1 Package uppaal

Package Overview Contains UPPAAL-specific sub-packages.

1.1 Class NTA

Overview A network of timed automata as basic input to UPPAAL.

Super Types

- NamedElement see Section 2.2 on Page 3 ,
- CommentableElement see Section 2.1 on Page 3

Class References Class NTA has the following references:

bool : PredefinedType see Section 9.4 on Page 38
The predefined type 'bool'.

chan : PredefinedType see Section 9.4 on Page 38
The predefined type 'chan'.

clock : PredefinedType see Section 9.4 on Page 38
The predefined type 'clock'.

globalDeclarations : GlobalDeclarations [0..1] see Section 3.12 on Page 8
The global declarations for the NTA.

int : PredefinedType see Section 9.4 on Page 38
The predefined type 'int'.

systemDeclarations : SystemDeclarations see Section 3.17 on Page 10

The declarations of process instantiations.

template : Template [1..*] see Section 8.9 on Page 35
The Timed Automata templates of the NTA.

void : PredefinedType see Section 9.4 on Page 38
The predefined dummy type 'void'.

Class Constraints Class NTA has the following constraints:

MatchingIntDetails:

```
                    (not self.int.oclIsUndefined())
implies
((self.int.type = types::BuiltInType::INT) and (
    self.int.name.equalsIgnoreCase('int')))
```

MatchingBoolDetails:

```
                    (not self.bool.oclIsUndefined())
    implies
    ((self.bool.type = types::BuiltInType::BOOL) and
        (self.bool.name.equalsIgnoreCase('bool')))
```

MatchingClockDetails:

```
                    (not self.clock.oclIsUndefined())
    implies
    ((self.clock.type = types::BuiltInType::CLOCK)
        and (self.clock.name.equalsIgnoreCase('clock')
        ))
```

MatchingChanDetails:

```
                    (not self.chan.oclIsUndefined())
    implies
    ((self.chan.type = types::BuiltInType::CHAN) and
        (self.chan.name.equalsIgnoreCase('chan')))
```

MatchingVoidDetails:

```
                    (not self.void.oclIsUndefined())
    implies
    ((self.void.type = types::BuiltInType::VOID) and
        (self.void.name.equalsIgnoreCase('void')))
```

UniqueTemplateNames:

```
                    self.template->isUnique(name)
```

2 Package `uppaal::core`

Package Overview Contains abstract general purpose classes.

2.1 Abstract Class `CommentableElement`

Overview Abstract base class for commentable model elements.

Class Attributes Class `CommentableElement` has the following attributes:

comment : EString [0..1]
The comment for the model element.

2.2 Abstract Class `NamedElement`

Overview Abstract base class for named model elements.

Class Attributes Class `NamedElement` has the following attributes:

name : EString
The name of the model element..

Class Constraints Class `NamedElement` has the following constraints:

NoWhitespace:

```
self.name.characters()->excludes
    (' ')
```

NoDigitStart:

```
Set{0..9}->excludes(self.name.
    characters()->first())
```

3 Package `uppaal::declarations`

Package Overview Support for all kinds of declarations, e.g. types, functions, or variables.

3.1 Class `ArrayInitializer`

Overview An initializer for array variables, referring to multiple sub-initializers.

Super Types

- Initializer see Section 3.14 on Page 9

Class References Class `ArrayInitializer` has the following references:

initializer : Initializer [1..∗] see Section 3.14 on Page 9
A number of sub-initializers, each one representing the initial value for one array index.

3.2 Enumeration `CallType`

Overview Represents call-by-value or call-by-reference parameters.

Enum Properties Enumeration `CallType` has the following literals:

CALL_BY_VALUE = 0
CALL_BY_REFERENCE = 1

3.3 Class `ChannelVariableDeclaration`

Overview A declaration of synchronization channel variables.

Super Types

- VariableDeclaration see Section 3.23 on Page 14

Class Attributes Class `ChannelVariableDeclaration` has the following attributes:

broadcast : EBoolean
Specifies whether the declared synchronization channels use broadcast.

urgent : EBoolean
Specifies the urgency of the declared synchronization channels.

Class Constraints Class `ChannelVariableDeclaration` has the following constraints:

MatchingType:

```
                    (not self.typeDefinition.
                        oclIsUndefined())
implies
self.typeDefinition.baseType = types::BuiltInType
    ::CHAN
```

3.4 Class `ClockVariableDeclaration`

Overview A declaration of clock variables.

Super Types

- VariableDeclaration see Section 3.23 on Page 14

Class Constraints Class `ClockVariableDeclaration` has the following constraints:

MatchingType:

```
                    (not self.typeDefinition.
                        oclIsUndefined())
implies
self.typeDefinition.baseType = types::BuiltInType
    ::CLOCK
```

3.5 Class `DataVariableDeclaration`

Overview A declaration of data variables.

Super Types

- VariableDeclaration see Section 3.23 on Page 14

Class Attributes Class `DataVariableDeclaration` has the following attributes:

prefix : DataVariablePrefix see Section 3.6 on Page 6
The prefix of the data variable declaration.

