors::SimpleTime

Class TimeExpression

A time expression defines a value specification that represents a time value.

Generalizations:

ValueSpecification

Found in Diagrams:

Simple Time

Owned Association Ends

+ **expr** : ValueSpecification [0..1]

The value of the time expression.

+ **observation** : Observation [0..*]

Refers to the time and duration observations that are involved in expr.

Package UML::CommonBehaviors::SimpleTime

Class TimeInterval

A time interval defines the range between two time expressions.

Generalizations:

Interval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **max** : TimeExpression [1..1] {redefines max}

Refers to the TimeExpression denoting the maximum value of the range.

+ **min** : TimeExpression [1..1] {redefines min}

Refers to the TimeExpression denoting the minimum value of the range.

Package UML::CommonBehaviors::SimpleTime

Class TimeObservation

A time observation is a reference to a time instant during an execution. It points out the element in the model to observe and whether the observation is when this model element is entered or when it is exited.

Generalizations:

Observation

Found in Diagrams:

Simple Time

Attributes

+ **firstEvent** : Boolean [1..1] = true

The value of firstEvent is related to event. If firstEvent is true, then the corresponding observation event is the first time instant the execution enters event. If firstEvent is false, then the corresponding observation event is the time instant the execution exits event.

Owned Association Ends

+ **event** : NamedElement [1..1]

The observation is determined by the entering or exiting of the event element during execution.

Package UML::CommonBehaviors::SimpleTime

Association A_event_durationObservation

Member Ends:

event, durationObservation

Found in Diagrams:

Simple Time

Owned Association Ends

+ **durationObservation** : DurationObservation [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_event_timeObservation

Member Ends:

event, timeObservation

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeObservation** : TimeObservation [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_expr_duration

Member Ends:

expr, duration

Found in Diagrams:

Simple Time

Owned Association Ends

+ **duration** : Duration [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_expr_timeExpression

Member Ends:

expr, timeExpression

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeExpression** : TimeExpression [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_max_durationInterval

Member Ends:

max, durationInterval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **durationInterval** : DurationInterval [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_max_interval

Member Ends:

max, interval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **interval** : Interval [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_max_timeInterval

Member Ends:

max, timeInterval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeInterval** : TimeInterval [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_min_durationInterval

Member Ends:

min, durationInterval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **durationInterval** : DurationInterval [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_min_interval

Member Ends:

min, interval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **interval** : Interval [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_min_timeInterval

Member Ends:

min, timeInterval

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeInterval** : TimeInterval [0..*]

Package UML::CommonBehaviors::SimpleTime

Association A_observation_duration

Member Ends:

observation, duration

Found in Diagrams:

Simple Time

Owned Association Ends

+ **duration** : Duration [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_observation_timeExpression

Member Ends:

observation, timeExpression

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeExpression** : TimeExpression [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_specification_durationConstraint

Member Ends:

specification, durationConstraint

Found in Diagrams:

Simple Time

Owned Association Ends

+ **durationConstraint** : DurationConstraint [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_specification_intervalConstraint

Member Ends:

specification, intervalConstraint

Found in Diagrams:

Simple Time

Owned Association Ends

+ **intervalConstraint** : IntervalConstraint [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_specification_timeConstraint

Member Ends:

specification, timeConstraint

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeConstraint** : TimeConstraint [0..1]

Package UML::CommonBehaviors::SimpleTime

Association A_when_timeEvent

Member Ends:

when, timeEvent

Found in Diagrams:

Simple Time

Owned Association Ends

+ **timeEvent** : TimeEvent [0..1]

Package UML::Components

Nesting Package:

UML

Imported Packages:

CompositeStructures

Nested Package Summary
BasicComponents
PackagingComponents

Package UML::Components::BasicComponents

Nesting Package:

Components

Imported Packages:

Ports

Merged Packages:

Dependencies, StructuredClasses

Diagram Summary
Component Construct
Component Wiring

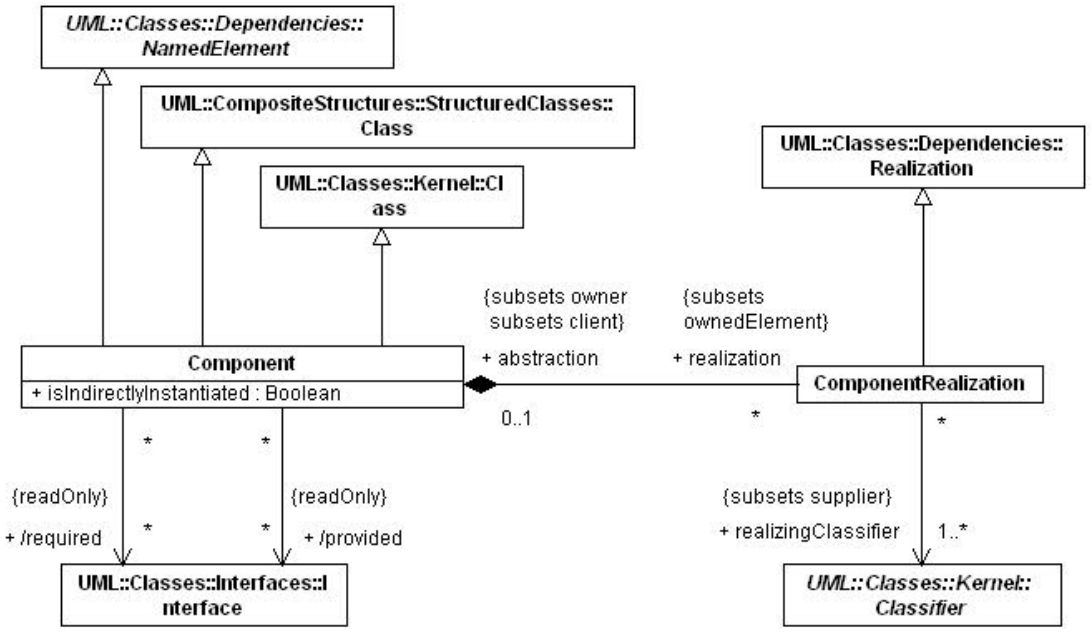
Class Summary
Component
ComponentRealization
Connector
ConnectorEnd

Enumeration Summary
ConnectorKind

Association Summary
A_contract_connector
A_end_connector
A_partWithPort_connectorEnd
A_provided_component
A_realization_abstraction
A_realizingClassifier_componentRealization
A_required_component
A_role_connectorEnd

Package UML::Components::BasicComponents

Diagram Component Construct



Classifiers Local to Package:

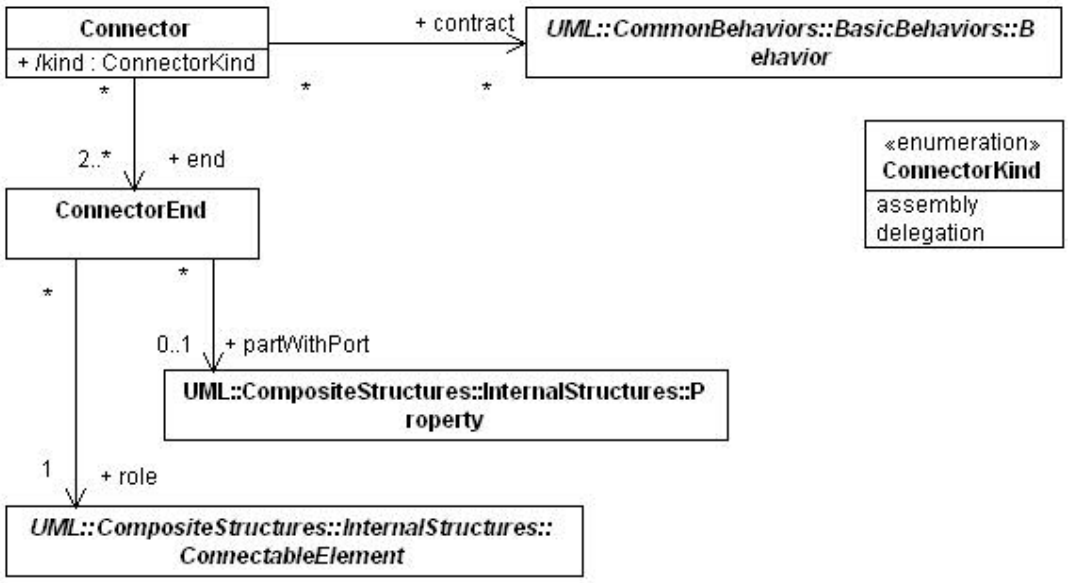
Component, ComponentRealization

Classifiers External to Package:

Class, Class, Classifier, Interface, NamedElement, Realization

Package UML::Components::BasicComponents

Diagram Component Wiring



Classifiers Local to Package:

Connector, ConnectorEnd, ConnectorKind

Classifiers External to Package:

Behavior, ConnectableElement, Property

Package UML::Components::BasicComponents

Class Component

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment.

Generalizations:

Class, Class, NamedElement

Found in Diagrams:

Component Construct

Attributes

+ **isIndirectlyInstantiated** : Boolean [1..1] = true

isIndirectlyInstantiated : Boolean {default = true} The kind of instantiation that applies to a Component. If false, the component is instantiated as an addressable object. If true, the Component is defined at design-time, but at run-time (or execution-time) an object specified by the Component does not exist, that is, the component is instantiated indirectly, through the instantiation of its realizing classifiers or parts. Several standard stereotypes use this meta attribute (e.g., «specification», «focus», «subsystem»).

Owned Association Ends

+ /**provided** : Interface [0..*] {readOnly}

The interfaces that the component exposes to its environment. These interfaces may be Realized by the Component or any of its realizingClassifiers, or they may be the Interfaces that are provided by its public Ports.

+ **realization** : ComponentRealization [0..*] {subsets ownedElement}

The set of Realizations owned by the Component. Realizations reference the Classifiers of which the Component is an abstraction; i.e., that realize its behavior.

+ /**required** : Interface [0..*] {readOnly}

The interfaces that the component requires from other components in its environment in order to be able to offer its full set of provided functionality. These interfaces may be used by the Component or any of its realizingClassifiers, or they may be the Interfaces that are required by its public Ports.

Operations

+ **provided** () : Interface [0..*] {query}

body (OCL): result = let realizedInterfaces : Set(Interface) = RealizedInterfaces(self) ,

Package UML::Components::BasicComponents

Class Component

realizingClassifiers : Set(Classifier) = Set{self.realizingClassifier}->union(self.allParents(). realizingClassifier), allRealizingClassifiers : Set(Classifier) = realizingClassifiers->union (realizingClassifiers.allParents()) , realizingClassifierInterfaces : Set(Interface) = allRealizingClassifiers->iterate(c; rci : Set(Interface) = Set{} | rci->union(RealizedInterfaces(c))) , ports : Set(Port) = self.ownedPort->union(allParents.oclAsType(Set(EncapsulatedClassifier)). ownedPort) , providedByPorts : Set(Interface) = ports.provided in realizedInterfaces->union (realizingClassifierInterfaces) ->union(providedByPorts)->asSet()

+ **realizedInterfaces** (classifier : Classifier) : Interface [0..*] {query}

Utility returning the set of realized interfaces of a component.

body (OCL): result = (classifier.clientDependency-> select(dependency|dependency.oclIsKindOf (Realization) and dependency.supplier.oclIsKindOf(Interface)))-> collect(dependency|dependency. client)

+ **required** () : Interface [0..*] {query}

body (OCL): result = let usedInterfaces : Set(Interface) = UsedInterfaces(self), realizingClassifiers : Set(Classifier) = Set{self.realizingClassifier}->union(self.allParents(). realizingClassifier), allRealizingClassifiers : Set(Classifier) = realizingClassifiers->union (realizingClassifiers.allParents()), realizingClassifierInterfaces : Set(Interface) = allRealizingClassifiers->iterate(c; rci : Set(Interface) = Set{} | rci->union(UsedInterfaces(c))), ports : Set(Port) = self.ownedPort->union(allParents.oclAsType(Set(EncapsulatedClassifier)). ownedPort), usedByPorts : Set(Interface) = ports.required in usedInterfaces->union (realizingClassifierInterfaces) ->union(usedByPorts)->asSet()

+ **usedInterfaces** (classifier : Classifier) : Interface [0..*] {query}

Utility returning the set of used interfaces of a component.

body (OCL): result = (classifier.supplierDependency-> select(dependency|dependency.oclIsKindOf (Usage) and dependency.supplier.oclIsKindOf(interface)))-> collect(dependency|dependency. supplier)

Constraints

no_nested_classifiers

A component cannot nest classifiers.

expression (OCL): self.nestedClassifier->isEmpty()

Package UML::Components::BasicComponents

Class ComponentRealization

The realization concept is specialized to (optionally) define the classifiers that realize the contract offered by a component in terms of its provided and required interfaces. The component forms an abstraction from these various classifiers.

Generalizations:

Realization

Found in Diagrams:

Component Construct

Owned Association Ends

+ **abstraction** : Component [0..1] {subsets owner, subsets supplier}

The Component that owns this ComponentRealization and which is implemented by its realizing classifiers.

+ **realizingClassifier** : Classifier [1..*] {subsets client}

The classifiers that are involved in the implementation of the Component that owns this Realization.

Package UML::Components::BasicComponents

Class Connector

A delegation connector is a connector that links the external contract of a component (as specified by its ports) to the realization of that behavior. It represents the forwarding of events (operation requests and events): a signal that arrives at a port that has a delegation connector to one or more parts or ports on parts will be passed on to those targets for handling.
An assembly connector is a connector between two or more parts or ports on parts that defines that one or more parts provide the services that other parts use.

Found in Diagrams:

Component Wiring

Attributes

+ /**kind** : ConnectorKind [1..1]

Indicates the kind of connector. This is derived: a connector with one or more ends connected to a Port which is not on a Part and which is not a behavior port is a delegation; otherwise it is an assembly.

Owned Association Ends

+ **contract** : Behavior [0..*]

The set of Behaviors that specify the valid interaction patterns across the connector.

+ **end** : ConnectorEnd [2..*]

Operations

+ **kind** () : ConnectorKind [1..1] {query}

body (OCL): result = if end->exists(role.oclIsKindOf(Port) and partWithPort->isEmpty() and not role.oclAsType(Port).isBehavior) then ConnectorKind::delegation else ConnectorKind::assembly endif

Constraints

between_interfaces_ports

Each feature of each of the required interfaces of each Port or Part at the end of a connector must have at least one compatible feature among the features of the provided interfaces of Ports or Parts at the other ends, where the required set of (interface) features of a delegating port from the context of the delegating connector is the set of features that exist in the port's provided interfaces, and the provided set of (interface) features of a delegating port from the context of the delegating connector is the set of features that exist in the port's required interfaces.

Package UML::Components::BasicComponents

Class Connector

expression (OCL): true

Package UML::Components::BasicComponents

Class ConnectorEnd

Found in Diagrams:

Component Wiring

Owned Association Ends

+ **partWithPort** : Property [0..1]

+ **role** : ConnectableElement [1..1]

Package UML::Components::BasicComponents

Enumeration ConnectorKind

ConnectorKind is an enumeration type.

Found in Diagrams:

Component Wiring

Enumeration Literals

assembly

Indicates that the connector is an assembly connector.

delegation

Indicates that the connector is a delegation connector.

Package UML::Components::BasicComponents

Association A_contract_connector

Member Ends:

contract, connector

Found in Diagrams:

Component Wiring

Owned Association Ends

+ **connector** : Connector [0..*]

Package UML::Components::BasicComponents

Association A_end_connector

Member Ends:

end, connector

Found in Diagrams:

Component Wiring

Owned Association Ends

+ **connector** : Connector [0..*]

Package UML::Components::BasicComponents

Association A_partWithPort_connectorEnd

Member Ends:

partWithPort, connectorEnd

Found in Diagrams:

Component Wiring

Owned Association Ends

+ **connectorEnd** : ConnectorEnd [0..*]

Package UML::Components::BasicComponents

Association A_provided_component

Member Ends:

provided, component

Found in Diagrams:

Component Construct

Owned Association Ends

+ **component** : Component [0..*]

Package UML::Components::BasicComponents

Association A_realization_abstraction

Member Ends:

realization, abstraction

Found in Diagrams:

Component Construct

Package UML::Components::BasicComponents

Association A_realizingClassifier_componentRealization

Member Ends:

realizingClassifier, componentRealization

Found in Diagrams:

Component Construct

Owned Association Ends

+ **componentRealization** : ComponentRealization [0..*]

Package UML::Components::BasicComponents

Association A_required_component

Member Ends:

required, component

Found in Diagrams:

Component Construct

Owned Association Ends

+ **component** : Component [0..*]

Package UML::Components::BasicComponents

Association A_role_connectorEnd

Member Ends:

role, connectorEnd

Found in Diagrams:

Component Wiring

Owned Association Ends

+ **connectorEnd** : ConnectorEnd [0..*]

Package UML::Components::PackagingComponents

Nesting Package:

Components

Merged Packages:

BasicComponents

Diagram Summary

Packaging Components

Class Summary

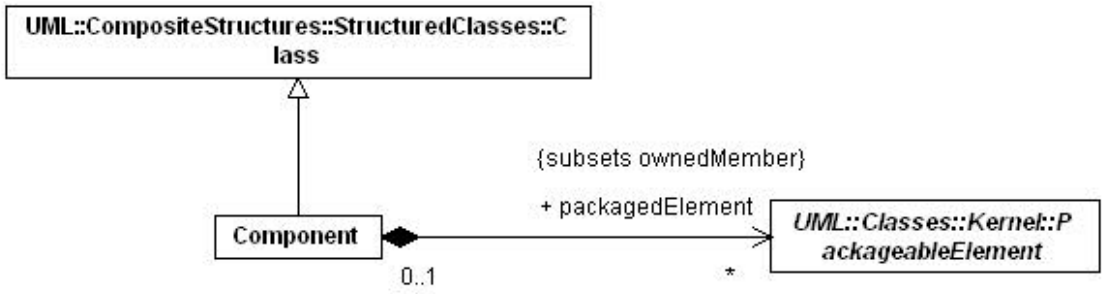
Component

Association Summary

A_packagedElement_component

Package UML::Components::PackagingComponents

Diagram Packaging Components



Classifiers Local to Package:

Component

Classifiers External to Package:

Class, PackageableElement

Package UML::Components::PackagingComponents

Class Component

In the namespace of a component, all model elements that are involved in or related to its definition are either owned or imported explicitly. This may include, for example, use cases and dependencies (e.g. mappings), packages, components, and artifacts.

