Chi Metamodel Reference Documentation (Incubation)

Version 2023-05-05

Contents

Chapter 1

Introduction

Chi is a modeling language for describing and analyzing performance of discrete event systems by means of simulation. It uses a process-based view, and uses synchronous point-to-point communication between processes. A process is written as an imperative program, with a syntax much inspired by the well-known Python language.

Chi is one of the tools of the Eclipse ESCET™ project [2].

The Eclipse ESCET project, including the Chi language and toolset, is currently in the *Incubation Phase* [1].

This documentation is out of sync with respect to the meta model.

Add general introduction texts about Ecore models/diagrams, from position metamodel, similar to position and CIF metamodel documentation.

Chapter 2

EMF model constraints

In this chapter, the static constraints on a Chi ecore model are listed.

2.1 Addressable expressions

In the ecore model, attributes *AssignmentStatement.lhs* (Section 3.1.8) and *ReceiveStatement.data* (Section 3.1.66) should be addressable, they should refer to variables that can be assigned a value. In the ecore model however, these attributes have the generic *Expression* (Section 3.1.36) type. In this section, additional constraints are specified to consider an *Expression* (Section 3.1.36) object as being addressable.

The (recursive) constraints are:

1. If the expression is a *VariableReference* (Section 3.1.95), the expression is addressable.
2. If the expression is a *BinaryExpression* (Section 3.1.11), and its *BinaryExpression.operator* (Section 3.1.11) attribute is *BinaryOperators.Projection* (Section 3.1.4), and the left-hand side (*BinaryExpression.left* (Section 3.1.11)) is a *VariableReference* (Section 3.1.95) or a *BinaryExpression* (Section 3.1.11), and the left-hand side is addressable, the expression is addressable.
3. If the expression is a *TupleExpression* (Section 3.1.83), and all of its fields are addressable, the expression is addressable.

Chapter 3

Chi metamodel

3.1 Package chi

Classes and attributes description of the EMF model of the discrete event Chi simulation language.

Package URI http://eclipse.org/escet/chi

Namespace prefix chi

Sub-packages none

3.1.1 ChiIdentifier (datatype)

Not yet described.

Name

Instance class name java.lang.String

Basetype

Pattern

3.1.2 ChiNumber (datatype)

Unsigned number literal value.

Name ChiNumber

Instance class name java.lang.String

Basetype string

Pattern `0|([1-9][0-9]*)`

3.1.3 ChiRealNumber (datatype)

Non-negative real number literal value.

Name ChiRealNumber

Instance class name java.lang.String

Basetype string

Pattern `(0|([1-9][0-9]*))((\.[0-9]+)|((\.[0-9]+)?[eE][\-\+]?[0-9]+))`

3.1.4 BinaryOperators (enumeration)

Available binary operators in the expressions. Type constraints are listed in the *BinaryExpression* (Section 3.1.11) class documentation.

literal **Addition** (default)
: Addition ($a + b$).

literal **Conjunction**
: Short circuit disjunction ($a \vee b$).

  Implementations must guarantee short circuit evaluation of this binary operator. Note that when manipulating expressions, the operands may only be swapped if the resulting expression evaluates to the same value as the original expression did, when using short circuit evaluation semantics for both the original and the resulting expression.

literal **Disjunction**
: Short circuit disjunction ($a \vee b$).

  Implementations must guarantee short circuit evaluation of this binary operator. Note that when manipulating expressions, the operands may only be swapped if the resulting expression evaluates to the same value as the original expression did, when using short circuit evaluation semantics for both the original and the resulting expression.

literal **Division**
: (Real) division (a/b).

  Note that division by zero results in a run-time error.

literal **ElementTest**
: Element test on lists, sets, and dictionaries ($a \in b$).

literal **Equal**
: Equality test ($a = b$).

literal **FloorDivision**
: Floor division ($a \div b \equiv \lfloor a/b \rfloor$). Note that floor division is not the same as truncated division, and neither is it round-to-nearest integer division.

a	b	$a \div b$	$a \bmod b$
7	4	1	3
7	-4	-2	-1
-7	4	-2	1
-7	-4	1	-3

Note that division by zero results in a run-time error.

literal **GreaterEqual**
Greater or equal ($a \geq b$).

literal **GreaterThan**
Greater than ($a > b$).

literal **LessEqual**
Less or equal ($a \leq b$).

literal **LessThan**
Less than ($a < b$).

literal **Maximum**
Maximum operator ($a \max b$).

literal **Minimum**
Minimum operator ($a \min b$).

literal **Modulus**
Modulus operator ($a \bmod b \equiv a - b \cdot (a \div b)$).