Class Constraints Class `DataVariableDeclaration` has the following constraints:

MatchingType:

```
                    (not self.typeDefinition.
                        oclIsUndefined())
implies
(self.typeDefinition.baseType <> types::
    BuiltInType::CHAN
and
self.typeDefinition.baseType <> types::
    BuiltInType::CLOCK)
```

3.6 Enumeration `DataVariablePrefix`

Overview Prefixes for data variables with base type 'int' or 'bool'.

Enum Properties Enumeration `DataVariablePrefix` has the following literals:

NONE = 0

CONST = 1

META = 2

3.7 Abstract Class `Declaration`

Overview Abstract base class representing a variable, function, or type declaration.

3.8 Abstract Class `Declarations`

Overview Represents a set of variable, type, function, or template declarations, that are either global, local to a template, local to a block, or system declarations.

Class References Class `Declarations` has the following references:

declaration : Declaration [0..*] see Section 3.7 on Page 6
The single declarations.

Class Constraints Class `Declarations` has the following constraints:

UniqueFunctionNames:

```
self.declaration->select(
    oclIsKindOf(
    FunctionDeclaration)).
    oclAsType(FunctionDeclaration)
    ->collect(function)->isUnique(
    name)
```

UniqueVariableNames:

```
self.declaration->select(
    oclIsKindOf(
    VariableDeclaration)).
    oclAsType(VariableDeclaration)
    ->collect(variable)->isUnique(
    name)
```

UniqueTypeNames:

```
self.declaration->select(
    oclIsKindOf(TypeDeclaration)).
    oclAsType(TypeDeclaration)->
    collect(type)->isUnique(name)
```

3.9 Class `ExpressionInitializer`

Overview An initializer that represents a single initial value by means of an expression.

Super Types

- Initializer see Section 3.14 on Page 9

Class References Class `ExpressionInitializer` has the following references:

expression : Expression see Section 6.11 on Page 23
The expression representing the initial value.

3.10 Class `Function`

Overview A function with a return type and optional parameters.

Super Types

- NamedElement see Section 2.2 on Page 3

Class References Class `Function` has the following references:

block : Block see Section 7.1 on Page 28
The block of statements representing the function body.

parameter : Parameter [0..*] see Section 3.16 on Page 10
The function's parameters.

returnType : TypeDefinition see Section 9.9 on Page 40
The return type of this function.

Class Constraints Class `Function` has the following constraints:

ReturnStatementExistsIfRequired:

```
                    ((not self.returnType.
                        oclIsUndefined()) and
    self.returnType.baseType <> types::BuiltInType::
        VOID)
    implies
    ((not self.block.oclIsUndefined()) and
    self.block.statement->exists(oclIsKindOf(
        statements::ReturnStatement)))
```

ValidReturnType:

```
                    (not returnType.oclIsUndefined())
    implies
    (returnType.baseType = types::BuiltInType::VOID
        or
     returnType.baseType = types::BuiltInType::INT or
     returnType.baseType = types::BuiltInType::BOOL)
```

UniqueParameterNames:

```
                    self.parameter->collect(
                        variableDeclaration)->collect(
                        variable)->isUnique(name)
```

3.11 Class `FunctionDeclaration`

Overview Declaration of a single function.

Super Types

- Declaration see Section 3.7 on Page 6

Class References Class `FunctionDeclaration` has the following references:

function : Function see Section 3.10 on Page 7
The return type of this function.

3.12 Class `GlobalDeclarations`

Overview Global declarations of an NTA.

Super Types

- Declarations see Section 3.8 on Page 6

Class References Class `GlobalDeclarations` has the following references:

channelPriority : ChannelPriority [0..1] see Section 4.2 on Page 15

The declaration of the synchronization channel priorities.

Class Constraints Class `GlobalDeclarations` has the following constraints:

NoTemplateDeclarations:

```
not self.declaration->exists(
    oclIsKindOf(system::
    TemplateDeclaration))
```

3.13 Abstract Class `Index`

Overview Abstract base-class for indexing variables or types.

3.14 Abstract Class `Initializer`

Overview An initializer specifies a variable's initial value.

3.15 Class `LocalDeclarations`

Overview Local declarations inside a template or block of statements.

Super Types

- Declarations see Section 3.8 on Page 6

Class Constraints Class `LocalDeclarations` has the following constraints:

NoTemplateDeclarations:

```
not self.declaration->exists(
    oclIsKindOf(system::
    TemplateDeclaration))
```

NoChannelDeclarations:

```
not self.declaration->exists(
    oclIsKindOf(
    ChannelVariableDeclaration))
```

3.16 Class `Parameter`

Overview A parameter of a function or template.

Class Attributes Class `Parameter` has the following attributes:

callType : CallType [0..1] see Section 3.2 on Page 4
Specifies whether call-by-value or call-by-reference semantics should be applied.

Class References Class `Parameter` has the following references:

variableDeclaration : VariableDeclaration see Section 3.23 on Page 14

A variable declaration containing the variable that represents the parameter.