Generalizations:

Class

Found in Diagrams:

Packaging Components

Owned Association Ends

+ **packagedElement** : PackageableElement [0..*] {subsets ownedMember}

The set of PackageableElements that a Component owns. In the namespace of a component, all model elements that are involved in or related to its definition may be owned or imported explicitly. These may include e.g. Classes, Interfaces, Components, Packages, Use cases, Dependencies (e.g. mappings), and Artifacts.

Constraints

no_packaged_elements

component nested in a Class cannot have any packaged elements.

expression (OCL): (not self.class->isEmpty()) implies self.packagedElement->isEmpty()

Package UML::Components::PackagingComponents

Association A_packagedElement_component

Member Ends:

packagedElement, component

Found in Diagrams:

Packaging Components

Owned Association Ends

+ **component** : Component [0..1]

Package UML::CompositeStructures

Nesting Package:

UML

Imported Packages:

Classes

Nested Package Summary
Collaborations
InternalStructures
InvocationActions
Ports
StructuredActivities
StructuredClasses

Package UML::CompositeStructures::Collaborations

Nesting Package:

CompositeStructures

Merged Packages:

InternalStructures

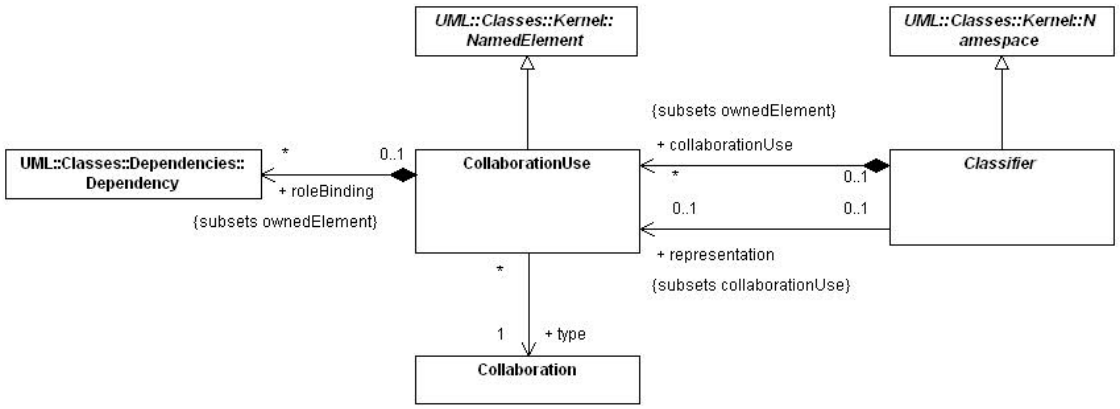
Diagram Summary
Collaboration Use and Role Binding

Class Summary
Classifier
Collaboration
CollaborationUse
Parameter

Association Summary
A_collaborationRole_collaboration
A_collaborationUse_classifier
A_representation_classifier
A_roleBinding_collaborationUse
A_type_collaborationUse

Package UML::CompositeStructures::Collaborations

Diagram Collaboration Use and Role Binding



Classifiers Local to Package:

Classifier, Collaboration, CollaborationUse

Classifiers External to Package:

Dependency, NamedElement, Namespace

Package UML::CompositeStructures::Collaborations

Class Classifier

A classifier has the capability to own collaboration uses. These collaboration uses link a collaboration with the classifier to give a description of the workings of the classifier.

Generalizations:

Namespace

Specializations:

Collaboration

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **collaborationUse** : CollaborationUse [0..*] {subsets ownedElement}

References the collaboration uses owned by the classifier.

+ **representation** : CollaborationUse [0..1] {subsets collaborationUse}

References a collaboration use which indicates the collaboration that represents this classifier.

Package UML::CompositeStructures::Collaborations

Class Collaboration

A collaboration use represents the application of the pattern described by a collaboration to a specific situation involving specific classes or instances playing the roles of the collaboration.

Generalizations:

BehavioredClassifier, Classifier, StructuredClassifier

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **collaborationRole** : ConnectableElement [0..*] {subsets role}

References connectable elements (possibly owned by other classifiers) which represent roles that instances may play in this collaboration.

Package UML::CompositeStructures::Collaborations

Class CollaborationUse

A collaboration use represents one particular use of a collaboration to explain the relationships between the properties of a classifier. A collaboration use shows how the pattern described by a collaboration is applied in a given context, by binding specific entities from that context to the roles of the collaboration. Depending on the context, these entities could be structural features of a classifier, instance specifications, or even roles in some containing collaboration. There may be multiple occurrences of a given collaboration within a classifier, each involving a different set of roles and connectors. A given role or connector may be involved in multiple occurrences of the same or different collaborations. Associated dependencies map features of the collaboration type to features in the classifier. These dependencies indicate which role in the classifier plays which role in the collaboration.

Generalizations:

NamedElement

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **roleBinding** : Dependency [0..*] {subsets ownedElement}

A mapping between features of the collaboration type and features of the owning classifier. This mapping indicates which connectable element of the classifier plays which role(s) in the collaboration. A connectable element may be bound to multiple roles in the same collaboration use (that is, it may play multiple roles).

+ **type** : Collaboration [1..1]

The collaboration which is used in this occurrence. The collaboration defines the cooperation between its roles which are mapped to properties of the classifier owning the collaboration use.

Constraints

client_elements

All the client elements of a roleBinding are in one classifier and all supplier elements of a roleBinding are in one collaboration and they are compatible.

expression (OCL): true

connectors

The connectors in the classifier connect according to the connectors in the collaboration

expression (OCL): true

every_role

Package UML::CompositeStructures::Collaborations

Class CollaborationUse

Every role in the collaboration is bound within the collaboration use to a connectable element within the owning classifier.

expression (OCL): true

Package UML::CompositeStructures::Collaborations

Class Parameter

Parameters are allowed to be treated as connectable elements.

Generalizations:

ConnectableElement

Constraints

connector_end

A parameter may only be associated with a connector end within the context of a collaboration.

expression (OCL): self.end.notEmpty() implies self.collaboration.notEmpty()

Package UML::CompositeStructures::Collaborations

Association A_collaborationRole_collaboration

Member Ends:

collaborationRole, collaboration

Owned Association Ends

+ **collaboration** : Collaboration [0..*]

Package UML::CompositeStructures::Collaborations

Association A_collaborationUse_classifier

Member Ends:

collaborationUse, classifier

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **classifier** : Classifier [0..1]

Package UML::CompositeStructures::Collaborations

Association A_representation_classifier

Member Ends:

representation, classifier

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **classifier** : Classifier [0..1]

Package UML::CompositeStructures::Collaborations

Association A_roleBinding_collaborationUse

Member Ends:

roleBinding, collaborationUse

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **collaborationUse** : CollaborationUse [0..1]

Package UML::CompositeStructures::Collaborations

Association A_type_collaborationUse

Member Ends:

type, collaborationUse

Found in Diagrams:

Collaboration Use and Role Binding

Owned Association Ends

+ **collaborationUse** : CollaborationUse [0..*]

Package UML::CompositeStructures::InternalStructures

Nesting Package:

CompositeStructures

Merged Packages:

Interfaces

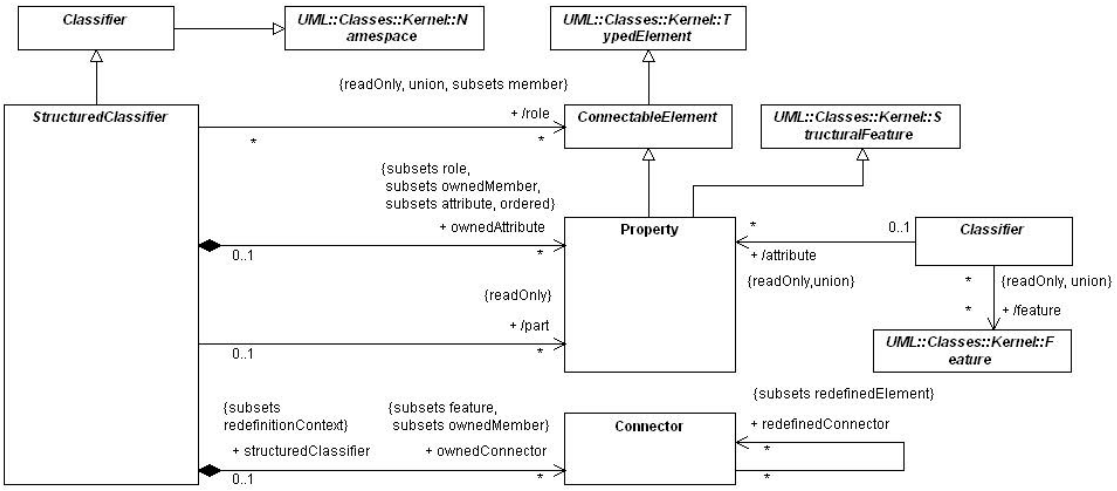
Diagram Summary
Structured Classifier

Class Summary
Classifier
ConnectableElement
Connector
ConnectorEnd
Property
StructuredClassifier

Association Summary
A_attribute_classifier
A_definingEnd_connectorEnd
A_end_connector
A_end_role
A_feature_classifier
A_ownedAttribute_structuredClassifier
A_ownedConnector_structuredClassifier
A_part_structuredClassifier
A_redefinedConnector_connector
A_role_structuredClassifier
A_type_connector

Package UML::CompositeStructures::InternalStructures

Diagram Structured Classifier



Classifiers Local to Package:

Classifier, ConnectableElement, Connector, Property, StructuredClassifier

Classifiers External to Package:

Feature, Namespace, StructuralFeature, TypedElement

Package UML::CompositeStructures::InternalStructures

Class Classifier

A classifier has the capability to own collaboration uses. These collaboration uses link a collaboration with the classifier to give a description of the workings of the classifier.

Generalizations:

Namespace

Specializations:

StructuredClassifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ /**attribute** : Property [0..*] {readOnly, union}

Refers to all of the Properties that are direct (i.e. not inherited or imported) attributes of the classifier.

+ /**feature** : Feature [0..*] {readOnly, union}

Package UML::CompositeStructures::InternalStructures

Class ConnectableElement

ConnectableElement is an abstract metaclass representing a set of instances that play roles of a classifier. Connectable elements may be joined by attached connectors and specify configurations of linked instances to be created within an instance of the containing classifier.

Generalizations:

TypedElement

Specializations:

Parameter, Property, Variable

Found in Diagrams:

Component Wiring, Lifelines, Structured Classifier

Owned Association Ends

+ /**end** : ConnectorEnd [0..*] {ordered}

Denotes a connector that attaches to this connectable element.

Operations

+ **end** () : ConnectorEnd [0..*]

body (OCL): result = ConnectorEnd.allInstances()->select(e | e.role=self)

Package UML::CompositeStructures::InternalStructures

Class Connector

Specifies a link that enables communication between two or more instances. This link may be an instance of an association, or it may represent the possibility of the instances being able to communicate because their identities are known by virtue of being passed in as parameters, held in variables or slots, or because the communicating instances are the same instance. The link may be realized by something as simple as a pointer or by something as complex as a network connection. In contrast to associations, which specify links between any instance of the associated classifiers, connectors specify links between instances playing the connected parts only.

Generalizations:

Feature

Found in Diagrams:

Messages, Structured Classifier

Owned Association Ends

+ **end** : ConnectorEnd [2..*] {ordered, subsets ownedElement}

A connector consists of at least two connector ends, each representing the participation of instances of the classifiers typing the connectable elements attached to this end. The set of connector ends is ordered.

+ **redefinedConnector** : Connector [0..*] {subsets redefinedElement}

A connector may be redefined when its containing classifier is specialized. The redefining connector may have a type that specializes the type of the redefined connector. The types of the connector ends of the redefining connector may specialize the types of the connector ends of the redefined connector. The properties of the connector ends of the redefining connector may be replaced.

+ **type** : Association [0..1]

An optional association that specifies the link corresponding to this connector.

Constraints

compatible

The connectable elements attached to the ends of a connector must be compatible.

expression (OCL): true

roles

The ConnectableElements attached as roles to each ConnectorEnd owned by a Connector must be roles of the Classifier that owned the Connector, or they must be ports of such roles.

Package UML::CompositeStructures::InternalStructures

Class Connector

expression (OCL): true

types

The types of the connectable elements that the ends of a connector are attached to must conform to the types of the association ends of the association that types the connector, if any.

expression (OCL): true

Package UML::CompositeStructures::InternalStructures

Class ConnectorEnd

A connector end is an endpoint of a connector, which attaches the connector to a connectable element. Each connector end is part of one connector.

Generalizations:

MultiplicityElement

Owned Association Ends

+ /**definingEnd** : Property [0..1] {readOnly}

A derived association referencing the corresponding association end on the association which types the connector owing this connector end. This association is derived by selecting the association end at the same place in the ordering of association ends as this connector end.

+ **role** : ConnectableElement [1..1]

The connectable element attached at this connector end. When an instance of the containing classifier is created, a link may (depending on the multiplicities) be created to an instance of the classifier that types this connectable element.

Constraints

multiplicity

The multiplicity of the connector end may not be more general than the multiplicity of the association typing the owning connector.

expression (OCL): true

Package UML::CompositeStructures::InternalStructures

Class Property

A property represents a set of instances that are owned by a containing classifier instance.

Generalizations:

ConnectableElement, StructuralFeature

Specializations:

Port

Found in Diagrams:

Component Wiring, Structured Classifier, The port metaclass

Package UML::CompositeStructures::InternalStructures

Class StructuredClassifier

A structured classifier is an abstract metaclass that represents any classifier whose behavior can be fully or partly described by the collaboration of owned or referenced instances.

Generalizations:

Classifier

Specializations:

Collaboration, EncapsulatedClassifier

Found in Diagrams:

Structured Classifier, The port metaclass

Owned Association Ends

+ **ownedAttribute** : Property [0..*] {ordered, subsets role, subsets attribute, subsets ownedMember}

References the properties owned by the classifier.

+ **ownedConnector** : Connector [0..*] {subsets ownedMember, subsets feature}

References the connectors owned by the classifier.

+ /**part** : Property [0..*] {readOnly}

References the properties specifying instances that the classifier owns by composition. This association is derived, selecting those owned properties where isComposite is true.

+ /**role** : ConnectableElement [0..*] {readOnly, union, subsets member}

References the roles that instances may play in this classifier.

Constraints

multiplicities

The multiplicities on connected elements must be consistent.

expression (OCL): true

Package UML::CompositeStructures::InternalStructures

Association A_attribute_classifier

Member Ends:

attribute, classifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **classifier** : Classifier [0..1]

Package UML::CompositeStructures::InternalStructures

Association A_definingEnd_connectorEnd

Member Ends:

definingEnd, connectorEnd

Owned Association Ends

+ **connectorEnd** : ConnectorEnd [0..*]

Package UML::CompositeStructures::InternalStructures

Association A_end_connector

Member Ends:

end, connector

Owned Association Ends

+ **connector** : Connector [1..1]

Package UML::CompositeStructures::InternalStructures

Association A_end_role

Member Ends:

end, role

Package UML::CompositeStructures::InternalStructures

Association A_feature_classifier

Member Ends:

feature, classifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **classifier** : Classifier [0..*]

Package UML::CompositeStructures::InternalStructures

Association A_ownedAttribute_structuredClassifier

Member Ends:

ownedAttribute, structuredClassifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **structuredClassifier** : StructuredClassifier [0..1]

Package UML::CompositeStructures::InternalStructures

Association A_ownedConnector_structuredClassifier

Member Ends:

ownedConnector, structuredClassifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **structuredClassifier** : StructuredClassifier [0..1] {subsets redefinitionContext}

Package UML::CompositeStructures::InternalStructures

Association A_part_structuredClassifier

Member Ends:

part, structuredClassifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **structuredClassifier** : StructuredClassifier [0..1]

Package UML::CompositeStructures::InternalStructures

Association A_redefinedConnector_connector

Member Ends:

redefinedConnector, connector

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **connector** : Connector [0..*]

Package UML::CompositeStructures::InternalStructures

Association A_role_structuredClassifier

Member Ends:

role, structuredClassifier

Found in Diagrams:

Structured Classifier

Owned Association Ends

+ **structuredClassifier** : StructuredClassifier [0..*]

Package UML::CompositeStructures::InternalStructures

Association A_type_connector

Member Ends:

type, connector

Owned Association Ends

+ **connector** : Connector [0..*]

Package UML::CompositeStructures::InvocationActions

Nesting Package:

CompositeStructures

Merged Packages:

BasicActions, Ports

Class Summary
InvocationAction
Trigger

Association Summary
A_onPort_invocationAction
A_port_trigger

Package UML::CompositeStructures::InvocationActions

Class InvocationAction

In addition to targeting an object, invocation actions can also invoke behavioral features on ports from where the invocation requests are routed onwards on links deriving from attached connectors. Invocation actions may also be sent to a target via a given port, either on the sending object or on another object.

Owned Association Ends

+ **onPort** : Port [0..1]

A optional port of the receiver object on which the behavioral feature is invoked.

Constraints

on_port_receiver

The onPort must be a port on the receiver object.

expression (OCL): true

Package UML::CompositeStructures::InvocationActions

Class Trigger

A trigger specification may be qualified by the port on which the event occurred.

Owned Association Ends

+ **port** : Port [0..*]

A optional port of the receiver object on which the behavioral feature is invoked.