Note that the following relation holds: $a = b \cdot (a \div b) + (a \bmod b)$. For examples, see the *BinaryOperators.FloorDivision* (Section 3.1.4) operator.

Note that it is considered a run-time error if the second operand evaluates to zero.

literal **Multiplication**
Multiplication ($a \cdot b$).

literal **NotEqual**
Not-equal ($a \neq b$).

literal **Power**
Power operator (a^b).

Note that it is considered a run-time error if one of the following conditions holds during evaluation:

- the base (a) is zero, and the exponent (b) is negative
- the base (a) is negative, and the exponent (b) is a non-integer number

literal **Projection**
Projection operation, extracts a value from a container value. Meaning and precise semantics depend on the type of the container, see the *BinaryExpression* (Section 3.1.11) class description for details.

literal **Subset**
Subset test ($a \subseteq b$).

literal **Subtraction**
Subtraction operator ($a - b$).

3.1.5 ChannelOps (enumeration)

Operations that may be performed on a channel data type.

literal **Receive** (default)
Only the receiving operation of a channel is allowed.

literal **Send**
Only the sending operation of a channel is allowed.

literal **SendReceive**
Both the sending and the receiving operations are allowed. This value means that the user explicitly stated allowance of both operations.

3.1.6 StdLibFunctions (enumeration)

Available standard library functions. Parameter and return types are listed in the *StdLibFunctionReference* (Section 3.1.77) class documentation.

Add a 'channel' function.

literal **Abs** (default)
Absolute value function.

literal **Acos**
Arc cosine function.

Note that it is considered a run-time error if evaluation of the absolute value of the argument evaluates to a number larger than one.

literal **Acosh**
Inverse hyperbolic cosine function.

Note that it is considered a run-time error if evaluation of the value of the argument evaluates to a number less than one.

literal **Asin**
Arc sine function.

Note that it is considered a run-time error if evaluation of the absolute value of the argument evaluates to a number larger than one.

literal **Asinh**
Inverse hyperbolic sine function.

literal **Atan**
Arc tangent function.

literal **Atanh**
Inverse hyperbolic tangent function.

Note that it is considered a run-time error if evaluation of the absolute value of the argument evaluates to a number greater than or equal to one.

literal **Bernoulli**
Bernoulli distribution function.

literal **Beta**
Beta distribution function.

literal **Binomial**
Binomial distribution function.

literal **Bool2String**
Convert boolean value to text.

literal **Cbrt**
Cubic root function.

literal **Ceil**
Round up (towards ∞). In other words, it results in the smallest integer value that is not less than the argument.

literal **Close**
Close a file.

literal **Constant**
Constant distribution function (useful for debugging).

literal **Cos**
Cosine function.

literal **Cosh**
Hyperbolic cosine function.

literal **DictKeys**
Retrieve the keys of a dictionary.

literal **DictValues**
Retrieve the values of a dictionary.

literal **DrawBernoulli**
Compute a sample according to a Bernoulli distribution.

literal **DrawBeta**
Compute a sample according to a Beta distribution.

literal **DrawBinomial**
Compute a sample according to a Binomial distribution.

literal **DrawErlang**
Compute a sample according to a Erlang distribution.

literal **DrawExponential**
Compute a sample according to a Exponential distribution.

literal **DrawGamma**
Compute a sample according to a Gamma distribution.

literal **DrawGeometric**
Compute a sample according to a Geometric distribution.

literal **DrawLogNormal**
Compute a sample according to a LogNormal distribution.

literal **DrawNormal**
Compute a sample according to a Normal distribution.

literal **DrawPoisson**
Compute a sample according to a Poisson distribution.

literal **DrawRandom**
Compute a sample according to a Random distribution.

literal **DrawTriangle**
Compute a sample according to a Triangle distribution.

literal **DrawUniform**
Compute a sample according to a Uniform distribution.

literal **DrawWeibull**
Compute a sample according to a Weibull distribution.

literal **Empty**
Tests whether its container argument is a empty. (Works for list, set, dictionary, and string.)

literal **Enumerate**
Return a list of pairs, where the first value is an index number and the second value is a value from its container argument.

literal **Erlang**
Erlang distribution function.

literal **Exp**
Exponential function.

literal **Exponential**
Exponential distribution function.

literal **Finished**
Returns whether the process instance has finished already.

literal **Floor**
Round down (towards $-\infty$). In other words, it results in the largest integer value that does not exceed the argument.

literal **Gamma**
Gamma distribution function.

literal **Geometric**
: Geometric distribution function.

literal **Insert**
: Insert a value in a sorted list.

literal **Int2Real**
: Convert integer number to real number.

literal **Int2String**
: Convert integer number to text.

literal **Length**
: Length of list, set, dictionary, or string.

literal **Ln**
: Natural logarithmic function.