Class Constraints Class `Parameter` has the following constraints:

SingleVariable:

```
                    (not self.variableDeclaration.
                        oclIsUndefined())
implies
self.variableDeclaration.variable->size() <= 1
```

3.17 Class `SystemDeclarations`

Overview System declarations consisting of process instantiations.

Super Types

- Declarations see Section 3.8 on Page 6

Class References Class `SystemDeclarations` has the following references:

progressMeasure : ProgressMeasure [0..1] see Section 5.2 on Page 17

The optional progress measure section.

system : System see Section 5.3 on Page 18
The system section describing the process instantiations.

Class Constraints Class `SystemDeclarations` has the following constraints:

UniqueTemplateNames:

```
self.declaration->select(
    oclIsKindOf(system::
    TemplateDeclaration)).
    oclAsType(system::
    TemplateDeclaration)->collect(
    declaredTemplate)->isUnique(
    name)
```

NoChannelDeclarations:

```
not self.declaration->exists(
    oclIsKindOf(
    ChannelVariableDeclaration))
```

3.18 Class `TypeDeclaration`

Overview A declaration of one or more types.

Super Types

- Declaration see Section 3.7 on Page 6

Class References Class `TypeDeclaration` has the following references:

type : DeclaredType [1..*] see Section 9.2 on Page 37
The types declared by this type declaration.

typeDefinition : TypeDefinition see Section 9.9 on Page 40
The type definition for declared types.

Class Constraints Class `TypeDeclaration` has the following constraints:

UniqueTypeNames:

```
self.type->isUnique(name)
```

3.19 Class `TypeIndex`

Overview An index specified by a bounded integer-based type.

Super Types

- Index see Section 3.13 on Page 9

Class References Class `TypeIndex` has the following references:

typeDefinition : TypeDefinition see Section 9.9 on Page 40

An integer-based type representing size and range of the indexed type or variable.

Class Constraints Class `TypeIndex` has the following constraints:

IntegerBasedIndex:

```
                    (not self.typeDefinition.
                        oclIsUndefined())
implies
self.typeDefinition.baseType = types::BuiltInType
    ::INT
```

3.20 Class `ValueIndex`

Overview An index specified by an expression value.

Super Types

- Index see Section 3.13 on Page 9

Class References Class `ValueIndex` has the following references:

sizeExpression : Expression see Section 6.11 on Page 23

An integer-based expression representing size and range of the indexed type or variable.

3.21 Class `Variable`

Overview A typed variable.

Super Types

- NamedElement see Section 2.2 on Page 3

Class References Class `Variable` has the following references:

container : VariableContainer see Section 3.22 on Page 13

The container of this variable.

index : Index [0..*] see Section 3.13 on Page 9

A set of array indexes for the variable.

initializer : Initializer [0..1] see Section 3.14 on Page 9

Represents the variable's initial value.

/typeDefinition : TypeDefinition see Section 9.9 on Page 40

derivation:

```
                    if self.container.
                        oclIsUndefined()
    then null
    else
    self.container.typeDefinition
    endif
```

The type definition of this variable.

Class Constraints Class `Variable` has the following constraints:

NoInitializerForClockAndChannelVariables:

```
                    ((not self.typeDefinition.
                        oclIsUndefined()) and
    (self.typeDefinition.baseType = types::
        BuiltInType::CHAN or
     self.typeDefinition.baseType = types::
        BuiltInType::CLOCK))
     implies self.initializer.oclIsUndefined()
```

NoArrayInitializerForSingleVariables:

```
                    (not self.initializer.
                        oclIsUndefined() and self.
                        initializer.oclIsKindOf(
                        ArrayInitializer)) implies
                        self.index->notEmpty()
```

3.22 Abstract Class `VariableContainer`

Overview Abstract base class for objects containing variables like variable declarations, iterations, quantifications or selections.

Class References Class `VariableContainer` has the following references:

typeDefinition : TypeDefinition see Section 9.9 on Page 40

The type definition for the contained variables.

variable : Variable [1..*] see Section 3.21 on Page 12

The contained variables.

Class Constraints Class `VariableContainer` has the following constraints:

NoVoidVariables:

```
                    (not self.typeDefinition.
                        oclIsUndefined())
implies
self.typeDefinition.baseType <> types::
    BuiltInType::VOID
```

UniqueVariableNames:

```
self.variable->isUnique(name)
```

3.23 Abstract Class `VariableDeclaration`

Overview A declaration of one or more variables.

Super Types

- Declaration see Section 3.7 on Page 6 ,
- VariableContainer see Section 3.22 on Page 13

4 Package uppaal::declarations::global

Package Overview Contains special classes that are relevant for the global declarations.

4.1 Class ChannelList

Overview A list of synchronization channel variables, used to assign these channels a common priority.

Super Types

- ChannelPriorityItem see Section 4.3 on Page 16

Class References Class ChannelList has the following references:

channelExpression : VariableExpression [1..*] see Section 6.24 on Page 27

The variable expressions representing the synchronization channels inside the channel list.

Class Constraints Class ChannelList has the following constraints:

ChannelVariablesOnly:

```
                self.channelExpression->forAll(
        (not variable.typeDefinition.
            oclIsUndefined()) implies variable.
            typeDefinition.baseType = types::
            BuiltInType::CHAN
)
```

4.2 Class ChannelPriority

Overview A priority ordering for synchronization channels.