Package UML::CompositeStructures::InvocationActions

Association A_onPort_invocationAction

Member Ends:

onPort, invocationAction

Owned Association Ends

+ **invocationAction** : InvocationAction [0..*]

Package UML::CompositeStructures::InvocationActions

Association A_port_trigger

Member Ends:

port, trigger

Owned Association Ends

+ **trigger** : Trigger [0..*]

Package UML::CompositeStructures::Ports

Nesting Package:

CompositeStructures

Imported Packages:

Interfaces, Kernel

Merged Packages:

Communications, InternalStructures

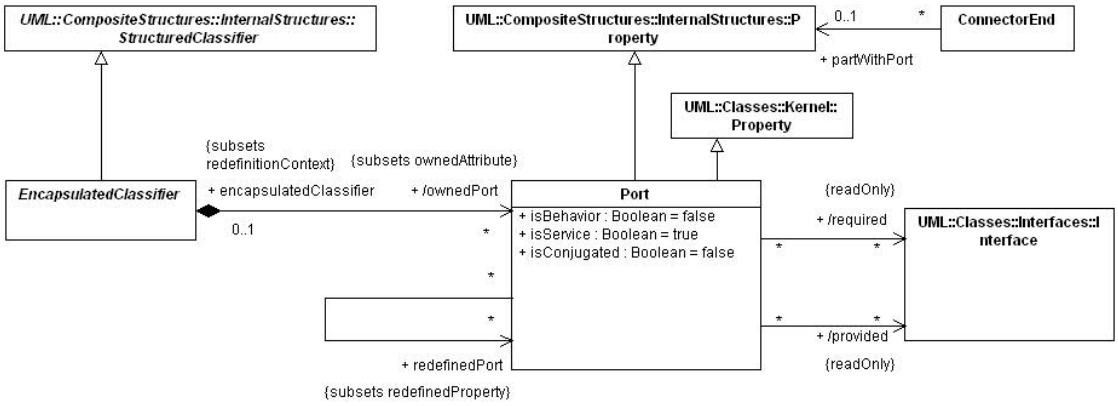
Diagram Summary
The port metaclass

Class Summary
ConnectorEnd
EncapsulatedClassifier
Port

Association Summary
A_ownedPort_encapsulatedClassifier
A_partWithPort_connectorEnd
A_provided_port
A_redefinedPort_port
A_required_port

Package UML::CompositeStructures::Ports

Diagram The port metaclass



Classifiers Local to Package:

ConnectorEnd, EncapsulatedClassifier, Port

Classifiers External to Package:

Interface, Property, Property, StructuredClassifier

Package UML::CompositeStructures::Ports

Class ConnectorEnd

A connector end is an endpoint of a connector, which attaches the connector to a connectable element. Each connector end is part of one connector.

Found in Diagrams:

The port metaclass

Owned Association Ends

+ **partWithPort** : Property [0..1]

Indicates the role of the internal structure of a classifier with the port to which the connector end is attached.

Constraints

part_with_port_empty

If a connector end is attached to a port of the containing classifier, partWithPort will be empty.

expression (OCL): true

role_and_part_with_port

If a connector end references a partWithPort, then the role must be a port that is defined by the type of the partWithPort.

expression (OCL): true

self_part_with_port

The property held in self.partWithPort must not be a Port.

expression (OCL): true

Package UML::CompositeStructures::Ports

Class EncapsulatedClassifier

A classifier has the ability to own ports as specific and type checked interaction points.

Generalizations:

StructuredClassifier

Specializations:

Class

Found in Diagrams:

The port metaclass

Owned Association Ends

+ /**ownedPort** : Port [0..*] {subsets ownedAttribute}

References a set of ports that an encapsulated classifier owns.

Package UML::CompositeStructures::Ports

Class Port

A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts. Ports are connected to properties of the classifier by connectors through which requests can be made to invoke the behavioral features of a classifier. A Port may specify the services a classifier provides (offers) to its environment as well as the services that a classifier expects (requires) of its environment.

Generalizations:

Property, Property

Found in Diagrams:

The port metaclass

Attributes

+ **isBehavior** : Boolean [1..1] = false

Specifies whether requests arriving at this port are sent to the classifier behavior of this classifier. Such ports are referred to as behavior port. Any invocation of a behavioral feature targeted at a behavior port will be handled by the instance of the owning classifier itself, rather than by any instances that this classifier may contain.

+ **isConjugated** : Boolean [1..1] = false

Specifies the way that the provided and required interfaces are derived from the Port's Type. The default value is false.

+ **isService** : Boolean [1..1] = true

If true indicates that this port is used to provide the published functionality of a classifier; if false, this port is used to implement the classifier but is not part of the essential externally-visible functionality of the classifier and can, therefore, be altered or deleted along with the internal implementation of the classifier and other properties that are considered part of its implementation.

Owned Association Ends

+ /**provided** : Interface [0..*] {readOnly}

References the interfaces specifying the set of operations and receptions that the classifier offers to its environment via this port, and which it will handle either directly or by forwarding it to a part of its internal structure. This association is derived according to the value of isConjugated. If isConjugated is false, provided is derived as the union of the sets of interfaces realized by the type of the port and its supertypes, or directly from the type of the port if the port is typed by an interface. If isConjugated is true, it is derived as the union of the sets of interfaces used by the type of the port and its supertypes.

Package UML::CompositeStructures::Ports

Class Port

+ **redefinedPort** : Port [0..*] {subsets redefinedProperty}

A port may be redefined when its containing classifier is specialized. The redefining port may have additional interfaces to those that are associated with the redefined port or it may replace an interface by one of its subtypes.

+ /**required** : Interface [0..*] {readOnly}

References the interfaces specifying the set of operations and receptions that the classifier expects its environment to handle via this port. This association is derived according to the value of isConjugated. If isConjugated is false, required is derived as the union of the sets of interfaces used by the type of the port and its supertypes. If isConjugated is true, it is derived as the union of the sets of interfaces realized by the type of the port and its supertypes, or directly from the type of the port if the port is typed by an interface.

Constraints

default_value

A defaultValue for port cannot be specified when the type of the Port is an Interface

expression (OCL): true

port_aggregation

Port.aggregation must be composite.

expression (OCL): true

port_destroyed

When a port is destroyed, all connectors attached to this port will be destroyed also.

expression (OCL): true

Package UML::CompositeStructures::Ports

Association A_ownedPort_encapsulatedClassifier

Member Ends:

ownedPort, encapsulatedClassifier

Found in Diagrams:

The port metaclass

Owned Association Ends

/ + **encapsulatedClassifier** : EncapsulatedClassifier [0..1] {subsets redefinitionContext}

Package UML::CompositeStructures::Ports

Association A_partWithPort_connectorEnd

Member Ends:

partWithPort, connectorEnd

Found in Diagrams:

The port metaclass

Owned Association Ends

/ + **connectorEnd** : ConnectorEnd [0..*]

Package UML::CompositeStructures::Ports

Association A_provided_port

Member Ends:

provided, port

Found in Diagrams:

The port metaclass

Owned Association Ends

+ **port** : Port [0..*]

Package UML::CompositeStructures::Ports

Association A_redefinedPort_port

Member Ends:

redefinedPort, port

Found in Diagrams:

The port metaclass

Owned Association Ends

+ **port** : Port [0..*]

Package UML::CompositeStructures::Ports

Association A_required_port

Member Ends:

required, port

Found in Diagrams:

The port metaclass

Owned Association Ends

+ **port** : Port [0..*]

Package UML::CompositeStructures::StructuredActivities

Nesting Package:

CompositeStructures

Merged Packages:

InternalStructures, StructuredActivities

Class Summary
Variable

Package UML::CompositeStructures::StructuredActivities

Class Variable

A variable is considered a connectable element.

Generalizations:

ConnectableElement

Package UML::CompositeStructures::StructuredClasses

Nesting Package:

CompositeStructures

Merged Packages:

Ports

Class Summary
Class

Package UML::CompositeStructures::StructuredClasses

Class Class

A class has the capability to have an internal structure and ports.

Generalizations:

EncapsulatedClassifier

Specializations:

Component, Component, Node

Found in Diagrams:

Component Construct, Packaging Components

Package UML::Deployments

Nesting Package:

UML

Imported Packages:

Components

Nested Package Summary
Artifacts
ComponentDeployments
Nodes

Package UML::Deployments::Artifacts

Nesting Package:

Deployments

Imported Packages:

Dependencies

Class Summary
Artifact
Manifestation

Association Summary
A_manifestation_artifact
A_nestedArtifact_artifact
A_ownedAttribute_artifact
A_ownedOperation_artifact
A_utilizedElement_manifestation

Package UML::Deployments::Artifacts

Class Artifact

An artifact is the specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system. Examples of artifacts include model files, source files, scripts, and binary executable files, a table in a database system, a development deliverable, or a word-processing document, a mail message.

Generalizations:

Classifier, NamedElement

Attributes

+ **fileName** : String [0..1]

A concrete name that is used to refer to the Artifact in a physical context. Example: file system name, universal resource locator.

Owned Association Ends

+ **manifestation** : Manifestation [0..*] {subsets clientDependency, subsets ownedElement}

The set of model elements that are manifested in the Artifact. That is, these model elements are utilized in the construction (or generation) of the artifact.

+ **nestedArtifact** : Artifact [0..*] {subsets ownedMember}

The Artifacts that are defined (nested) within the Artifact.
The association is a specialization of the ownedMember association from Namespace to NamedElement.

+ **ownedAttribute** : Property [0..*] {ordered, subsets attribute, subsets ownedMember}

The attributes or association ends defined for the Artifact.
The association is a specialization of the ownedMember association.

+ **ownedOperation** : Operation [0..*] {ordered, subsets feature, subsets ownedMember}

The Operations defined for the Artifact. The association is a specialization of the ownedMember association.

Package UML::Deployments::Artifacts

Class Manifestation

A manifestation is the concrete physical rendering of one or more model elements by an artifact.

Generalizations:

Abstraction

Owned Association Ends

+ **utilizedElement** : PackageableElement [1..1] {subsets supplier}

The model element that is utilized in the manifestation in an Artifact.

Package UML::Deployments::Artifacts

Association A_manifestation_artifact

Member Ends:

manifestation, artifact

Owned Association Ends

+ **artifact** : Artifact [1..1]

Package UML::Deployments::Artifacts

Association A_nestedArtifact_artifact

Member Ends:

nestedArtifact, artifact

Owned Association Ends

+ **artifact** : Artifact [0..1]

Package UML::Deployments::Artifacts

Association A_ownedAttribute_artifact

Member Ends:

ownedAttribute, artifact

Owned Association Ends

+ **artifact** : Artifact [0..1] {subsets namespace, subsets featuringClassifier, subsets classifier}

Package UML::Deployments::Artifacts

Association A_ownedOperation_artifact

Member Ends:

ownedOperation, artifact

Owned Association Ends

+ **artifact** : Artifact [0..1] {subsets redefinitionContext, subsets namespace, subsets featuringClassifier}

Package UML::Deployments::Artifacts

Association A_utilizedElement_manifestation

Member Ends:

utilizedElement, manifestation

Owned Association Ends

+ **manifestation** : Manifestation [0..*]

Package UML::Deployments::ComponentDeployments

Nesting Package:

Deployments

Imported Packages:

Kernel

Merged Packages:

Nodes

Class Summary
Deployment
DeploymentSpecification

Association Summary
A_configuration_deployment

Package UML::Deployments::ComponentDeployments

Class Deployment

A component deployment is the deployment of one or more artifacts or artifact instances to a deployment target, optionally parameterized by a deployment specification. Examples are executables and configuration files.

Generalizations:

Dependency

Owned Association Ends

+ **configuration** : DeploymentSpecification [0..*] {subsets ownedElement}

The specification of properties that parameterize the deployment and execution of one or more Artifacts.

Package UML::Deployments::ComponentDeployments

Class DeploymentSpecification

A deployment specification specifies a set of properties that determine execution parameters of a component artifact that is deployed on a node. A deployment specification can be aimed at a specific type of container. An artifact that reifies or implements deployment specification properties is a deployment descriptor.

Generalizations:

Artifact

Attributes

+ **deploymentLocation** : String [0..1]

The location where an Artifact is deployed onto a Node. This is typically a 'directory' or 'memory address'.

+ **executionLocation** : String [0..1]

The location where a component Artifact executes. This may be a local or remote location.

Owned Association Ends

+ **deployment** : Deployment [0..1]

The deployment with which the DeploymentSpecification is associated.

Constraints

deployed_elements

The deployedElements of a DeploymentTarget that are involved in a Deployment that has an associated Deployment-Specification is a kind of Component (i.e. the configured components).

expression (OCL): self.deployment->forAll (d | d.location.deployedElements->forAll (de | de.oclIsKindOf(Component)))

deployment_target

The DeploymentTarget of a DeploymentSpecification is a kind of ExecutionEnvironment.

expression (OCL): result = self.deployment->forAll (d | d.location..oclIsKindOf (ExecutionEnvironment))

Package UML::Deployments::ComponentDeployments

Association A_configuration_deployment

Member Ends:

configuration, deployment

Package UML::Deployments::Nodes

Nesting Package:

Deployments

Merged Packages:

Artifacts, StructuredClasses

Class Summary
Artifact
CommunicationPath
DeployedArtifact
Deployment
DeploymentTarget
Device
ExecutionEnvironment
InstanceSpecification
Node
Property

Association Summary
A_deployedArtifact_deployment
A_deployedElement_deploymentTarget
A_deployment_location
A_nestedNode_node

Package UML::Deployments::Nodes

Class Artifact

An artifact is the source of a deployment to a node.

Generalizations:

DeployedArtifact

Specializations:

DeploymentSpecification

Package UML::Deployments::Nodes

Class CommunicationPath

A communication path is an association between two deployment targets, through which they are able to exchange signals and messages.

Generalizations:

Association

Constraints

association_ends

The association ends of a CommunicationPath are typed by DeploymentTargets.

expression (OCL): result = self.endType->forAll (t | t.oclIsKindOf(DeploymentTarget))

Package UML::Deployments::Nodes

Class DeployedArtifact

A deployed artifact is an artifact or artifact instance that has been deployed to a deployment target.

Generalizations:

NamedElement

Specializations:

Artifact, InstanceSpecification

Package UML::Deployments::Nodes

Class Deployment

A deployment is the allocation of an artifact or artifact instance to a deployment target.

Generalizations:

Dependency

Owned Association Ends

+ **deployedArtifact** : DeployedArtifact [0..*] {subsets supplier}

The Artifacts that are deployed onto a Node. This association specializes the supplier association.

+ **location** : DeploymentTarget [1..1] {subsets client}

The DeployedTarget which is the target of a Deployment.

Package UML::Deployments::Nodes

Class DeploymentTarget

A deployment target is the location for a deployed artifact.

Generalizations:

NamedElement

Specializations:

InstanceSpecification, Node, Property

Owned Association Ends

+ /**deployedElement** : PackageableElement [0..*] {readOnly}

The set of elements that are manifested in an Artifact that is involved in Deployment to a DeploymentTarget.

+ **deployment** : Deployment [0..*] {subsets ownedElement, subsets clientDependency}

The set of Deployments for a DeploymentTarget.

Operations

+ **deployedElement** () : PackageableElement [0..*] {query}

body (OCL): result = ((self.deployment->collect(deployedArtifact))->collect(manifestation))->collect(utilizedElement)

Package UML::Deployments::Nodes

Class Device

A device is a physical computational resource with processing capability upon which artifacts may be deployed for execution. Devices may be complex (i.e., they may consist of other devices).

Generalizations:

Node

Package UML::Deployments::Nodes

Class ExecutionEnvironment

An execution environment is a node that offers an execution environment for specific types of components that are deployed on it in the form of executable artifacts.

Generalizations:

Node

Package UML::Deployments::Nodes

Class InstanceSpecification

An instance specification has the capability of being a deployment target in a deployment relationship, in the case that it is an instance of a node. It is also has the capability of being a deployed artifact, if it is an instance of an artifact.

Generalizations:

DeployedArtifact, DeploymentTarget

Constraints

deployment_artifact

An InstanceSpecification can be a DeployedArtifact if it is the instance specification of an Artifact.

expression (OCL): true

deployment_target

An InstanceSpecification can be a DeploymentTarget if it is the instance specification of a Node and functions as a part in the internal structure of an encompassing Node.

expression (OCL): true

Package UML::Deployments::Nodes

Class Node

A node is computational resource upon which artifacts may be deployed for execution.
Nodes can be interconnected through communication paths to define network structures.

Generalizations:

Class, DeploymentTarget

Specializations:

Device, ExecutionEnvironment

Owned Association Ends

+ **nestedNode** : Node [0..*] {subsets ownedMember}

The Nodes that are defined (nested) within the Node.

Constraints

internal_structure

The internal structure of a Node (if defined) consists solely of parts of type Node.

expression (OCL): true

Package UML::Deployments::Nodes

Class Property

A property has the capability of being a deployment target in a deployment relationship. This enables modeling the deployment to hierarchical nodes that have properties functioning as internal parts.