  Note that it is considered a run-time error if evaluation of the argument results in a non-positive number.

literal **Log**
: Logarithmic (base 10) function.

  Note that it is considered a run-time error if evaluation of the argument results in a non-positive number.

literal **LogNormal**
: LogNormal distribution function.

literal **Matrix**
: Construct a matrix from a list.

  Is this a stdlib function?

literal **Max**
: Take the maximum value of a list, dictionary, or set.

literal **Min**
: Take the minimum value of a list, dictionary, or set.

literal **Normal**
: Normal distribution function.

literal **Open**
: Open a file.

literal **Poisson**
: Poisson distribution function.

literal **Pop**
: Extract a value from a container.

literal **Random**
: Core random (uniform from $[0, 1)$) distribution function.

literal **Range**
Construct a list with numbers in the given range.

literal **Real2String**
Convert real number to text.

literal **Round**
Round to nearest integer value. If the value is exactly between two integer values, it is rounded up (towards ∞). The rounding of an argument r may be computed using the following expression: $\lfloor x + 0.5 \rfloor$.

literal **SampleFunc**
Compute a sample of a distribution.

literal **SetSeed**
Set seed of a distribution.

literal **Sign**
Sign function.

literal **Sin**
Sine function.

literal **Sinh**
Hyperbolic sine function.

literal **Sort**
Sort a list.

literal **Sqrt**
Square root function.

Note that it is considered a run-time error if evaluation of the argument results in a negative number.

literal **String2Bool**
Convert string containing textual boolean value to boolean.

Note that it is considered a run-time error if the argument is not an ASCII representation of a CIF boolean value (*true* or *false*).

literal **String2Int**
Convert textual signed integer number to integer number.

Note that it is considered a run-time error if the argument is not an ASCII representation of a CIF natural number (*ChiNumber* (Section 3.1.2)), optionally prefixed with the ASCII representation of a negation (*UnaryOperators.Negate* (Section 3.1.7)).

literal **String2Real**
Convert textual real value to a real number.

Note that it is considered a run-time error if the argument is not an ASCII representation of a CIF real number (*ChiRealNumber* (Section 3.1.3)), optionally prefixed with the ASCII representation of a negation (*UnaryOperators.Negate* (Section 3.1.7)).

literal **Tan**
Tangent function.

literal **Tanh**
: Hyperbolic tangent function.

literal **Timeout**
: Test whether its timer argument has timed out.

Add/replace with timer_ready(timer)→ bool, and time_left(timer)→real

literal **Triangle**
: Triangle distribution function.

literal **Uniform**
: Uniform distribution function.

literal **Weibull**
: Weibull distribution function.

3.1.7 UnaryOperators (enumeration)

Expression operator with one child expression.

literal **Inverse** (default)
: (Boolean) inverse operator ($\neg a$).

literal **Negate**
: Negate operator ($-a$).

literal **Plus**
: Unary plus operator (mostly for completeness only.

literal **Sample**
: Sample operator, draws a sample from a stochastic distribution.

Decide where it may be used safely.

3.1.8 AssignmentStatement (class)

Assigns values to variables.

- **AssignmentStatement.type** The type of the left-hand side and the type of the right-hand side must be equal. Sequences of values are interpreted as record value.
- **AssignmentStatement.count** The number of addressed variables at the left-hand side must either be 1 (in which case the values at the right-hand side are packed in a tuple), it must be equal to the number of values at the right-hand side (in which case, a number of one-to-one assignments are performed simultaneously), or the number of values at the right-hand side must be 1 (in which case the right-hand side is unpacked to the variables at the left-hand side).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement* (Section 3.1.76)
└ *AssignmentStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **lhs** [1] : *Expression*
Left-hand side of the assignment statement. The expression list must be addressable. See Section 2.1 for the rules of addressable expressions.

cont **rhs** [1] : *Expression*
Right-hand side of the assignment statement. A list of expressions is interpreted as a record of values.

3.1.9 BaseFunctionReference (abstract class)

Base class for function references.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *BaseFunctionReference*

Direct derived classes: *FunctionReference* (Section 3.1.41), *StdLibFunctionReference* (Section 3.1.77)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

3.1.10 BehaviourDeclaration (abstract class)

Declaration with behaviour (a model, process, or function definition).