Class References Class ChannelPriority has the following references:

item : ChannelPriorityItem [1..*] see Section 4.3 on Page 16

The items of the channel priority ordering.

Class Constraints Class ChannelPriority has the following constraints:

AtMostOneDefaultItem:

```
self.item->select(oclIsKindOf(
    DefaultChannelPriority))->size
    () <= 1
```

EachChannelContainedAtMostOnce:

```
self.item->select(oclIsKindOf(
    ChannelList)).oclAsType(
    ChannelList)->collect(
    channelExpression)->isUnique(
    variable)
```

4.3 Abstract Class `ChannelPriorityItem`

Overview Abstract base class for items inside a channel priority.

4.4 Class `DefaultChannelPriority`

Overview A 'default' item inside a channel priority, representing all channels not listed explicitly.

Super Types

- ChannelPriorityItem see Section 4.3 on Page 16

5 Package uppaal::declarations::system

Package Overview Contains special classes that are relevant for the system declarations.

5.1 Class InstantiationList

Overview Represents a list of templates to be instantiated using a common priority.

Class References Class InstantiationList has the following references:

template : AbstractTemplate [1..*] see Section 8.1 on Page 32
The list of instantiations.

Class Constraints Class InstantiationList has the following constraints:

OnlyLegalParamsForPartialInstantiation:

```
                self.template->forAll(
        parameter->forAll(
                callType = declarations::CallType
                    ::CALL_BY_VALUE
                and
                ((not variableDeclaration.
                    oclIsUndefined())
                        implies
                 (variableDeclaration.
                     typeDefinition.oclIsKindOf(
                     types::RangeTypeSpecification
                     ) or
                  variableDeclaration.
                     typeDefinition.oclIsKindOf(
                     types::
                     ScalarTypeSpecification)))
        )
    )
```

5.2 Class ProgressMeasure

Overview A progress measure consisting of monotonically increasing expressions.

Class References Class ProgressMeasure has the following references:

expression : Expression [1..*] see Section 6.11 on Page 23
The progress measure expressions.

5.3 Class `System`

Overview A system contains declarations of template instantiations.

Class References Class `System` has the following references:

instantiationList : InstantiationList [1..*] see Section 5.1 on Page 17

A list of process instantiation sublists, ordered by decreasing priority. The templates referenced inside the sublists are instantiated to be part of the system at runtime.

Class Constraints Class `System` has the following constraints:

EachTemplateReferencedAtMostOnce:

```
self.instantiationList->collect(
    template)->isUnique(t :
    templates::AbstractTemplate |
    t)
```

5.4 Class `TemplateDeclaration`

Overview A declaration of a template redefinition.

Super Types

- Declaration see Section 3.7 on Page 6

Class References Class `TemplateDeclaration` has the following references:

argument : Expression [0..*] see Section 6.11 on Page 23

A number of arguments that describe how the referred template's parameters should be mapped towards the declared template's parameters.

declaredTemplate : RedefinedTemplate see Section 8.5 on Page 34

The template being declared.

Class Constraints Class `TemplateDeclaration` has the following constraints:

NumberOfArgumentsMatchesDeclaration:

```
(not self.declaredTemplate.
    oclIsUndefined() and not self.
    declaredTemplate.
    referredTemplate.
    oclIsUndefined())
implies
```

```
self.argument->size() = self.declaredTemplate.
    referredTemplate.parameter->size()
```

6 Package uppaal::expressions

Package Overview Introduces all kinds of expressions.

6.1 Class ArithmeticExpression

Overview A binary expression representing an arithemtic operation.

Super Types

- BinaryExpression see Section 6.5 on Page 21

Class Attributes Class ArithmeticExpression has the following attributes:

operator : ArithmeticOperator see Section 6.2 on Page 20
The arithmetic operator to be applied.

6.2 Enumeration ArithmeticOperator

Overview Representing all arithmetic operators.

Enum Properties Enumeration ArithmeticOperator has the following literals:

ADD = 0
SUBTRACT = 1
MULTIPLICATE = 2
DIVIDE = 3
MODULO = 4

6.3 Class AssignmentExpression

Overview A binary assignment expression using a specific assignment operator.

Super Types

- BinaryExpression see Section 6.5 on Page 21

Class Attributes Class AssignmentExpression has the following attributes:

operator : AssignmentOperator see Section 6.4 on Page 21
The operator for the assignment.

6.4 Enumeration `AssignmentOperator`

Overview Representing all assignment operators.

Enum Properties Enumeration `AssignmentOperator` has the following literals:

EQUAL = 0
COLON_EQUAL = 1
PLUS_EQUAL = 2
MINUS_EQUAL = 3
TIMES_EQUAL = 4
DIVIDE_EQUAL = 5
MODULO_EQUAL = 6

6.5 Abstract Class `BinaryExpression`

Overview Abstract base class for all binary expressions connecting two sub-expressions.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `BinaryExpression` has the following references:

firstExpr : Expression see Section 6.11 on Page 23
The first sub-expression.

secondExpr : Expression see Section 6.11 on Page 23
The second sub-expression.