Generalizations:

DeploymentTarget

Constraints

deployment_target

A Property can be a DeploymentTarget if it is a kind of Node and functions as a part in the internal structure of an encompassing Node.

expression (OCL): true

Package UML::Deployments::Nodes

Association A_deployedArtifact_deployment

Member Ends:

deployedArtifact, deployment

Owned Association Ends

+ **deployment** : Deployment [0..*]

Package UML::Deployments::Nodes

Association A_deployedElement_deploymentTarget

Member Ends:

deployedElement, deploymentTarget

Owned Association Ends

+ **deploymentTarget** : DeploymentTarget [0..*]

Package UML::Deployments::Nodes

Association A_deployment_location

Member Ends:

deployment, location

Package UML::Deployments::Nodes

Association A_nestedNode_node

Member Ends:

nestedNode, node

Owned Association Ends

+ **node** : Node [0..1]

Package UML::Interactions

Nesting Package:

UML

Imported Packages:

CommonBehaviors, CompositeStructures

Nested Package Summary
BasicInteractions
Fragments

Package UML::Interactions::BasicInteractions

Nesting Package:

Interactions

Imported Packages:

BasicActions

Merged Packages:

BasicBehaviors, InternalStructures

Diagram Summary
Interactions
Lifelines
Messages

Class Summary
ActionExecutionSpecification
BehaviorExecutionSpecification
CreationEvent
DestructionEvent
ExecutionEvent
ExecutionOccurrenceSpecification
ExecutionSpecification
GeneralOrdering
Interaction
InteractionFragment
Lifeline
Message
MessageEnd
MessageOccurrenceSpecification
OccurrenceSpecification
ReceiveOperationEvent
ReceiveSignalEvent
SendOperationEvent
SendSignalEvent
StateInvariant

Enumeration Summary

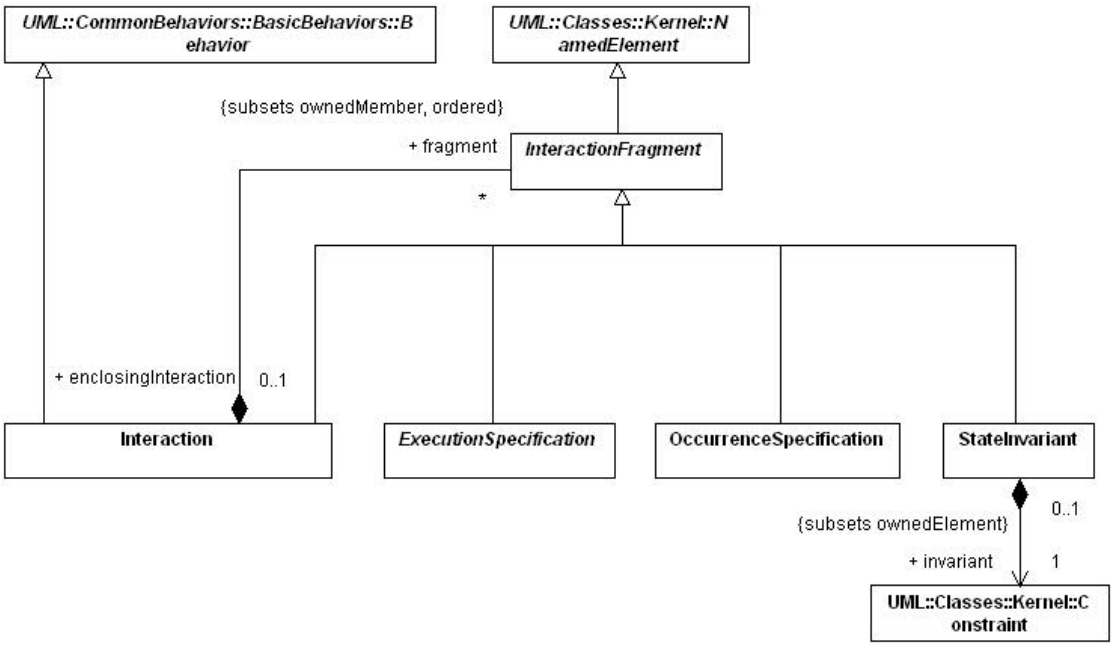
Package UML::Interactions::BasicInteractions

MessageKind
MessageSort

Association Summary
A_action_actionExecutionSpecification
A_action_interaction
A_argument_message
A_before_toAfter
A_behavior_behaviorExecutionSpecification
A_connector_message
A_covered_coveredBy
A_covered_events
A_covered_stateInvariant
A_event_executionOccurrenceSpecification
A_event_occurrenceSpecification
A_execution_executionOccurrenceSpecification
A_finish_executionSpecification
A_fragment_enclosingInteraction
A_generalOrdering_interactionFragment
A_invariant_stateInvariant
A_lifeline_interaction
A_message_interaction
A_message_messageEnd
A_operation_receiveOperationEvent
A_operation_sendOperationEvent
A_receiveEvent_message
A_represents_lifeline
A_selector_lifeline
A_sendEvent_message
A_signal_receiveSignalEvent
A_signal_sendSignalEvent
A_signature_message
A_start_executionSpecification
A_toBefore_after

Package UML::Interactions::BasicInteractions

Diagram Interactions



Classifiers Local to Package:

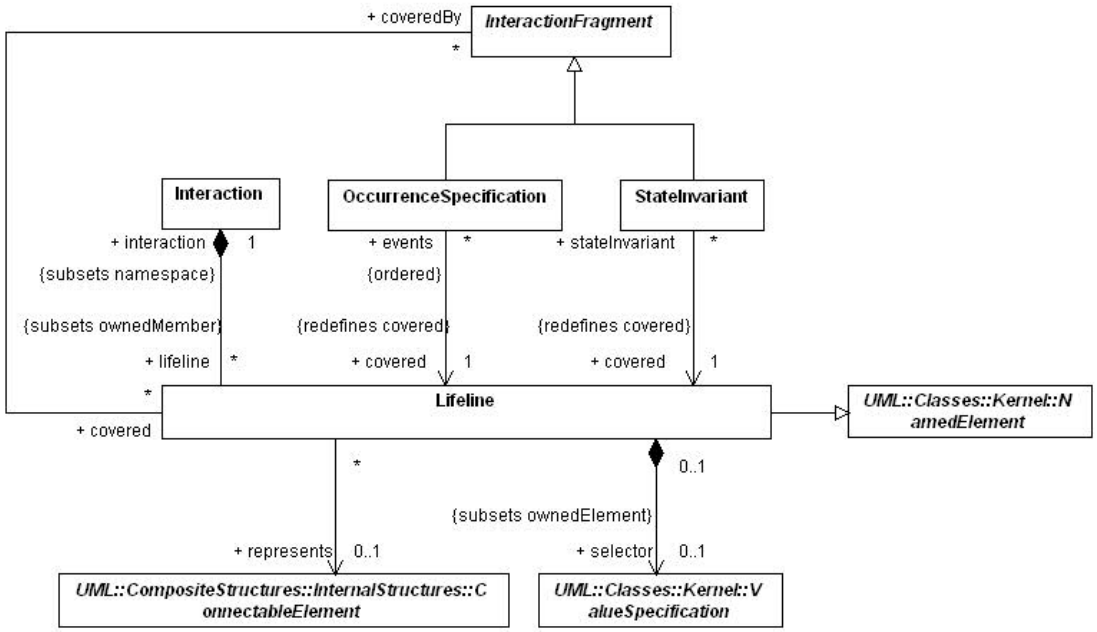
ExecutionSpecification, Interaction, InteractionFragment, OccurrenceSpecification, StateInvariant

Classifiers External to Package:

Behavior, Constraint, NamedElement

Package UML::Interactions::BasicInteractions

Diagram Lifelines



Classifiers Local to Package:

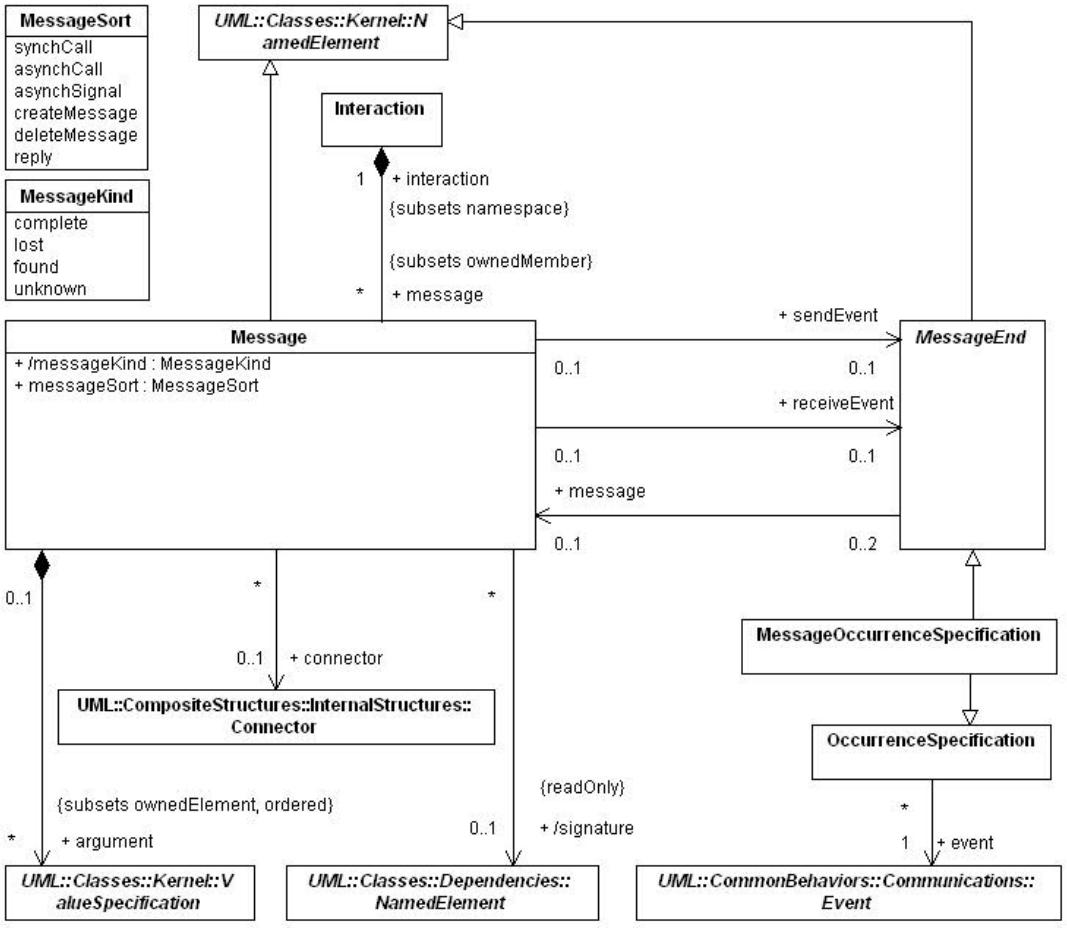
Interaction, InteractionFragment, Lifeline, OccurrenceSpecification, StateInvariant

Classifiers External to Package:

ConnectableElement, NamedElement, ValueSpecification

Package UML::Interactions::BasicInteractions

Diagram Messages



Classifiers Local to Package:

Interaction, Message, MessageEnd, MessageKind, MessageOccurrenceSpecification, MessageSort, OccurrenceSpecification

Classifiers External to Package:

Connector, Event, NamedElement, NamedElement, ValueSpecification

Package UML::Interactions::BasicInteractions

Class ActionExecutionSpecification

An action execution specification is a kind of execution specification representing the execution of an action.

Generalizations:

ExecutionSpecification

Owned Association Ends

+ **action** : Action [1..1]

Action whose execution is occurring.

Constraints

action_referenced

The Action referenced by the ActionExecutionSpecification, if any, must be owned by the Interaction owning the ActionExecutionOccurrence.

expression (OCL): true

Package UML::Interactions::BasicInteractions

Class BehaviorExecutionSpecification

A behavior execution specification is a kind of execution specification representing the execution of a behavior.

Generalizations:

ExecutionSpecification

Owned Association Ends

+ **behavior** : Behavior [0..1]

Behavior whose execution is occurring.

Package UML::Interactions::BasicInteractions

Class CreationEvent

A creation event models the creation of an object.

Generalizations:

Event

Constraints

no_occurrence_above

No othet OccurrenceSpecification may appear above an OccurrenceSpecification which references a CreationEvent on a given Lifeline in an InteractionOperand.

expression (OCL): true

Package UML::Interactions::BasicInteractions

Class DestructionEvent

A destruction event models the destruction of an object.

Generalizations:

Event

Constraints

no_occurrence_specifications_below

No other OccurrenceSpecifications may appear below an OccurrenceSpecification which references a DestructionEvent on a given Lifeline in an InteractionOperand.

expression (OCL): true

Package UML::Interactions::BasicInteractions

Class ExecutionEvent

An ExecutionEvent models the start or finish of an execution specification.

Generalizations:

Event

Package UML::Interactions::BasicInteractions

Class ExecutionOccurrenceSpecification

An execution occurrence specification represents moments in time at which actions or behaviors start or finish.

Generalizations:

OccurrenceSpecification

Owned Association Ends

+ **event** : ExecutionEvent [1..1] {redefines event}

The event referenced is restricted to an execution event.

+ **execution** : ExecutionSpecification [1..1]

References the execution specification describing the execution that is started or finished at this execution event.

Package UML::Interactions::BasicInteractions

Class ExecutionSpecification

An execution specification is a specification of the execution of a unit of behavior or action within the lifeline. The duration of an execution specification is represented by two cccurrence specifications, the start occurrence specification and the finish occurrence specification.

Generalizations:

InteractionFragment

Specializations:

ActionExecutionSpecification, BehaviorExecutionSpecification

Found in Diagrams:

Interactions

Owned Association Ends

+ **finish** : OccurrenceSpecification [1..1]

References the OccurrenceSpecification that designates the finish of the Action or Behavior.

+ **start** : OccurrenceSpecification [1..1]

References the OccurrenceSpecification that designates the start of the Action or Behavior

Constraints

same_lifeline

The startEvent and the finishEvent must be on the same Lifeline

expression (OCL): start.lifeline = finish.lifeline

Package UML::Interactions::BasicInteractions

Class GeneralOrdering

A general ordering represents a binary relation between two occurrence specifications, to describe that one occurrence specification must occur before the other in a valid trace. This mechanism provides the ability to define partial orders of occurrence cpecifications that may otherwise not have a specified order.

Generalizations:

NamedElement

Owned Association Ends

+ **after** : OccurrenceSpecification [1..1]

The OccurrenceSpecification referenced comes after the OccurrenceSpecification referenced by before.

+ **before** : OccurrenceSpecification [1..1]

The OccurrenceSpecification referenced comes before the OccurrenceSpecification referenced by after.

Constraints

irreflexsive_transitive_closure

An occurrence specification must not be ordered relative to itself through a series of general orderings. (In other words, the transitive closure of the general orderings is irreflexive.)

expression (OCL): start.lifeline = finish.lifeline

Package UML::Interactions::BasicInteractions

Class Interaction

An interaction is a unit of behavior that focuses on the observable exchange of information between connectable elements.

Generalizations:

Behavior, InteractionFragment

Found in Diagrams:

Interactions, Lifelines, Messages

Owned Association Ends

+ **action** : Action [0..*] {subsets ownedElement}

Actions owned by the Interaction.

+ **fragment** : InteractionFragment [0..*] {ordered, subsets ownedMember}

The ordered set of fragments in the Interaction.

+ **lifeline** : Lifeline [0..*] {subsets ownedMember}

Specifies the participants in this Interaction.

+ **message** : Message [0..*] {subsets ownedMember}

The Messages contained in this Interaction.

Package UML::Interactions::BasicInteractions

Class InteractionFragment

InteractionFragment is an abstract notion of the most general interaction unit. An interaction fragment is a piece of an interaction. Each interaction fragment is conceptually like an interaction by itself.

Generalizations:

NamedElement

Specializations:

ExecutionSpecification, Interaction, OccurrenceSpecification, StateInvariant

Found in Diagrams:

Interactions, Lifelines

Owned Association Ends

+ **covered** : Lifeline [0..*]

References the Lifelines that the InteractionFragment involves.

+ **enclosingInteraction** : Interaction [0..1]

The Interaction enclosing this InteractionFragment.

+ **generalOrdering** : GeneralOrdering [0..*] {subsets ownedElement}

The general ordering relationships contained in this fragment.

Package UML::Interactions::BasicInteractions

Class Lifeline

A lifeline represents an individual participant in the interaction. While parts and structural features may have multiplicity greater than 1, lifelines represent only one interacting entity.

Generalizations:

NamedElement

Found in Diagrams:

Lifelines

Owned Association Ends

+ **coveredBy** : InteractionFragment [0..*]

References the InteractionFragments in which this Lifeline takes part.

+ **interaction** : Interaction [1..1] {subsets namespace}

References the Interaction enclosing this Lifeline.

+ **represents** : ConnectableElement [0..1]

References the ConnectableElement within the classifier that contains the enclosing interaction.

+ **selector** : ValueSpecification [0..1] {subsets ownedElement}

If the referenced ConnectableElement is multivalued, then this specifies the specific individual part within that set.

Constraints

interaction_uses_share_lifeline

If two (or more) InteractionUses within one Interaction, refer to Interactions with 'common Lifelines,' those Lifelines must also appear in the Interaction with the InteractionUses. By common Lifelines we mean Lifelines with the same selector and represents associations.

expression (OCL): true

same_classifier

The classifier containing the referenced ConnectableElement must be the same classifier, or an ancestor, of the classifier that contains the interaction enclosing this lifeline.

expression (OCL): if (represents->notEmpty()) then (if selector->notEmpty() then represents.isMultivalued() else not represents.isMultivalued())

selector_specified

Package UML::Interactions::BasicInteractions

Class Lifeline

The selector for a Lifeline must only be specified if the referenced Part is multivalued.

expression (OCL): (self.selector->isEmpty() implies not self.represents.isMultivalued()) or (not self.selector->isEmpty() implies self.represents.isMultivalued())

Package UML::Interactions::BasicInteractions

Class Message

A message defines a particular communication between lifelines of an interaction.

Generalizations:

NamedElement

Found in Diagrams:

Messages

Attributes

+ /**messageKind** : MessageKind [1..1] = unknown {readOnly}

The derived kind of the Message (complete, lost, found or unknown)

+ **messageSort** : MessageSort [1..1] = synchCall

The sort of communication reflected by the Message

Owned Association Ends

+ **argument** : ValueSpecification [0..*] {ordered, subsets ownedElement}

The arguments of the Message

+ **connector** : Connector [0..1]

The Connector on which this Message is sent.

+ **interaction** : Interaction [1..1] {subsets namespace}

The enclosing Interaction owning the Message

+ **receiveEvent** : MessageEnd [0..1]

References the Receiving of the Message

+ **sendEvent** : MessageEnd [0..1]

References the Sending of the Message.

+ /**signature** : NamedElement [0..1] {readOnly}

The definition of the type or signature of the Message (depending on its kind). The associated named element is derived from the message end that constitutes the sending or receiving message

Package UML::Interactions::BasicInteractions

Class Message

event. If both a sending event and a receiving message event are present, the signature is obtained from the sending event.