6.6 Class `CompareExpression`

Overview A comparison between two expression values using a specific comparison operator.

Super Types

- BinaryExpression see Section 6.5 on Page 21

Class Attributes Class `CompareExpression` has the following attributes:

operator : CompareOperator see Section 6.7 on Page 22
The comparison operator to be applied.

6.7 Enumeration `CompareOperator`

Overview Representing all comparison operators.

Enum Properties Enumeration `CompareOperator` has the following literals:

EQUAL = 0
GREATER = 1
GREATER_OR_EQUAL = 2
LESS = 3
LESS_OR_EQUAL = 4
UNEQUAL = 5

6.8 Class `ConditionExpression`

Overview An expression representing a conditional redirection to one of the sub-expressions. Uses tokens '?' and ':' for delimitation.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `ConditionExpression` has the following references:

elseExpression : Expression see Section 6.11 on Page 23
The else-expression.

ifExpression : Expression see Section 6.11 on Page 23
The boolean if-expression.

thenExpression : Expression see Section 6.11 on Page 23
The then-expression.

6.9 Class `ConjunctionExpression`

Overview A conjunction between logical expressions.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `ConjunctionExpression` has the following references:

conjunctionExpression : Expression [2..*] see Section 6.11 on Page 23

The sub-expressions of the conjunction.

6.10 Class `DisjunctionExpression`

Overview A disjunction between logical expressions.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `DisjunctionExpression` has the following references:

disjunctionExpression : Expression [2..*] see Section 6.11 on Page 23

A sub-expressions of the disjunction.

6.11 Abstract Class `Expression`

Overview Abstract base class for all kinds of expressions.

6.12 Class `FieldAccessExpression`

Overview An expression used to access a field of a 'struct' variable. Uses a dot for delimination between variable and field.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `FieldAccessExpression` has the following references:

fieldExpression : VariableExpression see Section 6.24 on Page 27
An expression that refers to a field of the 'struct' variable.

pathExpression : Expression see Section 6.11 on Page 23
An expression that represents a path to a 'struct' variable.

6.13 Class `FunctionCallExpression`

Overview An expression representing a call to a function.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `FunctionCallExpression` has the following references:

argument : Expression [0..*] see Section 6.11 on Page 23
A set of expressions representing the argument values for the function call. Must conform to the parameters of the function declaration.

function : Function see Section 3.10 on Page 7

The function to be called.

Class Constraints Class `FunctionCallExpression` has the following constraints:

NumberOfArgumentsMatchesDeclaration:

```
                    (not self.function.oclIsUndefined
                        ())
implies
self.argument->size() = self.function.parameter->
    size()
```

6.14 Class `ImplicationExpression`

Overview An implication between two logical expressions.

Super Types

- BinaryExpression see Section 6.5 on Page 21

6.15 Class `IncrementDecrementExpression`

Overview An expression describing increment (++) or decrement (—) of an integer-based expression.

Super Types

- Expression see Section 6.11 on Page 23

Class Attributes Class `IncrementDecrementExpression` has the following attributes:

operator : IncrementDecrementOperator see Section 6.16 on Page 25

Specifies increment or decrement.

position : IncrementDecrementPosition see Section 6.17 on Page 25

Specifies pre- or post-evaluation.

Class References Class `IncrementDecrementExpression` has the following references:

expression : Expression see Section 6.11 on Page 23

The expression to be incremented or decremented.

6.16 Enumeration `IncrementDecrementOperator`

Overview Representing increment and decrement operators.

Enum Properties Enumeration `IncrementDecrementOperator` has the following literals:

INCREMENT = 0
DECREMENT = 1

6.17 Enumeration `IncrementDecrementPosition`

Overview Representing pre- or post-processing inside increment/decrement expressions.

Enum Properties Enumeration `IncrementDecrementPosition` has the following literals:

PRE = 0
POST = 1

6.18 Class `LiteralExpression`

Overview An expression referring to a literal of any type.

Super Types

- Expression see Section 6.11 on Page 23

Class Attributes Class `LiteralExpression` has the following attributes:

text : EString
The textual description of the literal.

6.19 Class `MinusExpression`

Overview An inversion of an integer-based expression using the '-' token.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `MinusExpression` has the following references:

invertedExpression : Expression see Section 6.11 on Page 23
The expression negated by this negation.

6.20 Class `NegationExpression`

Overview A negation of an expression.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `NegationExpression` has the following references:

negatedExpression : Expression see Section 6.11 on Page 23
The expression negated by this negation.

6.21 Class `PlusExpression`

Overview A confirmation of an integer-based expression using the '+' token.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `PlusExpression` has the following references:

confirmedExpression : Expression see Section 6.11 on Page 23
The expression negated by this negation.

6.22 Class `QuantificationExpression`

Overview A quantification expression introducing a quantified variable.

Super Types

- Expression see Section 6.11 on Page 23 ,
- VariableContainer see Section 3.22 on Page 13

Class Attributes Class `QuantificationExpression` has the following attributes:

quantifier : Quantifier see Section 6.23 on Page 27
The quantifier to be applied.

Class References Class `QuantificationExpression` has the following references:

expression : Expression see Section 6.11 on Page 23
The quantified expression.

Class Constraints Class `QuantificationExpression` has the following constraints:

SingleVariable:

```
self.variable->size() <= 1
```

6.23 Enumeration `Quantifier`

Overview Representing existential and universal quantification.

Enum Properties Enumeration `Quantifier` has the following literals:

EXISTENTIAL = 0
UNIVERSAL = 1

6.24 Class `VariableExpression`

Overview An expression referring to a variable.

Super Types

- Expression see Section 6.11 on Page 23

Class References Class `VariableExpression` has the following references:

index : Expression [0..*] see Section 6.11 on Page 23
A set of expressions that refer to the array indexes of the variable.

variable : Variable see Section 3.21 on Page 12
The referred variable.

7 Package `uppaal::statements`

Package Overview Support for statements inside functions.

7.1 Class `Block`

Overview A block of one or more statements.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `Block` has the following references:

declarations : LocalDeclarations [0..1] see Section 3.15 on Page 9
The local declarations for the function's body.

statement : Statement [1..*] see Section 7.9 on Page 31
The statements inside the funtion's body.

Class Constraints Class `Block` has the following constraints:

DataVariableDeclarationsOnly:

```
                    (not self.declarations.
                        oclIsUndefined())
implies
(self.declarations.declaration->forAll(
    oclIsKindOf(declarations::
    DataVariableDeclaration)))
```

7.2 Class `DoWhileLoop`

Overview A do-while-loop statement.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `DoWhileLoop` has the following references:

expression : Expression see Section 6.11 on Page 23
A boolean expression for the while loop.

statement : Statement see Section 7.9 on Page 31
The statement to be evaluated for every value.

7.3 Class `EmptyStatement`

Overview An empty statement represented by a semicolon only.

Super Types

- Statement see Section 7.9 on Page 31

7.4 Class `ExpressionStatement`

Overview A statement that refers to an arbitrary expression.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `ExpressionStatement` has the following references:

expression : Expression see Section 6.11 on Page 23
The expression this statement refers to.

7.5 Class `ForLoop`

Overview A for-loop statement.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `ForLoop` has the following references:

condition : Expression see Section 6.11 on Page 23
The condition of the for loop, represented by a boolean expression.

initialization : Expression see Section 6.11 on Page 23
The initialization expression of the for loop.

iteration : Expression see Section 6.11 on Page 23
The iteration statements of the for loop.

statement : Statement see Section 7.9 on Page 31
The statement to be evaluated for every value.

7.6 Class `IfStatement`

Overview An if-then-else statement.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `IfStatement` has the following references:

elseStatement : Statement [0..1] see Section 7.9 on Page 31
The else-statement.

ifExpression : Expression see Section 6.11 on Page 23
The boolean if-expression.

thenStatement : Statement see Section 7.9 on Page 31
The then-statement.

7.7 Class `Iteration`

Overview An iteration over all possible values of a bounded type using the 'for' keyword.

Super Types

- Statement see Section 7.9 on Page 31 ,
- VariableContainer see Section 3.22 on Page 13

Class References Class `Iteration` has the following references:

statement : Statement see Section 7.9 on Page 31
The statement to be evaluated for every value.

Class Constraints Class `Iteration` has the following constraints:

SingleVariable:

```
self.variable->size() <= 1
```

7.8 Class `ReturnStatement`

Overview A statement used to return from a function's body, optionally carrying a return value.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `ReturnStatement` has the following references:

returnExpression : Expression [0..1] see Section 6.11 on Page 23
The expression representing the return value.

7.9 Abstract Class `Statement`

Overview Abstract base-class for all statements inside a function's body.

7.10 Class `WhileLoop`

Overview A while-loop statement.

Super Types

- Statement see Section 7.9 on Page 31

Class References Class `WhileLoop` has the following references:

expression : Expression see Section 6.11 on Page 23
A boolean expression for the while loop.

statement : Statement see Section 7.9 on Page 31
The statement to be evaluated for every value.

8 Package uppaal::templates

Package Overview Support for timed automata templates consisting of locations and edges.

8.1 Abstract Class AbstractTemplate

Overview Abstract base class for ordinary timed automata templates as well as redefined templates.

Super Types

- NamedElement see Section 2.2 on Page 3 ,
- CommentableElement see Section 2.1 on Page 3

Class References Class AbstractTemplate has the following references:

parameter : Parameter [0..*] see Section 3.16 on Page 10
The parameter declarations of the template.

Class Constraints Class AbstractTemplate has the following constraints:

UniqueParameterNames:

```
self.parameter->collect(
    variableDeclaration)->collect(
    variable)->isUnique(name)
```

8.2 Class Edge

Overview An edge connecting two locations inside a template.