Constraints

arguments

Arguments of a Message must only be:
i) attributes of the sending lifeline
ii) constants
iii) symbolic values (which are wildcard values representing any legal value)
iv) explicit parameters of the enclosing Interaction
v) attributes of the class owning the Interaction

expression (OCL): true

cannot_cross_boundaries

Messages cannot cross bounderies of CombinedFragments or their operands.

expression (OCL): true

occurrence_specifications

If the MessageEnds are both OccurrenceSpecifications then the connector must go between the Parts represented by the Lifelines of the two MessageEnds.

expression (OCL): true

sending_receiving_message_event

If the sending MessageEvent and the receiving MessageEvent of the same Message are on the same Lifeline, the sending MessageEvent must be ordered before the receiving MessageEvent.

expression (OCL): true

signature_is_operation

In the case when the Message signature is an Operation, the arguments of the Message must correspond to the parameters of the Operation. A Parameter corresponds to an Argument if the Argument is of the same Class or a specialization of that of the Parameter.

expression (OCL): true

signature_is_signal

In the case when the Message signature is a Signal, the arguments of the Message must correspond to the attributes of the Signal. A Message Argument corresponds to a Signal Attribute if the Arguement is of the same Class or a specialization of that of the Attribute.

expression (OCL): true

Package UML::Interactions::BasicInteractions

Class Message

signature_refer_to

The signature must either refer an Operation (in which case messageSort is either synchCall or asynchCall) or a Signal (in which case messageSort is asynchSignal). The name of the NamedElement referenced by signature must be the same as that of the Message.

expression (OCL): true

Package UML::Interactions::BasicInteractions

Class MessageEnd

MessageEnd is an abstract specialization of NamedElement that represents what can occur at the end of a message.

Generalizations:

NamedElement

Specializations:

Gate, MessageOccurrenceSpecification

Found in Diagrams:

Messages

Owned Association Ends

+ **message** : Message [0..1]

References a Message.

Package UML::Interactions::BasicInteractions

Class MessageOccurrenceSpecification

A message occurrence specification pecifies the occurrence of message events, such as sending and receiving of signals or invoking or receiving of operation calls. A message occurrence specification is a kind of message end. Messages are generated either by synchronous operation calls or asynchronous signal sends. They are received by the execution of corresponding accept event actions.

Generalizations:

MessageEnd, OccurrenceSpecification

Found in Diagrams:

Messages

Package UML::Interactions::BasicInteractions

Class OccurrenceSpecification

An occurrence specification is the basic semantic unit of interactions. The sequences of occurrences specified by them are the meanings of interactions.

Generalizations:

InteractionFragment

Specializations:

ExecutionOccurrenceSpecification, MessageOccurrenceSpecification

Found in Diagrams:

Interactions, Lifelines, Messages

Owned Association Ends

+ **covered** : Lifeline [1..1] {redefines covered}

References the Lifeline on which the OccurrenceSpecification appears.

+ **event** : Event [1..1]

References a specification of the occurring event.

+ **toAfter** : GeneralOrdering [0..*]

References the GeneralOrderings that specify EventOcurrences that must occur after this OccurrenceSpecification

+ **toBefore** : GeneralOrdering [0..*]

References the GeneralOrderings that specify EventOcurrences that must occur before this OccurrenceSpecification

Package UML::Interactions::BasicInteractions

Class ReceiveOperationEvent

A receive operation event specifies the event of receiving an operation invocation for a particular operation by the target entity.

Generalizations:

MessageEvent

Owned Association Ends

+ **operation** : Operation [1..1]

The operation associated with this event.

Package UML::Interactions::BasicInteractions

Class ReceiveSignalEvent

A receive signal event specifies the event of receiving a signal by the target entity.

Generalizations:

MessageEvent

Owned Association Ends

+ **signal** : Signal [1..1]

The signal associated with this event.

Package UML::Interactions::BasicInteractions

Class SendOperationEvent

A send operation event models the invocation of an operation call.

Generalizations:

MessageEvent

Owned Association Ends

+ **operation** : Operation [1..1]

The operation associated with this event.

Package UML::Interactions::BasicInteractions

Class SendSignalEvent

A send signal event models the sending of a signal.

Generalizations:

MessageEvent

Owned Association Ends

+ **signal** : Signal [1..1]

The signal associated with this event.

Package UML::Interactions::BasicInteractions

Class StateInvariant

A state invariant is a runtime constraint on the participants of the interaction. It may be used to specify a variety of different kinds of constraints, such as values of attributes or variables, internal or external states, and so on. A state invariant is an interaction fragment and it is placed on a lifeline.

Generalizations:

InteractionFragment

Found in Diagrams:

Interactions, Lifelines

Owned Association Ends

+ **covered** : Lifeline [1..1] {redefines covered}

References the Lifeline on which the StateInvariant appears.

+ **invariant** : Constraint [1..1] {subsets ownedElement}

A Constraint that should hold at runtime for this StateInvariant

Package UML::Interactions::BasicInteractions

Enumeration MessageKind

This is an enumerated type that identifies the type of message.

Found in Diagrams:

Messages

Enumeration Literals

complete

sendEvent and receiveEvent are present

found

sendEvent absent and receiveEvent present

lost

sendEvent present and receiveEvent absent

unknown

sendEvent and receiveEvent absent (should not appear)

Package UML::Interactions::BasicInteractions

Enumeration MessageSort

This is an enumerated type that identifies the type of communication action that was used to generate the message.

Found in Diagrams:

Messages

Enumeration Literals

asynchCall

The message was generated by an asynchronous call to an operation; i.e., a CallAction with isSynchronous
= false.

asynchSignal

The message was generated by an asynchronous send action.

createMessage

The message designating the creation of another lifeline object.

deleteMessage

The message designating the termination of another lifeline.

reply

The message is a reply message to an operation call.

synchCall

The message was generated by a synchronous call to an operation.

Package UML::Interactions::BasicInteractions

Association A_action_actionExecutionSpecification

Member Ends:

action, actionExecutionSpecification

Owned Association Ends

+ **actionExecutionSpecification** : ActionExecutionSpecification [0..*]

Package UML::Interactions::BasicInteractions

Association A_action_interaction

Member Ends:

action, interaction

Owned Association Ends

+ **interaction** : Interaction [0..1] {subsets owner}

Package UML::Interactions::BasicInteractions

Association A_argument_message

Member Ends:

argument, message

Found in Diagrams:

Messages

Owned Association Ends

+ **message** : Message [0..1]

Package UML::Interactions::BasicInteractions

Association A_before_toAfter

Member Ends:

before, toAfter

Package UML::Interactions::BasicInteractions

Association A_behavior_behaviorExecutionSpecification

Member Ends:

behavior, behaviorExecutionSpecification

Owned Association Ends

+ **behaviorExecutionSpecification** : BehaviorExecutionSpecification [0..*]

Package UML::Interactions::BasicInteractions

Association A_connector_message

Member Ends:

connector, message

Found in Diagrams:

Messages

Owned Association Ends

+ **message** : Message [0..*]

Package UML::Interactions::BasicInteractions

Association A_covered_coveredBy

This association shows the lifelines that make up an interaction. A lifeline may be part of more than one interaction use.

Member Ends:

covered, coveredBy

Found in Diagrams:

Lifelines

Package UML::Interactions::BasicInteractions

Association A_covered_events

Member Ends:

covered, events

Found in Diagrams:

Lifelines

Owned Association Ends

+ **events** : OccurrenceSpecification [0..*] {ordered}

Package UML::Interactions::BasicInteractions

Association A_covered_stateInvariant

Member Ends:

covered, stateInvariant

Found in Diagrams:

Lifelines

Owned Association Ends

+ **stateInvariant** : StateInvariant [0..*]

Package UML::Interactions::BasicInteractions

Association A_event_executionOccurrenceSpecification

Member Ends:

event, executionOccurrenceSpecification

Owned Association Ends

+ **executionOccurrenceSpecification** : ExecutionOccurrenceSpecification [0..*]

Package UML::Interactions::BasicInteractions

Association A_event_occurrenceSpecification

Member Ends:

event, occurrenceSpecification

Found in Diagrams:

Messages

Owned Association Ends

+ **occurrenceSpecification** : OccurrenceSpecification [0..*]

Package UML::Interactions::BasicInteractions

Association A_execution_executionOccurrenceSpecification

Member Ends:

execution, executionOccurrenceSpecification

Owned Association Ends

+ **executionOccurrenceSpecification** : ExecutionOccurrenceSpecification [1..1]

Package UML::Interactions::BasicInteractions

Association A_finish_executionSpecification

The event shows the time point at which the action completes execution.

Member Ends:

finish, executionSpecification

Owned Association Ends

+ **executionSpecification** : ExecutionSpecification [0..*]

Package UML::Interactions::BasicInteractions

Association A_fragment_enclosingInteraction

Member Ends:

fragment, enclosingInteraction

Found in Diagrams:

Interactions

Package UML::Interactions::BasicInteractions

Association A_generalOrdering_interactionFragment

Member Ends:

generalOrdering, interactionFragment

Owned Association Ends

+ **interactionFragment** : InteractionFragment [0..1]

Package UML::Interactions::BasicInteractions

Association A_invariant_stateInvariant

Member Ends:

invariant, stateInvariant

Found in Diagrams:

Interactions

Owned Association Ends

+ **stateInvariant** : StateInvariant [0..1]

Package UML::Interactions::BasicInteractions

Association A_lifeline_interaction

Member Ends:

lifeline, interaction

Found in Diagrams:

Lifelines

Package UML::Interactions::BasicInteractions

Association A_message_interaction

Member Ends:

message, interaction

Found in Diagrams:

Messages

Package UML::Interactions::BasicInteractions

Association A_message_messageEnd

Member Ends:

message, messageEnd

Found in Diagrams:

Messages

Owned Association Ends

+ **messageEnd** : MessageEnd [0..2]

Package UML::Interactions::BasicInteractions

Association A_operation_receiveOperationEvent

Member Ends:

operation, receiveOperationEvent

Owned Association Ends

+ **receiveOperationEvent** : ReceiveOperationEvent [0..*]

Package UML::Interactions::BasicInteractions

Association A_operation_sendOperationEvent

Member Ends:

operation, sendOperationEvent

Owned Association Ends

+ **sendOperationEvent** : SendOperationEvent [0..*]

Package UML::Interactions::BasicInteractions

Association A_receiveEvent_message

Member Ends:

receiveEvent, message

Found in Diagrams:

Messages

Owned Association Ends

+ **message** : Message [0..1]

Package UML::Interactions::BasicInteractions

Association A_represents_lifeline

If a Part has multiplicity, multiple lifelines might be used to show it.

Member Ends:

represents, lifeline

Found in Diagrams:

Lifelines

Owned Association Ends

+ **lifeline** : Lifeline [0..*]

Package UML::Interactions::BasicInteractions

Association A_selector_lifeline

Member Ends:

selector, lifeline

Found in Diagrams:

Lifelines

Owned Association Ends

+ **lifeline** : Lifeline [0..1]

Package UML::Interactions::BasicInteractions

Association A_sendEvent_message

Member Ends:

sendEvent, message

Found in Diagrams:

Messages

Owned Association Ends

+ **message** : Message [0..1]

Package UML::Interactions::BasicInteractions

Association A_signal_receiveSignalEvent

Member Ends:

signal, receiveSignalEvent

Owned Association Ends

+ **receiveSignalEvent** : ReceiveSignalEvent [0..*]

Package UML::Interactions::BasicInteractions

Association A_signal_sendSignalEvent

Member Ends:

signal, sendSignalEvent

Owned Association Ends

+ **sendSignalEvent** : SendSignalEvent [0..*]

Package UML::Interactions::BasicInteractions

Association A_signature_message

Member Ends:

signature, message

Found in Diagrams:

Messages

Owned Association Ends

+ **message** : Message [0..*]

Package UML::Interactions::BasicInteractions

Association A_start_executionSpecification

The event shows the time point at which the action begins execution.

Member Ends:

start, executionSpecification

Owned Association Ends

+ **executionSpecification** : ExecutionSpecification [0..*]

Package UML::Interactions::BasicInteractions

Association A_toBefore_after

Member Ends:

toBefore, after

Package UML::Interactions::Fragments

Nesting Package:

Interactions

Merged Packages:

BasicInteractions

Class Summary
CombinedFragment
ConsiderIgnoreFragment
Continuation
Gate
Interaction
InteractionConstraint
InteractionFragment
InteractionOperand
InteractionUse
Lifeline
PartDecomposition

Enumeration Summary
InteractionOperatorKind

Association Summary
A_actualGate_interactionUse
A_argument_interactionUse
A_cfragmentGate_combinedFragment
A_decomposedAs_lifeline
A_formalGate_interaction
A_fragment_enclosingOperand
A_guard_interactionOperand
A_maxint_interactionConstraint
A_message_considerIgnoreFragment
A_minint_interactionConstraint
A_operand_combinedFragment
A_refersTo_interactionUse

Package UML::Interactions::Fragments

Class CombinedFragment

A combined fragment defines an expression of interaction fragments. A combined fragment is defined by an interaction operator and corresponding interaction operands. Through the use of combined fragments the user will be able to describe a number of traces in a compact and concise manner.

Generalizations:

InteractionFragment

Specializations:

ConsiderIgnoreFragment

Attributes

+ **interactionOperator** : InteractionOperatorKind [1..1] = seq

Specifies the operation which defines the semantics of this combination of InteractionFragments.

Owned Association Ends

+ **cfragmentGate** : Gate [0..*] {subsets ownedElement}

Specifies the gates that form the interface between this CombinedFragment and its surroundings

+ **operand** : InteractionOperand [1..*] {ordered, subsets ownedElement}

The set of operands of the combined fragment.

Constraints

break

If the interactionOperator is break, the corresponding InteractionOperand must cover all Lifelines within the enclosing InteractionFragment.

expression (OCL): true

consider_and_ignore

The interaction operators 'consider' and 'ignore' can only be used for the CombineIgnoreFragment subtype of CombinedFragment

expression (OCL): ((interactionOperator = #consider) or (interactionOperator = #ignore)) implies oclsisTypeOf(CombineIgnoreFragment)

minint_and_maxint

The InteractionConstraint with minint and maxint only apply when attached to an InteractionOperand where the interactionOperator is loop.

Package UML::Interactions::Fragments

Class CombinedFragment

expression (OCL): true

opt_loop_break_neg

If the interactionOperator is opt, loop, break, assert or neg, there must be exactly one operand.

expression (OCL): true

Package UML::Interactions::Fragments

Class ConsiderIgnoreFragment

A consider ignore fragment is a kind of combined fragment that is used for the consider and ignore cases, which require lists of pertinent messages to be specified.

Generalizations:

CombinedFragment

Owned Association Ends

+ **message** : NamedElement [0..*]

The set of messages that apply to this fragment

Constraints

consider_or_ignore

The interaction operator of a ConsiderIgnoreFragment must be either 'consider' or 'ignore'.

expression (OCL): (interactionOperator = #consider) or (interactionOperator = #ignore)

type

The NamedElements must be of a type of element that identifies a message (e.g., an Operation, Reception, or a Signal).

expression (OCL): message->forAll(m | m.oclIsKindOf(Operation) or m.oclIsKindOf(Reception) or m.oclIsKindOf(Signal))

Package UML::Interactions::Fragments

Class Continuation

A continuation is a syntactic way to define continuations of different branches of an alternative combined fragment. Continuations is intuitively similar to labels representing intermediate points in a flow of control.

Generalizations:

InteractionFragment

Attributes

+ **setting** : Boolean [1..1] = true

True: when the Continuation is at the end of the enclosing InteractionFragment and False when it is in the beginning.

Constraints

first_or_last_interaction_fragment

Continuations always occur as the very first InteractionFragment or the very last InteractionFragment of the enclosing InteractionFragment.

expression (OCL): true

global

Continuations are always global in the enclosing InteractionFragment e.g. it always covers all Lifelines covered by the enclosing InteractionFragment.

expression (OCL): true

same_name

Continuations with the same name may only cover the same set of Lifelines (within one Classifier).

expression (OCL): true

Package UML::Interactions::Fragments

Class Gate

A gate is a connection point for relating a message outside an interaction fragment with a message inside the interaction fragment.

Generalizations:

MessageEnd

Constraints

messages_actual_gate

The message leading to/from an actualGate of an InteractionUse must correspond to the message leading from/to the formalGate with the same name of the Interaction referenced by the InteractionUse.

expression (OCL): true

messages_combined_fragment

The message leading to/from an (expression) Gate within a CombinedFragment must correspond to the message leading from/to the CombinedFragment on its outside.

expression (OCL): true

Package UML::Interactions::Fragments

Class Interaction

Generalizations:

Behavior

Owned Association Ends

+ **formalGate** : Gate [0..*] {subsets ownedMember}

Specifies the gates that form the message interface between this Interaction and any InteractionUses which reference it.

Package UML::Interactions::Fragments

Class InteractionConstraint

An interaction constraint is a Boolean expression that guards an operand in a combined fragment.

Generalizations:

Constraint

Owned Association Ends

+ **maxint** : ValueSpecification [0..1] {subsets ownedElement}

The maximum number of iterations of a loop

+ **minint** : ValueSpecification [0..1] {subsets ownedElement}

The minimum number of iterations of a loop

Constraints

dynamic_variables

The dynamic variables that take part in the constraint must be owned by the ConnectableElement corresponding to the covered Lifeline.

expression (OCL): true

global_data

The constraint may contain references to global data or write-once data.

expression (OCL): true

maxint_greater_equal_minint

If maxint is specified, then minint must be specified and the evaluation of maxint must be >= the evaluation of minint

expression (OCL): true

maxint_positive

If maxint is specified, then the expression must evaluate to a positive integer.

expression (OCL): true

minint_maxint

Minint/maxint can only be present if the InteractionConstraint is associated with the operand of a loop CombinedFragment.

expression (OCL): true

minint_non_negative

Package UML::Interactions::Fragments

Class InteractionConstraint

If minint is specified, then the expression must evaluate to a non-negative integer.

expression (OCL): true

Package UML::Interactions::Fragments

Class InteractionFragment

Generalizations:

NamedElement

Specializations:

CombinedFragment, Continuation, InteractionOperand, InteractionUse

Owned Association Ends

/ + **enclosingOperand** : InteractionOperand [0..1] {subsets namespace}

The operand enclosing this InteractionFragment (they may nest recursively)

Package UML::Interactions::Fragments

Class InteractionOperand

An interaction operand is contained in a combined fragment. An interaction operand represents one operand of the expression given by the enclosing combined fragment.

Generalizations:

InteractionFragment, Namespace

Owned Association Ends

+ **fragment** : InteractionFragment [0..*] {ordered, subsets ownedMember}

The fragments of the operand.

+ **guard** : InteractionConstraint [0..1] {subsets ownedElement}

Constraint of the operand.

Constraints

guard_contain_references

The guard must contain only references to values local to the Lifeline on which it resides, or values global to the whole Interaction.

expression (OCL): true

guard_directly_prior

The guard must be placed directly prior to (above) the OccurrenceSpecification that will become the first OccurrenceSpecification within this InteractionOperand.

expression (OCL): true

Package UML::Interactions::Fragments

Class InteractionUse

An interaction use refers to an interaction. The interaction use is a shorthand for copying the contents of the referenced interaction where the interaction use is. To be accurate the copying must take into account substituting parameters with arguments and connect the formal gates with the actual ones.

Generalizations:

InteractionFragment

Specializations:

PartDecomposition

Owned Association Ends

+ **actualGate** : Gate [0..*] {subsets ownedElement}

The actual gates of the InteractionUse

+ **argument** : Action [0..*] {ordered}

The actual arguments of the Interaction

+ **refersTo** : Interaction [1..1]

Refers to the Interaction that defines its meaning

Constraints

all_lifelines

The InteractionUse must cover all Lifelines of the enclosing Interaction that represent the same properties as lifelines within the referred Interaction.

expression (OCL): true

arguments_are_constants

The arguments must only be constants, parameters of the enclosing Interaction or attributes of the classifier owning the enclosing Interaction.

expression (OCL): true

arguments_correspond_to_parameters

The arguments of the InteractionUse must correspond to parameters of the referred Interaction

expression (OCL): true

gates_match

Package UML::Interactions::Fragments

Class InteractionUse

Actual Gates of the InteractionUse must match Formal Gates of the referred Interaction. Gates match when their names are equal.

expression (OCL): true

Package UML::Interactions::Fragments

Class Lifeline

Owned Association Ends

+ **decomposedAs** : PartDecomposition [0..1]

References the Interaction that represents the decomposition.

Package UML::Interactions::Fragments

Class PartDecomposition

A part decomposition is a description of the internal interactions of one lifeline relative to an interaction.

Generalizations:

InteractionUse

Constraints

assume

Assume that within Interaction X, Lifeline L is of class C and decomposed to D. Within X there is a sequence of constructs along L (such constructs are CombinedFragments, InteractionUse and (plain) OccurrenceSpecifications). Then a corresponding sequence of constructs must appear within D, matched one-to-one in the same order.

i) CombinedFragment covering L are matched with an extra-global CombinedFragment in D
ii) An InteractionUse covering L are matched with a global (i.e. covering all Lifelines) InteractionUse in D.
iii) A plain OccurrenceSpecification on L is considered an actualGate that must be matched by a formalGate of D

expression (OCL): true

commutativity_of_decomposition

Assume that within Interaction X, Lifeline L is of class C and decomposed to D. Assume also that there is within X an
InteractionUse (say) U that covers L. According to the constraint above U will have a counterpart CU within D. Within the Interaction referenced by U, L should also be decomposed, and the decomposition should reference CU. (This rule is called commutativity of decomposition)

expression (OCL): true

parts_of_internal_structures

PartDecompositions apply only to Parts that are Parts of Internal Structures not to Parts of Collaborations.

expression (OCL): true

Package UML::Interactions::Fragments

Enumeration InteractionOperatorKind

InteractionOperatorKind is an enumeration designating the different kinds of operators of combined fragments. The interaction operand defines the type of operator of a combined fragment.

Enumeration Literals

alt

The interactionOperator alt designates that the CombinedFragment represents a choice of behavior. At most one of the operands will be chosen. The chosen operand must have an explicit or implicit guard expression that evaluates to true at this point in the interaction. An implicit true guard is implied if the operand has no guard.

assert

The interactionOperator assert designates that the CombinedFragment represents an assertion. The sequences of the operand of the assertion are the only valid continuations. All other continuations result in an invalid trace.

break

The interactionOperator break designates that the CombinedFragment represents a breaking scenario in the sense that the operand is a scenario that is performed instead of the remainder of the enclosing InteractionFragment. A break operator with a guard is chosen when the guard is true and the rest of the enclosing Interaction Fragment is ignored. When the guard of the break operand is false, the break operand is ignored and the rest of the enclosing InteractionFragment is chosen. The choice between a break operand without a guard and the rest of the enclosing InteractionFragment is done non-deterministically.

consider

The interactionOperator consider designates which messages should be considered within this combined fragment. This is equivalent to defining every other message to be ignored.

critical

The interactionOperator critical designates that the CombinedFragment represents a critical region. A critical region means that the traces of the region cannot be interleaved by other OccurrenceSpecifications (on those Lifelines covered by the region). This means that the region is treated atomically by the enclosing fragment when determining the set of valid traces. Even though enclosing CombinedFragments may imply that some OccurrenceSpecifications may interleave into the region, such as e.g. with par-operator, this is prevented by defining a region.

ignore

The interacionOperator ignore designates that there are some message types that are not shown within this combined fragment. These message types can be considered insignificant and are implicitly ignored if they appear in a corresponding execution. Alternatively, one can understand ignore to mean that the message types that are ignored can appear anywhere in the traces.

Package UML::Interactions::Fragments

Enumeration InteractionOperatorKind

loop

The interactionOperator loop designates that the CombinedFragment represents a loop. The loop operand will be repeated a number of times.

neg

The interactionOperator neg designates that the CombinedFragment represents traces that are defined to be invalid.

opt

The interactionOperator opt designates that the CombinedFragment represents a choice of behavior where either the (sole) operand happens or nothing happens. An option is semantically equivalent to an alternative CombinedFragment where there is one operand with non-empty content and the second operand is empty.

par

The interactionOperator par designates that the CombinedFragment represents a parallel merge between the behaviors of the operands. The OccurrenceSpecifications of the different operands can be interleaved in any way as long as the ordering imposed by each operand as such is preserved.

seq

The interactionOperator seq designates that the CombinedFragment represents a weak sequencing between the behaviors of the operands.

strict

The interactionOperator strict designates that the CombinedFragment represents a strict sequencing between the behaviors of the operands. The semantics of strict sequencing defines a strict ordering of the operands on the first level within the CombinedFragment with interactionOperator strict. Therefore OccurrenceSpecifications within contained CombinedFragment will not directly be compared with other OccurrenceSpecifications of the enclosing CombinedFragment.

Package UML::Interactions::Fragments

Association A_actualGate_interactionUse

Member Ends:

actualGate, interactionUse

Owned Association Ends

+ **interactionUse** : InteractionUse [0..1]

Package UML::Interactions::Fragments

Association A_argument_interactionUse

Member Ends:

argument, interactionUse

Owned Association Ends

+ **interactionUse** : InteractionUse [0..1]

Package UML::Interactions::Fragments

Association A_cfragmentGate_combinedFragment

Member Ends:

cfragmentGate, combinedFragment

Owned Association Ends

+ **combinedFragment** : CombinedFragment [0..1]

Package UML::Interactions::Fragments

Association A_decomposedAs_lifeline

Member Ends:

decomposedAs, lifeline

Owned Association Ends

+ **lifeline** : Lifeline [1..1]

Package UML::Interactions::Fragments

Association A_formalGate_interaction

Member Ends:

formalGate, interaction

Owned Association Ends

+ **interaction** : Interaction [0..1]

Package UML::Interactions::Fragments

Association A_fragment_enclosingOperand

Member Ends:

fragment, enclosingOperand

Package UML::Interactions::Fragments

Association A_guard_interactionOperand

Member Ends:

guard, interactionOperand

Owned Association Ends

+ **interactionOperand** : InteractionOperand [1..1]

Package UML::Interactions::Fragments

Association A_maxint_interactionConstraint

Member Ends:

maxint, interactionConstraint

Owned Association Ends

+ **interactionConstraint** : InteractionConstraint [0..1]

Package UML::Interactions::Fragments

Association A_message_considerIgnoreFragment

Member Ends:

message, considerIgnoreFragment

Owned Association Ends

+ **considerIgnoreFragment** : ConsiderIgnoreFragment [0..*]

Package UML::Interactions::Fragments

Association A_minint_interactionConstraint

Member Ends:

minint, interactionConstraint

Owned Association Ends

+ **interactionConstraint** : InteractionConstraint [0..1]

Package UML::Interactions::Fragments

Association A_operand_combinedFragment

Member Ends:

operand, combinedFragment

Owned Association Ends

+ **combinedFragment** : CombinedFragment [0..1]

Package UML::Interactions::Fragments

Association A_refersTo_interactionUse

Member Ends:

refersTo, interactionUse

Owned Association Ends

+ **interactionUse** : InteractionUse [0..*]

Package UML::StateMachines

Nesting Package:

UML

Imported Packages:

CommonBehaviors, CompositeStructures

Nested Package Summary
BehaviorStateMachines
ProtocolStateMachines

Package UML::StateMachines::BehaviorStateMachines

Nesting Package:

StateMachines

Imported Packages:

BasicBehaviors, StructuredActivities

Merged Packages:

Communications

Class Summary
ConnectionPointReference
FinalState
Pseudostate
Region
State
StateMachine
TimeEvent
Transition
Vertex

Enumeration Summary
PseudostateKind
TransitionKind

Association Summary
A_connectionPoint_state
A_connectionPoint_stateMachine
A_connection_state
A_deferrableTrigger_state
A_doActivity_state
A_effect_transition
A_entry_connectionPointReference
A_entry_state
A_exit_connectionPointReference
A_exit_state
A_extendedRegion_region
A_extendedStateMachine_stateMachine

Package UML::StateMachines::BehaviorStateMachines

A_guard_transition
A_incoming_target
A_outgoing_source
A_redefinedState_state
A_redefinedTransition_transition
A_redefinitionContext_region
A_redefinitionContext_state
A_redefinitionContext_transition
A_region_state
A_region_stateMachine
A_stateInvariant_owningState
A_submachineState_submachine
A_subvertex_container
A_transition_container
A_trigger_transition

Package UML::StateMachines::BehaviorStateMachines

Class ConnectionPointReference

A connection point reference represents a usage (as part of a submachine state) of an entry/exit point defined in the statemachine reference by the submachine state.

Generalizations:

Vertex

Owned Association Ends

+ **entry** : Pseudostate [0..*]

The entryPoint kind pseudo states corresponding to this connection point.

+ **exit** : Pseudostate [0..*]

The exitPoints kind pseudo states corresponding to this connection point.

+ **state** : State [0..1] {subsets namespace}

The State in which the connection point refreshens are defined.

Constraints

entry_pseudostates

The entry Pseudostates must be Pseudostates with kind entryPoint.

expression (OCL): entry->notEmpty() implies entry->forAll(e | e.kind = #entryPoint)

exit_pseudostates

The exit Pseudostates must be Pseudostates with kind exitPoint.

expression (OCL): exit->notEmpty() implies exit->forAll(e | e.kind = #exitPoint)

Package UML::StateMachines::BehaviorStateMachines

Class FinalState

A special kind of state signifying that the enclosing region is completed. If the enclosing region is directly contained in a state machine and all other regions in the state machine also are completed, then it means that the entire state machine is completed.

Generalizations:

State

Constraints

cannot_reference_submachine

A final state cannot reference a submachine.

expression (OCL): self.submachine->isEmpty()

no_entry_behavior

A final state has no entry behavior.

expression (OCL): self.entry->isEmpty()

no_exit_behavior

A final state has no exit behavior.

expression (OCL): self.exit->isEmpty()

no_outgoing_transitions

A final state cannot have any outgoing transitions.

expression (OCL): self.outgoing->size() = 0

no_regions

A final state cannot have regions.

expression (OCL): self.region->size() = 0

no_state_behavior

A final state has no state (doActivity) behavior.

expression (OCL): self.doActivity->isEmpty()

Package UML::StateMachines::BehaviorStateMachines

Class Pseudostate

A pseudostate is an abstraction that encompasses different types of transient vertices in the state machine graph.

Generalizations:

Vertex

Attributes

+ **kind** : PseudostateKind [1..1] = initial

Determines the precise type of the Pseudostate and can be one of: entryPoint, exitPoint, initial, deepHistory, shallowHistory, join, fork, junction, terminate or choice.

Owned Association Ends

+ **state** : State [0..1] {subsets owner}

The State that owns this pseudostate and in which it appears.

+ **stateMachine** : StateMachine [0..1] {subsets namespace}

The StateMachine in which this Pseudostate is defined. This only applies to Pseudostates of the kind entryPoint or exitPoint.

Constraints

choice_vertex

In a complete statemachine, a choice vertex must have at least one incoming and one outgoing transition.

expression (OCL): (self.kind = #choice) implies ((self.incoming->size >= 1) and (self.outgoing->size >= 1))

fork_vertex

In a complete statemachine, a fork vertex must have at least two outgoing transitions and exactly one incoming transition.

expression (OCL): (self.kind = #fork) implies ((self.incoming->size = 1) and (self.outgoing->size >= 2))

history_vertices

History vertices can have at most one outgoing transition.

expression (OCL): ((self.kind = #deepHistory) or (self.kind = #shallowHistory)) implies (self.

Package UML::StateMachines::BehaviorStateMachines

Class Pseudostate

outgoing->size <= 1)

initial_vertex

An initial vertex can have at most one outgoing transition.

expression (OCL): (self.kind = #initial) implies (self.outgoing->size <= 1)

join_vertex

In a complete statemachine, a join vertex must have at least two incoming transitions and exactly one outgoing transition.

expression (OCL): (self.kind = #join) implies ((self.outgoing->size = 1) and (self.incoming->size >= 2))

junction_vertex

In a complete statemachine, a junction vertex must have at least one incoming and one outgoing transition.

expression (OCL): (self.kind = #junction) implies ((self.incoming->size >= 1) and (self.outgoing->size >= 1))

outgoing_from_initial

The outgoing transition from and initial vertex may have a behavior, but not a trigger or a guard.

expression (OCL): (self.kind = #initial) implies (self.outgoing.guard->isEmpty() and self.outgoing.trigger->isEmpty())

transitions_incoming

All transitions incoming a join vertex must originate in different regions of an orthogonal state.

expression (OCL): (self.kind = #join) implies self.incoming->forAll (t1, t2 | t1<>t2 implies (self.stateMachine.LCA(t1.source, t2.source).container.isOrthogonal))

transitions_outgoing

All transitions outgoing a fork vertex must target states in different regions of an orthogonal state.

expression (OCL): (self.kind = #fork) implies self.outgoing->forAll (t1, t2 | t1<>t2 implies (self.stateMachine.LCA(t1.target, t2.target).container.isOrthogonal))

Package UML::StateMachines::BehaviorStateMachines

Class Region

A region is an orthogonal part of either a composite state or a state machine. It contains states and transitions.

Generalizations:

Namespace, RedefinableElement

Owned Association Ends

+ **extendedRegion** : Region [0..1] {subsets redefinedElement}

The region of which this region is an extension.

+ /**redefinitionContext** : Classifier [1..1] {readOnly, redefines redefinitionContext}

References the classifier in which context this element may be redefined.

+ **state** : State [0..1] {subsets namespace}

The State that owns the Region. If a Region is owned by a State, then it cannot also be owned by a StateMachine.

+ **stateMachine** : StateMachine [0..1] {subsets namespace}

The StateMachine that owns the Region. If a Region is owned by a StateMachine, then it cannot also be owned by a State.

+ **subvertex** : Vertex [0..*] {subsets ownedMember}

The set of vertices that are owned by this region.

+ **transition** : Transition [0..*] {subsets ownedMember}

The set of transitions owned by the region.

Operations

+ **containingStateMachine** () : StateMachine [1..1] {query}

The operation containingStateMachine() returns the sate machine in which this Region is defined

body (OCL): result = if stateMachine->isEmpty() then state.containingStateMachine() else stateMachine endif

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies that a redefining region is consistent with a redefined

Package UML::StateMachines::BehaviorStateMachines

Class Region

region provided that the redefining region is an extension of the redefined region, i.e. it adds vertices and transitions and it redefines states and transitions of the redefined region.

body (OCL): result = true

+ **isRedefinitionContextValid** (redefined : Region) : Boolean [1..1] {query}

The query isRedefinitionContextValid() specifies whether the redefinition contexts of a region are properly related to the redefinition contexts of the specified region to allow this element to redefine the other. The containing statemachine/state of a redefining region must redefine the containing statemachine/state of the redefined region.

body (OCL): result = true

+ **redefinitionContext** () : Classifier [1..1] {query}

The redefinition context of a region is the nearest containing statemachine

body (OCL): result = let sm = containingStateMachine() in if sm.context->isEmpty() or sm.general ->notEmpty() then sm else sm.context endif

Constraints

deep_history_vertex

A region can have at most one deep history vertex

expression (OCL): self.subvertex->select (v | v.oclIsKindOf(Pseudostate))-> select(p : Pseudostate | p.kind = #deepHistory)->size() <= 1

initial_vertex

A region can have at most one initial vertex

expression (OCL): self.subvertex->select (v | v.oclIsKindOf(Pseudostate))-> select(p : Pseudostate | p.kind = #initial)->size() <= 1

owned

If a Region is owned by a StateMachine, then it cannot also be owned by a State and vice versa.

expression (OCL): (stateMachine->notEmpty() implies state->isEmpty()) and (state->notEmpty() implies stateMachine->isEmpty())

shallow_history_vertex

A region can have at most one shallow history vertex

expression (OCL): self.subvertex->select(v | v.oclIsKindOf(Pseudostate))-> select(p : Pseudostate | p.kind = #shallowHistory)->size() <= 1

Package UML::StateMachines::BehaviorStateMachines

Class State

A state models a situation during which some (usually implicit) invariant condition holds.