Super Types

- LinearElement see Section 10.2 on Page 42 ,
- CommentableElement see Section 2.1 on Page 3 ,
- ColoredElement see Section 10.1 on Page 42

Class References Class Edge has the following references:

guard : Expression [0..1] see Section 6.11 on Page 23
The guard expression of the edge.

parentTemplate : Template see Section 8.9 on Page 35
The parent template containing the edge.

selection : Selection [0..*] see Section 8.6 on Page 34
A set of non-deterministic value selections.

source : Location see Section 8.3 on Page 33
The source location of the edge.

synchronization : Synchronization [0..1] see Section 8.7 on Page 35

A synchronization performed when the edge fires.

target : Location see Section 8.3 on Page 33
The target location of the edge.

update : Expression [0..*] see Section 6.11 on Page 23
A set of update expressions for the edge, evaluated if the edge fires.

Class Constraints Class `Edge` has the following constraints:

UniqueParentTemplate:

```
                    (not (self.source.oclIsUndefined
                        () or self.target.
                        oclIsUndefined()))
implies
self.source.parentTemplate = self.target.
    parentTemplate
```

8.3 Class `Location`

Overview A location inside a template.

Super Types

- NamedElement see Section 2.2 on Page 3 ,
- CommentableElement see Section 2.1 on Page 3 ,
- PlanarElement see Section 10.3 on Page 42 ,
- ColoredElement see Section 10.1 on Page 42

Class Attributes Class `Location` has the following attributes:

locationTimeKind : LocationKind see Section 8.4 on Page 34
Specifies the kind of location (default, urgent, or committed).

Class References Class `Location` has the following references:

invariant : Expression [0..1] see Section 6.11 on Page 23
A boolean expression representing the location's invariant.

parentTemplate : Template see Section 8.9 on Page 35
The parent template containing the location.

8.4 Enumeration `LocationKind`

Overview Location types.

Enum Properties Enumeration `LocationKind` has the following literals:

NORMAL = 0
URGENT = 1
COMMITED = 2

8.5 Class `RedefinedTemplate`

Overview A template resulting from redefinition of another referred template, altering its name and parametrization.

Super Types

- AbstractTemplate see Section 8.1 on Page 32

Class References Class `RedefinedTemplate` has the following references:

declaration : TemplateDeclaration see Section 5.4 on Page 18
The declaration of this template.

referredTemplate : AbstractTemplate see Section 8.1 on Page 32
The template that serves as basis for redefinition.

8.6 Class `Selection`

Overview A non-deterministic selection of a value from a range. The range is specified by a bounded type.

Super Types

- VariableContainer see Section 3.22 on Page 13

Class Constraints Class `Selection` has the following constraints:

SingleVariable:

```
self.variable->size() <= 1
```

IntegerBasedType:

```
(not self.typeDefinition.
    oclIsUndefined())
implies
self.typeDefinition.baseType = types::BuiltInType
    ::INT
```

8.7 Class `Synchronization`

Overview A sent or received synchronization between two templates using a specific synchronization channel.

Class Attributes Class `Synchronization` has the following attributes:

kind : SynchronizationKind see Section 8.8 on Page 35

The kind of synchronization (sent or received).

Class References Class `Synchronization` has the following references:

channelExpression : VariableExpression see Section 6.24 on Page 27

An expression representing the channel variable used for synchronization.

Class Constraints Class `Synchronization` has the following constraints:

ChannelVariablesOnly:

```
                    (not self.channelExpression.
                        oclIsUndefined())
and
(not self.channelExpression.variable.
    oclIsUndefined())
and
(not self.channelExpression.variable.
    typeDefinition.oclIsUndefined())
and
self.channelExpression.variable.typeDefinition.
    baseType = types::BuiltInType::CHAN
```

8.8 Enumeration `SynchronizationKind`

Overview Representing the type of synchronization.

Enum Properties Enumeration `SynchronizationKind` has the following literals:

RECEIVE = 0

SEND = 1

8.9 Class `Template`

Overview An UPPAAL template representing a single timed automaton.

Super Types

- AbstractTemplate see Section 8.1 on Page 32

Class References Class `Template` has the following references:

declarations : LocalDeclarations [0..1] see Section 3.15 on Page 9
The local declarations of the template.

edge : Edge [0..*] see Section 8.2 on Page 32
The edges inside this template.

init : Location see Section 8.3 on Page 33
The initial location of this template.

location : Location [1..*] see Section 8.3 on Page 33
The locations inside this template.

Class Constraints Class `Template` has the following constraints:

UniqueLocationNames:

```
self.location->isUnique(name)
```

9 Package uppaal::types

Package Overview Provides support for predefined and user-defined types.

9.1 Enumeration BuiltInType

Overview All built-in types.

Enum Properties Enumeration BuiltInType has the following literals:

INT = 0
CLOCK = 1
CHAN = 2
BOOL = 3
VOID = 4

9.2 Class DeclaredType

Overview A user-declared type.

Super Types

- Type see Section 9.8 on Page 39

Class References Class DeclaredType has the following references:

typeDeclaration : TypeDeclaration see Section 3.18 on Page 11
The declaration that declares this type.

/typeDefinition : TypeDefinition see Section 9.9 on Page 40
derivation:

```
                    if self.typeDeclaration.
                       oclIsUndefined()
  then null
  else self.typeDeclaration.typeDefinition
  endif
```

The definition of the declared type. Usually a type specification, but can also be a type reference to a "renamed" type.