Generalizations:

Namespace, RedefinableElement, Vertex

Specializations:

FinalState

Attributes

+ **/isComposite** : Boolean [1..1] = false {readOnly}

A state with isComposite=true is said to be a composite state. A composite state is a state that contains at least one region.

+ **/isOrthogonal** : Boolean [1..1] = false {readOnly}

A state with isOrthogonal=true is said to be an orthogonal composite state. An orthogonal composite state contains two or more regions.

+ **/isSimple** : Boolean [1..1] = true {readOnly}

A state with isSimple=true is said to be a simple state. A simple state does not have any regions and it does not refer to any submachine state machine.

+ **/isSubmachineState** : Boolean [1..1] = false {readOnly}

A state with isSubmachineState=true is said to be a submachine state. Such a state refers to a state machine (submachine).

Owned Association Ends

+ **connection** : ConnectionPointReference [0..*] {subsets ownedMember}

The entry and exit connection points used in conjunction with this (submachine) state, i.e. as targets and sources, respectively, in the region with the submachine state. A connection point reference references the corresponding definition of a connection point pseudostate in the statemachine referenced by the submachinestate.

+ **connectionPoint** : Pseudostate [0..*] {subsets ownedElement}

The entry and exit pseudostates of a composite state. These can only be entry or exit Pseudostates, and they must have different names. They can only be defined for composite states.

Package UML::StateMachines::BehaviorStateMachines

Class State

+ **deferrableTrigger** : Trigger [0..*]

A list of triggers that are candidates to be retained by the state machine if they trigger no transitions out of the state (not consumed). A deferred trigger is retained until the state machine reaches a state configuration where it is no longer deferred.

+ **doActivity** : Behavior [0..1] {subsets ownedElement}

An optional behavior that is executed while being in the state. The execution starts when this state is entered, and stops either by itself, or when the state is exited, whichever comes first.

+ **entry** : Behavior [0..1] {subsets ownedElement}

An optional behavior that is executed whenever this state is entered regardless of the transition taken to reach the state. If defined, entry actions are always executed to completion prior to any internal behavior or transitions performed within the state.

+ **exit** : Behavior [0..1] {subsets ownedElement}

An optional behavior that is executed whenever this state is exited regardless of which transition was taken out of the state. If defined, exit actions are always executed to completion only after all internal activities and transition actions have completed execution.

+ **redefinedState** : State [0..1] {subsets redefinedElement}

The state of which this state is a redefinition.

+ /**redefinitionContext** : Classifier [1..1] {readOnly, redefines redefinitionContext}

References the classifier in which context this element may be redefined.

+ **region** : Region [0..*] {subsets ownedMember}

The regions owned directly by the state.

+ **stateInvariant** : Constraint [0..1] {subsets ownedElement}

Specifies conditions that are always true when this state is the current state. In protocol state machines, state invariants are additional conditions to the preconditions of the outgoing transitions, and to the postcondition of the incoming transitions.

+ **submachine** : StateMachine [0..1]

Package UML::StateMachines::BehaviorStateMachines

Class State

The state machine that is to be inserted in place of the (submachine) state.

Operations

+ **containingStateMachine** () : StateMachine [1..1] {query}

The query containingStateMachine() returns the state machine that contains the state either directly or transitively.

body (OCL): result = container.containingStateMachine()

+ **isComposite** () : Boolean [1..1] {query}

A composite state is a state with at least one region.

body (OCL): result = region.notEmpty()

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies that a redefining state is consistent with a redefined state provided that the redefining state is an extension of the redefined state: A simple state can be redefined (extended) to become a composite state (by adding a region) and a composite state can be redefined (extended) by adding regions and by adding vertices, states, and transitions to inherited regions. All states may add or replace entry, exit, and 'doActivity' actions.

body (OCL): result = true

+ **isOrthogonal** () : Boolean [1..1] {query}

An orthogonal state is a composite state with at least 2 regions

body (OCL): result = (region->size () > 1)

+ **isRedefinitionContextValid** (redefined : State) : Boolean [1..1] {query}

The query isRedefinitionContextValid() specifies whether the redefinition contexts of a state are properly related to the redefinition contexts of the specified state to allow this element to redefine the other. The containing region of a redefining state must redefine the containing region of the redefined state.

body (OCL): result = true

+ **isSimple** () : Boolean [1..1] {query}

A simple state is a state without any regions.

body (OCL): result = region.isEmpty()

+ **isSubmachineState** () : Boolean [1..1] {query}

Only submachine states can have a reference statemachine.

Package UML::StateMachines::BehaviorStateMachines

Class State

body (OCL): result = submachine.notEmpty()

+ **redefinitionContext** () : Classifier [1..1] {query}

The redefinition context of a state is the nearest containing statemachine.

body (OCL): result = let sm = containingStateMachine() in if sm.context->isEmpty() or sm.general ->notEmpty() then sm else sm.context endif

Constraints

composite_states

Only composite states can have entry or exit pseudostates defined.

expression (OCL): connectionPoint->notEmpty() implies isComoposite

destinations_or_sources_of_transitions

The connection point references used as destinations/sources of transitions associated with a submachine state must be defined as entry/exit points in the submachine state machine.

expression (OCL): self.isSubmachineState implies (self.connection->forAll (cp | cp.entry->forAll (p | p.statemachine = self.submachine) and cp.exit->forAll (p | p.statemachine = self.submachine)))

entry_or_exit

Only entry or exit pseudostates can serve as connection points.

expression (OCL): connectionPoint->forAll(cp|cp.kind = #entry or cp.kind = #exit)

submachine_or_regions

A state is not allowed to have both a submachine and regions.

expression (OCL): isComposite implies not isSubmachineState

submachine_states

Only submachine states can have connection point references.

expression (OCL): isSubmachineState implies connection->notEmpty ( )

Package UML::StateMachines::BehaviorStateMachines

Class StateMachine

State machines can be used to express the behavior of part of a system. Behavior is modeled as a traversal of a graph of state nodes interconnected by one or more joined transition arcs that are triggered by the dispatching of series of (event) occurrences. During this traversal, the state machine executes a series of activities associated with various elements of the state machine.

Generalizations:

Behavior

Specializations:

ProtocolStateMachine

Owned Association Ends

+ **connectionPoint** : Pseudostate [0..*] {subsets ownedMember}

The connection points defined for this state machine. They represent the interface of the state machine when used as part of submachine state.

+ **extendedStateMachine** : StateMachine [0..*] {subsets redefinedElement}

The state machines of which this is an extension.

+ **region** : Region [1..*] {subsets ownedMember}

The regions owned directly by the state machine.

+ **submachineState** : State [0..*]

References the submachine(s) in case of a submachine state. Multiple machines are referenced in case of a concurrent state.

Operations

+ **LCA** (s1 : State, s2 : State) : Namespace [1..1] {query}

The operation LCA(s1,s2) returns an orthogonal state or region which is the least common ancestor of states s1 and s2, based on the statemachine containment hierarchy.

body (OCL): true

+ **ancestor** (s1 : State, s2 : State) : Boolean [1..1] {query}

The query ancestor(s1, s2) checks whether s1 is an ancestor state of state s2.

body (OCL): result = if (s2 = s1) then true else if (s2.container->isEmpty() or not s2.container.owner.oclIsKindOf(State)) then false else ancestor(s1, s2.container.owner.oclAsType(State)) endif

Package UML::StateMachines::BehaviorStateMachines

Class StateMachine

endif

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies that a redefining state machine is consistent with a redefined state machine provided that the redefining state machine is an extension of the redefined state machine: Regions are inherited and regions can be added, inherited regions can be redefined. In case of multiple redefining state machines, extension implies that the redefining state machine gets orthogonal regions for each of the redefined state machines.

body (OCL): result = true

+ **isRedefinitionContextValid** (redefined : StateMachine) : Boolean [1..1] {query}

The query isRedefinitionContextValid() specifies whether the redefinition contexts of a statemachine are properly related to the redefinition contexts of the specified statemachine to allow this element to redefine the other. The containing classifier of a redefining statemachine must redefine the containing classifier of the redefined statemachine.

body (OCL): result = true

Constraints

classifier_context

The classifier context of a state machine cannot be an interface.

expression (OCL): context->notEmpty() implies not context.oclIsKindOf(Interface)

connection_points

The connection points of a state machine are pseudostates of kind entry point or exit point.

expression (OCL): conectionPoint->forAll (c | c.kind = #entryPoint or c.kind = #exitPoint)

context_classifier

The context classifier of the method state machine of a behavioral feature must be the classifier that owns the behavioral feature.

expression (OCL): specification->notEmpty() implies (context->notEmpty() and specification->featuringClassifier->exists (c | c = context))

method

A state machine as the method for a behavioral feature cannot have entry/exit connection points.

expression (OCL): specification->notEmpty() implies connectionPoint->isEmpty()

Package UML::StateMachines::BehaviorStateMachines

Class TimeEvent

A time event can be defined relative to entering the current state of the executing state machine.

Constraints

starting_time

The starting time for a relative time event may only be omitted for a time event that is the trigger of a state machine.

expression (OCL): true

Package UML::StateMachines::BehaviorStateMachines

Class Transition

A transition is a directed relationship between a source vertex and a target vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to an occurrence of an event of a particular type.

Generalizations:

Namespace, RedefinableElement

Specializations:

ProtocolTransition

Attributes

+ **kind** : TransitionKind [1..1] = external

Indicates the precise type of the transition.

Owned Association Ends

+ **container** : Region [1..1] {subsets namespace}

Designates the region that owns this transition.

+ **effect** : Behavior [0..1] {subsets ownedElement}

Specifies an optional behavior to be performed when the transition fires.

+ **guard** : Constraint [0..1] {subsets ownedRule}

A guard is a constraint that provides a fine-grained control over the firing of the transition. The guard is evaluated when an event occurrence is dispatched by the state machine. If the guard is true at that time, the transition may be enabled, otherwise, it is disabled. Guards should be pure expressions without side effects. Guard expressions with side effects are ill formed.

+ **redefinedTransition** : Transition [0..1] {subsets redefinedElement}

The transition that is redefined by this transition.

+ /**redefinitionContext** : Classifier [1..1] {readOnly, redefines redefinitionContext}

References the classifier in which context this element may be redefined.

+ **source** : Vertex [1..1]

Designates the originating vertex (state or pseudostate) of the transition.

Package UML::StateMachines::BehaviorStateMachines

Class Transition

+ **target** : Vertex [1..1]

Designates the target vertex that is reached when the transition is taken.

+ **trigger** : Trigger [0..*]

Specifies the triggers that may fire the transition.

Operations

+ **containingStateMachine** () : StateMachine [1..1] {query}

The query containingStateMachine() returns the state machine that contains the transition either directly or transitively.

body (OCL): result = container.containingStateMachine()

+ **isConsistentWith** (redefinee : RedefinableElement) : Boolean [1..1] {query}

The query isConsistentWith() specifies that a redefining transition is consistent with a redefined transition provided that the redefining transition has the following relation to the redefined transition: A redefining transition redefines all properties of the corresponding redefined transition, except the source state and the trigger.

precondition (): redefinee.isRedefinitionContextValid(self)

body (OCL): result = (redefinee.oclIsKindOf(Transition) and let trans: Transition = redefinee.oclAsType(Transition) in (source() = trans.source() and trigger() = tran.trigger())

+ **redefinitionContext** () : Classifier [1..1] {query}

The redefinition context of a transition is the nearest containing statemachine.

body (OCL): result = let sm = containingStateMachine() in if sm.context->isEmpty() or sm.general->notEmpty() then sm else sm.context endif

Constraints

fork_segment_guards

A fork segment must not have guards or triggers.

expression (OCL): (source.oclIsKindOf(Pseudostate) and source.kind = #fork) implies (guard->isEmpty() and trigger->isEmpty())

fork_segment_state

A fork segment must always target a state.

Package UML::StateMachines::BehaviorStateMachines

Class Transition

expression (OCL): (source.oclIsKindOf(Pseudostate) and source.kind = #fork) implies (target.oclIsKindOf(State))

initial_transition

An initial transition at the topmost level (region of a statemachine) either has no trigger or it has a trigger with the stereotype <<create>>.

expression (OCL): self.source.oclIsKindOf(Pseudostate) implies (self.source.oclAsType(Pseudostate).kind = #initial) implies (self.source.container = self.stateMachine.top) implies ((self.trigger->isEmpty) or (self.trigger.stereotype.name = 'create'))

join_segment_guards

A join segment must not have guards or triggers.

expression (OCL): (target.oclIsKindOf(Pseudostate) and target.kind = #join) implies (guard->isEmpty() and trigger->isEmpty())

join_segment_state

A join segment must always originate from a state.

expression (OCL): (target.oclIsKindOf(Pseudostate) and target.kind = #join) implies (source.oclIsKindOf(State))

outgoing_pseudostates

Transitions outgoing pseudostates may not have a trigger.

expression (OCL): source.oclIsKindOf(Pseudostate) and (source.kind <> #initial)) implies trigger->isEmpty()

signatures_compatible

In case of more than one trigger, the signatures of these must be compatible in case the parameters of the signal are assigned to local variables/attributes.

expression (OCL): true

state_is_external

A transition with kind external can source any vertex except entry points.

expression (OCL): (kind = TransitionKind::external) implies not (source.oclIsKindOf(Pseudostate) and source.oclAsType(Pseudostate).kind = PseudostateKind::entryPoint)

state_is_internal

A transition with kind internal must have a state as its source, and its source and target must be equal.

expression (OCL): (kind = TransitionKind::internal) implies (source.oclIsKindOf (State) and

Package UML::StateMachines::BehaviorStateMachines

Class Transition

source = target)

state_is_local

A transition with kind local must have a composite state or an entry point as its source.

expression (OCL): (kind = TransitionKind::local) implies ((source.oclIsKindOf (State) and source.oclAsType(State).isComposite) or (source.oclIsKindOf (Pseudostate) and source.oclAsType (Pseudostate).kind = PseudostateKind::entryPoint))

Package UML::StateMachines::BehaviorStateMachines

Class Vertex

A vertex is an abstraction of a node in a state machine graph. In general, it can be the source or destination of any number of transitions.

Generalizations:

NamedElement

Specializations:

ConnectionPointReference, Pseudostate, State

Owned Association Ends

+ **container** : Region [0..1] {subsets namespace}

The region that contains this vertex.

+ /**incoming** : Transition [0..*]

Specifies the transitions entering this vertex.

+ /**outgoing** : Transition [0..*]

Specifies the transitions departing from this vertex.

Operations

+ **containingStateMachine** () : StateMachine [1..1] {query}

The operation containingStateMachine() returns the state machine in which this Vertex is defined

body (OCL): result = if not container->isEmpty() then -- the container is a region container. containingStateMachine() else if (oclIsKindOf(Pseudostate)) then -- entry or exit point? if (kind = # entryPoint) or (kind = #exitPoint) then stateMachine else if (oclIsKindOf (ConnectionPointReference)) then state.containingStateMachine() -- no other valid cases possible endif

+ **incoming** () : Transition [0..*]

body (OCL): result = Transition.allInstances()->select(t | t.target=self)

+ **outgoing** () : Transition [0..*]

body (OCL): result = Transition.allInstances()->select(t | t.source=self)

Package UML::StateMachines::BehaviorStateMachines

Enumeration PseudostateKind

PseudostateKind is an enumeration type.

Enumeration Literals

choice

Choice vertices which, when reached, result in the dynamic evaluation of the guards of the triggers of its outgoing transitions. This realizes a dynamic conditional branch. It allows splitting of transitions into multiple outgoing paths such that the decision on which path to take may be a function of the results of prior actions performed in the same run-tocompletion step. If more than one of the guards evaluates to true, an arbitrary one is selected. If none of the guards evaluates to true, then the model is considered ill-formed. (To avoid this, it is recommended to define one outgoing transition with the predefined else guard for every choice vertex.) Choice vertices should be distinguished from static branch points that are based on junction points (described above).

deepHistory

DeepHistory represents the most recent active configuration of the composite state that directly contains this pseudostate; e.g. the state configuration that was active when the composite state was last exited. A composite state can have at most one deep history vertex. At most one transition may originate from the history connector to the default deep history state. This transition is taken in case the composite state had never been active before. Entry actions of states entered on the path to the state represented by a deep history are performed.

entryPoint

An entry point pseudostate is an entry point of a state machine or composite state. In each region of the state machine or composite state it has a single transition to a vertex within the same region.

exitPoint

An exit point pseudostate is an exit point of a state machine or composite state. Entering an exit point within any region of the composite state or state machine referenced by a submachine state implies the exit of this composite state or submachine state and the triggering of the transition that has this exit point as source in the state machine enclosing the submachine or composite state.

fork

Fork vertices serve to split an incoming transition into two or more transitions terminating on orthogonal target vertices
(i.e. vertices in different regions of a composite state). The segments outgoing from a fork vertex must not have guards or triggers.

initial

An initial pseudostate represents a default vertex that is the source for a single transition to the default state of a composite state. There can be at most one initial vertex in a region. The outgoing transition from the initial vertex may have a behavior, but not a trigger or guard.

Package UML::StateMachines::BehaviorStateMachines

Enumeration PseudostateKind

join

Join vertices serve to merge several transitions emanating from source vertices in different orthogonal regions. The transitions entering a join vertex cannot have guards or triggers.

junction

Junction vertices are semantic-free vertices that are used to chain together multiple transitions. They are used to construct compound transition paths between states. For example, a junction can be used to converge multiple incoming transitions into a single outgoing transition representing a shared transition path (this is known as an merge). Conversely, they can be used to split an incoming transition into multiple outgoing transition segments with different guard conditions. This realizes a static conditional branch. (In the latter case, outgoing transitions whose guard conditions evaluate to false are disabled. A predefined guard denoted 'else' may be defined for at most one outgoing transition. This transition is enabled if all the guards labeling the other transitions are false.) Static conditional branches are distinct from dynamic conditional branches that are realized by choice vertices (described below).

shallowHistory

ShallowHistory represents the most recent active substate of its containing state (but not the substates of that substate). A composite state can have at most one shallow history vertex. A transition coming into the shallow history vertex is equivalent to a transition coming into the most recent active substate of a state. At most one transition may originate from the history connector to the default shallow history state. This transition is taken in case the composite state had never been active before. Entry actions of states entered on the path to the state represented by a shallow history are performed.

terminate

Entering a terminate pseudostate implies that the execution of this state machine by means of its context object is terminated. The state machine does not exit any states nor does it perform any exit actions other than those associated with the transition leading to the terminate pseudostate. Entering a terminate pseudostate is equivalent to invoking a DestroyObjectAction.