9.3 Class IntegerBounds

Overview Used to restrict the 'int' type to a range of values.

Class References Class IntegerBounds has the following references:

lowerBound : Expression see Section 6.11 on Page 23
An integer-based expression representing the lower bound.

upperBound : Expression see Section 6.11 on Page 23
An integer-based expression representing the upper bound.

9.4 Class `PredefinedType`

Overview One of the predefined types 'int', 'bool', 'chan', 'clock' or 'void'.

Super Types

- Type see Section 9.8 on Page 39

Class Attributes Class `PredefinedType` has the following attributes:

type : BuiltInType see Section 9.1 on Page 37
Stores the concrete literal that represents the predefined type.

9.5 Class `RangeTypeSpecification`

Overview A type specification restricting the 'int' type to a range of values.

Super Types

- TypeSpecification see Section 9.11 on Page 41

Class References Class `RangeTypeSpecification` has the following references:

bounds : IntegerBounds see Section 9.3 on Page 37
The bounds that restrict the type specification.

9.6 Class `ScalarTypeSpecification`

Overview A specification of a 'scalar' type.

Super Types

- TypeSpecification see Section 9.11 on Page 41

Class References Class `ScalarTypeSpecification` has the following references:

sizeExpression : Expression see Section 6.11 on Page 23
An integer-based expression that represents the size of the scalar type.

9.7 Class `StructTypeSpecification`

Overview A specification of a 'struct' type.

Super Types

- TypeSpecification see Section 9.11 on Page 41

Class References Class `StructTypeSpecification` has the following references:

declaration : DataVariableDeclaration [1..*] see Section 3.5 on Page 5

The variable declarations representing the fields of the 'struct' type.

Class Constraints Class `StructTypeSpecification` has the following constraints:

UniqueFieldNames:

```
self.declaration->collect(
    variable)->isUnique(name)
```

9.8 Abstract Class `Type`

Overview Abstract base class for all types.

Super Types

- NamedElement see Section 2.2 on Page 3

Class Attributes Class `Type` has the following attributes:

/baseType : BuiltInType [0..1] see Section 9.1 on Page 37

derivation:

```
                    if self.oclIsKindOf(
                        DeclaredType)
  then
          if self.oclAsType(DeclaredType).
              typeDefinition.oclIsUndefined()
          then null
          else self.oclAsType(DeclaredType).
              typeDefinition.baseType
          endif
  else
          if self.oclIsKindOf(PredefinedType)
          then self.oclAsType(PredefinedType).
              type
          else null
          endif
  endif
```

Class References Class `Type` has the following references:

index : Index [0..*] see Section 3.13 on Page 9
A set of array indexes for the type.

9.9 Abstract Class `TypeDefinition`

Overview Abstract base class for type definitions of all typed elements. A type definition is either a references to an existing type, or a new type specified in place.

Class Attributes Class `TypeDefinition` has the following attributes:

/baseType : BuiltInType [0..1] see Section 9.1 on Page 37
derivation:

```
                    if self.oclIsKindOf(
                        TypeReference)
  then
            if self.oclAsType(TypeReference).
                referredType.oclIsUndefined()
            then null
            else self.oclAsType(TypeReference).
                referredType.baseType
            endif
  else
            if self.oclIsKindOf(
                ScalarTypeSpecification) or self.
                oclIsKindOf(RangeTypeSpecification)
            then BuiltInType::INT
            else null
            endif
  endif
```

The built-in base type this type definition relies on. Can be 'null' in case of a 'struct' type definition involved.

9.10 Class `TypeReference`

Overview A reference to an existing type. This could be a predefined type or a user-declared type.

Super Types

- TypeDefinition see Section 9.9 on Page 40

Class References Class `TypeReference` has the following references:

referredType : Type see Section 9.8 on Page 39
The referred type.

9.11 Abstract Class `TypeSpecification`

Overview Abstract base class for the specification of new types, using either the 'struct' or 'scalar' keywords, or restricting a type to a range of values.

Super Types

- TypeDefinition see Section 9.9 on Page 40

10 Package `uppaal::visuals`

Package Overview Provides support for the visual representation of model elements.

10.1 Abstract Class `ColoredElement`

Overview A model element that has an optional color.

Class Attributes Class `ColoredElement` has the following attributes:

colorCode : EString [0..1]
The hexadecimal color code of the model element.

10.2 Abstract Class `LinearElement`

Overview A linear model element that has a set of bend points.

Class References Class `LinearElement` has the following references:

bendPoint : Point [0..∗] see Section 10.4 on Page 42
The bend points of the linear model element.

10.3 Abstract Class `PlanarElement`

Overview A planar model element that has an optional position.

Class References Class `PlanarElement` has the following references:

position : Point [0..1] see Section 10.4 on Page 42
The planar position of the model element.

10.4 Class `Point`

Overview Represents a point in the two-dimensional space.

Class Attributes Class `Point` has the following attributes:

x : EInt
The horizontal component of the point.

y : EInt
The vertical component of the point.