Package UML::StateMachines::BehaviorStateMachines

Enumeration TransitionKind

TransitionKind is an enumeration type.

Enumeration Literals

external

Implies that the transition, if triggered, will exit the composite (source) state.

internal

Implies that the transition, if triggered, occurs without exiting or entering the source state. Thus, it does not cause a state change. This means that the entry or exit condition of the source state will not be invoked. An internal transition can be taken even if the state machine is in one or more regions nested within this state.

local

Implies that the transition, if triggered, will not exit the composite (source) state, but it will apply to any state within the composite state, and these will be exited and entered.

Package UML::StateMachines::BehaviorStateMachines

Association A_connectionPoint_state

Member Ends:

connectionPoint, state

Package UML::StateMachines::BehaviorStateMachines

Association A_connectionPoint_stateMachine

Member Ends:

connectionPoint, stateMachine

Package UML::StateMachines::BehaviorStateMachines

Association A_connection_state

Member Ends:

connection, state

Package UML::StateMachines::BehaviorStateMachines

Association A_deferrableTrigger_state

Member Ends:

deferrableTrigger, state

Owned Association Ends

+ **state** : State [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_doActivity_state

Member Ends:

doActivity, state

Owned Association Ends

+ **state** : State [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_effect_transition

Member Ends:

effect, transition

Owned Association Ends

+ **transition** : Transition [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_entry_connectionPointReference

Member Ends:

entry, connectionPointReference

Owned Association Ends

+ **connectionPointReference** : ConnectionPointReference [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_entry_state

Member Ends:

entry, state

Owned Association Ends

+ **state** : State [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_exit_connectionPointReference

Member Ends:

exit, connectionPointReference

Owned Association Ends

+ **connectionPointReference** : ConnectionPointReference [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_exit_state

Member Ends:

exit, state

Owned Association Ends

+ **state** : State [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_extendedRegion_region

Member Ends:

extendedRegion, region

Owned Association Ends

+ **region** : Region [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_extendedStateMachine_stateMachine

Member Ends:

extendedStateMachine, stateMachine

Owned Association Ends

+ **stateMachine** : StateMachine [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_guard_transition

Member Ends:

guard, transition

Owned Association Ends

+ **transition** : Transition [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_incoming_target

Member Ends:

incoming, target

Package UML::StateMachines::BehaviorStateMachines

Association A_outgoing_source

Member Ends:

outgoing, source

Package UML::StateMachines::BehaviorStateMachines

Association A_redefinedState_state

Member Ends:

redefinedState, state

Owned Association Ends

+ **state** : State [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_redefinedTransition_transition

Member Ends:

redefinedTransition, transition

Owned Association Ends

+ **transition** : Transition [0..1]

Package UML::StateMachines::BehaviorStateMachines

Association A_redefinitionContext_region

Member Ends:

redefinitionContext, region

Owned Association Ends

+ **region** : Region [0..*]

Package UML::StateMachines::BehaviorStateMachines

Association A_redefinitionContext_state

Member Ends:

redefinitionContext, state

Owned Association Ends

+ **state** : State [0..*]

Package UML::StateMachines::BehaviorStateMachines

Association A_redefinitionContext_transition

Member Ends:

redefinitionContext, transition

Owned Association Ends

+ **transition** : Transition [0..*]

Package UML::StateMachines::BehaviorStateMachines

Association A_region_state

Member Ends:

region, state

Package UML::StateMachines::BehaviorStateMachines

Association A_region_stateMachine

Member Ends:

region, stateMachine

Package UML::StateMachines::BehaviorStateMachines

Association A_stateInvariant_owningState

Member Ends:

stateInvariant, owningState

Owned Association Ends

+ **owningState** : State [0..1] {subsets owner}

Package UML::StateMachines::BehaviorStateMachines

Association A_submachineState_submachine

Member Ends:

submachineState, submachine

Package UML::StateMachines::BehaviorStateMachines

Association A_subvertex_container

Member Ends:

subvertex, container

Package UML::StateMachines::BehaviorStateMachines

Association A_transition_container

Member Ends:

transition, container

Package UML::StateMachines::BehaviorStateMachines

Association A_trigger_transition

Member Ends:

trigger, transition

Owned Association Ends

+ **transition** : Transition [0..1]

Package UML::StateMachines::ProtocolStateMachines

Nesting Package:

StateMachines

Imported Packages:

Kernel

Merged Packages:

BehaviorStateMachines, Interfaces, Ports

Class Summary
Interface
Port
ProtocolConformance
ProtocolStateMachine
ProtocolTransition
Region
State

Association Summary
A_conformance_specificMachine
A_generalMachine_protocolConformance
A_postCondition_owningTransition
A_preCondition_protocolTransition
A_protocol_interface
A_protocol_port
A_referred_protocolTransition

Package UML::StateMachines::ProtocolStateMachines

Class Interface

Since an interface specifies conformance characteristics, it does not own detailed behavior specifications. Instead, interfaces may own a protocol state machine that specifies event sequences and pre/post conditions for the operations and receptions described by the interface.

Generalizations:

Classifier

Owned Association Ends

+ **protocol** : ProtocolStateMachine [0..1] {subsets ownedMember}

References a protocol state machine specifying the legal sequences of the invocation of the behavioral features described in the interface.

Package UML::StateMachines::ProtocolStateMachines

Class Port

A port has an associated protocol state machine.

Owned Association Ends

+ **protocol** : ProtocolStateMachine [0..1]

References an optional protocol state machine which describes valid interactions at this interaction point.

Package UML::StateMachines::ProtocolStateMachines

Class ProtocolConformance

Protocol state machines can be redefined into more specific protocol state machines, or into behavioral state machines. Protocol conformance declares that the specific protocol state machine specifies a protocol that conforms to the general state machine one, or that the specific behavioral state machine abide by the protocol of the general protocol state machine.

Generalizations:

DirectedRelationship

Owned Association Ends

+ **generalMachine** : ProtocolStateMachine [1..1] {subsets target}

Specifies the protocol state machine to which the specific state machine conforms.

+ **specificMachine** : ProtocolStateMachine [1..1] {subsets source, subsets owner}

Specifies the state machine which conforms to the general state machine.

Package UML::StateMachines::ProtocolStateMachines

Class ProtocolStateMachine

A protocol state machine is always defined in the context of a classifier. It specifies which operations of the classifier can be called in which state and under which condition, thus specifying the allowed call sequences on the classifier's operations. A protocol state machine presents the possible and permitted transitions on the instances of its context classifier, together with the operations which carry the transitions. In this manner, an instance lifecycle can be created for a classifier, by specifying the order in which the operations can be activated and the states through which an instance progresses during its existence.

Generalizations:

StateMachine

Owned Association Ends

+ **conformance** : ProtocolConformance [0..*] {subsets ownedElement}

Conformance between protocol state machines.

Constraints

classifier_context

A protocol state machine must only have a classifier context, not a behavioral feature context.

expression (OCL): (not context->isEmpty( )) and specification->isEmpty()

deep_or_shallow_history

Protocol state machines cannot have deep or shallow history pseudostates.

expression (OCL): region->forAll (r | r.subvertex->forAll (v | v.oclIsKindOf(Psuedostate) implies ((v.kind <> #deepHistory) and (v.kind <> #shallowHistory)))))

entry_exit_do

The states of a protocol state machine cannot have entry, exit, or do activity actions.

expression (OCL): region->forAll(r | r.subvertex->forAll(v | v.oclIsKindOf(State) implies (v.entry ->isEmpty() and v.exit->isEmpty() and v.doActivity->isEmpty())))

ports_connected

If two ports are connected, then the protocol state machine of the required interface (if defined) must be conformant to the protocol state machine of the provided interface (if defined).

expression (OCL): true

protocol_transitions

All transitions of a protocol state machine must be protocol transitions. (transitions as extended by the ProtocolStateMachines package)

Package UML::StateMachines::ProtocolStateMachines

Class ProtocolStateMachine

expression (OCL): region->forAll(r | r.transition->forAll(t | t.oclIsTypeOf(ProtocolTransition)))

Package UML::StateMachines::ProtocolStateMachines

Class ProtocolTransition

A protocol transition specifies a legal transition for an operation. Transitions of protocol state machines have the following information: a pre condition (guard), on trigger, and a post condition. Every protocol transition is associated to zero or one operation (referred BehavioralFeature) that belongs to the context classifier of the protocol state machine.

Generalizations:

Transition

Owned Association Ends

+ **postCondition** : Constraint [0..1] {subsets ownedRule}

Specifies the post condition of the transition which is the condition that should be obtained once the transition is triggered. This post condition is part of the post condition of the operation connected to the transition.

+ **preCondition** : Constraint [0..1] {subsets guard}

Specifies the precondition of the transition. It specifies the condition that should be verified before triggering the transition. This guard condition added to the source state will be evaluated as part of the precondition of the operation referred by the transition if any.

+ /**referred** : Operation [0..*] {readOnly}

This association refers to the associated operation. It is derived from the operation of the call trigger when applicable.

Constraints

associated_actions

A protocol transition never has associated actions.

expression (OCL): effect->isEmpty()

belongs_to_psm

A protocol transition always belongs to a protocol state machine.

expression (OCL): container.belongsToPSM()

refers_to_operation

If a protocol transition refers to an operation (i. e. has a call trigger corresponding to an operation), then that operation should apply to the context classifier of the state machine of the protocol transition.

Package UML::StateMachines::ProtocolStateMachines

Class ProtocolTransition

expression (OCL): true

Package UML::StateMachines::ProtocolStateMachines

Class Region

Operations

+ **belongsToPSM** () : Boolean [1..1] {query}

The operation belongsToPSM () checks if the region belongs to a protocol state machine

body (OCL): result = if not stateMachine->isEmpty() then oclIsTypeOf(ProtocolStateMachine) else if not state->isEmpty() then state.container.belongsToPSM () else false

Package UML::StateMachines::ProtocolStateMachines

Class State

The states of protocol state machines are exposed to the users of their context classifiers. A protocol state represents an exposed stable situation of its context classifier: when an instance of the classifier is not processing any operation, users of this instance can always know its state configuration.

Generalizations:

Namespace

Package UML::StateMachines::ProtocolStateMachines

Association A_conformance_specificMachine

Member Ends:

conformance, specificMachine

Package UML::StateMachines::ProtocolStateMachines

Association A_generalMachine_protocolConformance

Member Ends:

generalMachine, protocolConformance

Owned Association Ends

+ **protocolConformance** : ProtocolConformance [0..*]

Package UML::StateMachines::ProtocolStateMachines

Association A_postCondition_owningTransition

Member Ends:

postCondition, owningTransition

Owned Association Ends

+ **owningTransition** : ProtocolTransition [0..1] {subsets owner}

Package UML::StateMachines::ProtocolStateMachines

Association A_preCondition_protocolTransition

Member Ends:

preCondition, protocolTransition

Owned Association Ends

+ **protocolTransition** : ProtocolTransition [0..1]

Package UML::StateMachines::ProtocolStateMachines

Association A_protocol_interface

Member Ends:

protocol, interface

Owned Association Ends

+ **interface** : Interface [0..1] {subsets namespace}

Specifies the namespace in which the protocol state machine is defined.

Package UML::StateMachines::ProtocolStateMachines

Association A_protocol_port

Member Ends:

protocol, port

Owned Association Ends

+ **port** : Port [0..*]

Package UML::StateMachines::ProtocolStateMachines

Association A_referred_protocolTransition

Member Ends:

referred, protocolTransition

Owned Association Ends

+ **protocolTransition** : ProtocolTransition [0..*]

Package UML::UseCases

Nesting Package:

UML

Merged Packages:

BasicBehaviors

Class Summary
Actor
Classifier
Extend
ExtensionPoint
Include
UseCase

Association Summary
A_addition_include
A_condition_extend
A_extend_extension
A_extendedCase_extend
A_extensionLocation_extension
A_extensionPoint_useCase
A_include_includingCase
A_ownedUseCase_classifier
A_subject_useCase

Package UML::UseCases

Class Actor

An actor specifies a role played by a user or any other system that interacts with the subject.

Generalizations:

BehavioredClassifier

Constraints

associations

An actor can only have associations to use cases, components and classes. Furthermore these associations must be binary.

expression (OCL): self.ownedAttribute->forAll (a | (a.association->notEmpty()) implies ((a.association.memberEnd.size() = 2) and (a.opposite.class.oclIsKindOf(UseCase) or (a.opposite.class.oclIsKindOf(Class) and not a.opposite.class.oclIsKindOf(Behavior))))

must_have_name

An actor must have a name.

expression (OCL): name->notEmpty()

Package UML::UseCases

Class Classifier

A classifier has the capability to own use cases. Although the owning classifier typically represents the subject to which the owned use cases apply, this is not necessarily the case. In principle, the same use case can be applied to multiple subjects, as identified by the subject association role of a use case.

Generalizations:

Namespace

Owned Association Ends

+ **ownedUseCase** : UseCase [0..*] {subsets ownedMember}

References the use cases owned by this classifier.

+ **useCase** : UseCase [0..*]

The set of use cases for which this Classifier is the subject.

Package UML::UseCases

Class Extend

A relationship from an extending use case to an extended use case that specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the extended use case.

Generalizations:

DirectedRelationship, NamedElement

Owned Association Ends

+ **condition** : Constraint [0..1] {subsets ownedElement}

References the condition that must hold when the first extension point is reached for the extension to take place. If no constraint is associated with the extend relationship, the extension is unconditional.

+ **extendedCase** : UseCase [1..1] {subsets target}

References the use case that is being extended.

+ **extension** : UseCase [1..1] {subsets source}

References the use case that represents the extension and owns the extend relationship.

+ **extensionLocation** : ExtensionPoint [1..*] {ordered}

An ordered list of extension points belonging to the extended use case, specifying where the respective behavioral fragments of the extending use case are to be inserted. The first fragment in the extending use case is associated with the first extension point in the list, the second fragment with the second point, and so on. (Note that, in most practical cases, the extending use case has just a single behavior fragment, so that the list of extension points is trivial.)

Constraints

extension_points

The extension points referenced by the extend relationship must belong to the use case that is being extended.

expression (OCL): extensionLocation->forAll (xp | extendedCase.extensionPoint->includes(xp))

Package UML::UseCases

Class ExtensionPoint

An extension point identifies a point in the behavior of a use case where that behavior can be extended by the behavior of some other (extending) use case, as specified by an extend relationship.

Generalizations:

RedefinableElement

Owned Association Ends

+ **useCase** : UseCase [1..1]

References the use case that owns this extension point.

Constraints

must_have_name

An ExtensionPoint must have a name.

expression (OCL): self.name->notEmpty ()

Package UML::UseCases

Class Include

An include relationship defines that a use case contains the behavior defined in another use case.

Generalizations:

DirectedRelationship, NamedElement

Owned Association Ends

+ **addition** : UseCase [1..1] {subsets target}

References the use case that is to be included.

+ **includingCase** : UseCase [1..1] {subsets source}

References the use case which will include the addition and owns the include relationship.

Package UML::UseCases

Class UseCase

A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.

Generalizations:

BehavioredClassifier

Owned Association Ends

+ **extend** : Extend [0..*] {subsets ownedMember}

References the Extend relationships owned by this use case.

+ **extensionPoint** : ExtensionPoint [0..*] {subsets ownedMember}

References the ExtensionPoints owned by the use case.

+ **include** : Include [0..*] {subsets ownedMember}

References the Include relationships owned by this use case.

+ **subject** : Classifier [0..*]

References the subjects to which this use case applies. The subject or its parts realize all the use cases that apply to this subject. Use cases need not be attached to any specific subject, however. The subject may, but need not, own the use cases that apply to it.

Operations

+ **allIncludedUseCases** () : UseCase [0..*] {query}

The query allIncludedUseCases() returns the transitive closure of all use cases (directly or indirectly) included by this use case.

body (OCL): result = self.include->union(self.include->collect(in | in.allIncludedUseCases()))

Constraints

binary_associations

UseCases can only be involved in binary Associations.

expression (OCL): true

cannot_include_self

A use case cannot include use cases that directly or indirectly include it.

Package UML::UseCases

Class UseCase

expression (OCL): not self.allIncludedUseCases()->includes(self)

must_have_name

A UseCase must have a name.

expression (OCL): self.name -> notEmpty ()

no_association_to_use_case

UseCases can not have Associations to UseCases specifying the same subject.

expression (OCL): true

Package UML::UseCases

Association A_addition_include

Member Ends:

addition, include

Owned Association Ends

+ **include** : Include [0..*]

Package UML::UseCases

Association A_condition_extend

Member Ends:

condition, extend

Owned Association Ends

+ **extend** : Extend [0..1]

Package UML::UseCases

Association A_extend_extension

Member Ends:

extend, extension

Package UML::UseCases

Association A_extendedCase_extend

Member Ends:

extendedCase, extend

Owned Association Ends

+ **extend** : Extend [0..*]

Package UML::UseCases

Association A_extensionLocation_extension

Member Ends:

extensionLocation, extension

Owned Association Ends

+ **extension** : Extend [0..*]

Package UML::UseCases

Association A_extensionPoint_useCase

Member Ends:

extensionPoint, useCase

Package UML::UseCases

Association A_include_includingCase

Member Ends:

include, includingCase

Package UML::UseCases

Association A_ownedUseCase_classifier

Member Ends:

ownedUseCase, classifier

Owned Association Ends

+ **classifier** : Classifier [0..1]

Package UML::UseCases

Association A_subject_useCase

Member Ends:

subject, useCase