EObject

└ *ChiObject* (Section 3.1.20)
└ *Declaration* (Section 3.1.26)
└ *BehaviourDeclaration*

Direct derived classes: *FunctionDeclaration* (Section 3.1.40), *ModelDeclaration* (Section 3.1.54), *ProcessDeclaration* (Section 3.1.59)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **statements** [1..*] : *Statement*
Body of statements of the definition.

cont **variables** [0..*] : *VariableDeclaration*
Formal parameters and local variables of the definition. Both kinds of variables are read/write, as parameters are always call-by-value.

3.1.11 BinaryExpression (class)

Binary operator in an expression.

- **BinaryExpression.type** The allowed types of the left hand side, the right hand side, and the result depend on the operator. The tables below list them for each possible operator.

Addition operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
ListType (Section 3.1.50)	*ListType* (Section 3.1.50)	*ListType* (Section 3.1.50)
TupleType (Section 3.1.85)	*TupleType* (Section 3.1.85)	*TupleType* (Section 3.1.85)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*StringType* (Section 3.1.79)
SetType (Section 3.1.73)	*SetType* (Section 3.1.73)	*SetType* (Section 3.1.73)

For the lists, the element types of the left side, of the right side, and of the result are all the same.

For records, the element types of the result are the element types of the left side followed by the element types of the right side, with all field names removed.

For set union, the element types of all the set types must be the same. For union over dictionaries, the key types must all be the same and the value type must all be the same.

Conjunction operator

Left type	Right type	Result type
BoolType (Section 3.1.14)	*BoolType* (Section 3.1.14)	*BoolType* (Section 3.1.14)

Disjunction operator

Left type	Right type	Result type
BoolType (Section 3.1.14)	*BoolType* (Section 3.1.14)	*BoolType* (Section 3.1.14)

Division operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)

ElementTest operator

Left type	Right type	Result type
t	*ListType* (Section 3.1.50)	*BoolType* (Section 3.1.14)
t	*SetType* (Section 3.1.73)	*BoolType* (Section 3.1.14)
t	*DictType* (Section 3.1.28)	*BoolType* (Section 3.1.14)

At the left side, any static type t may be used. At the right side, for lists and sets, the element type must be the same type t. For element test on dictionaries, the *DictType.keyType* (Section 3.1.28) must be the same type t.

Equal operator

Left type	Right type	Result type
t	t	*BoolType* (Section 3.1.14)

Any two values with same type t can be compared with each other for equality.

FloorDivision operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)

GreaterEqual operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*BoolType* (Section 3.1.14)

GreaterThan operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*BoolType* (Section 3.1.14)

LessEqual operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*BoolType* (Section 3.1.14)

LessThan operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*BoolType* (Section 3.1.14)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*BoolType* (Section 3.1.14)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*BoolType* (Section 3.1.14)

Maximum operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*StringType* (Section 3.1.79)

Minimum operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
StringType (Section 3.1.79)	*StringType* (Section 3.1.79)	*StringType* (Section 3.1.79)

Modulus operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)

Multiplication operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
SetType (Section 3.1.73)	*SetType* (Section 3.1.73)	*SetType* (Section 3.1.73)
DictType (Section 3.1.28)	*DictType* (Section 3.1.28)	*DictType* (Section 3.1.28)

NotEqual operator

Left type	Right type	Result type
t	t	*BoolType* (Section 3.1.14)

Any two values with same type t can be compared with each other for unequality.

Power operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)

Projection operator

Left type	Right type	Right class	Result type
ListType (Section 3.1.50)	*IntType* (Section 3.1.46)	-	t_l
DictType (Section 3.1.28)	t_k	-	t_v
TupleType (Section 3.1.85)	-	*FieldReference* (Section 3.1.37)	t_i
TupleType (Section 3.1.85)	*IntType* (Section 3.1.46)	-	t_i

For projection of lists, the result type t_l is the same as the element type of the list. For dictionaries, the right hand side must have the same type t_k as the key type of the dictionary expression, and the result type t_v is the same type as the value type of the dictionary expression. For a record type, there are two cases. The first case is where the right hand side is a *FieldReference* (Section 3.1.37). In such a case, the referred field must match a field in the record type, and the result type of the projection is the type of the field addressed. The second case is where the right hand side has a *IntType* (Section 3.1.46). In such a case, the result type of the projection is the type of the field referred to by the right hand side expression, which is the zero-based projection index.

Subset operator

Left type	Right type	Result type
SetType (Section 3.1.73)	*SetType* (Section 3.1.73)	*BoolType* (Section 3.1.14)
DictType (Section 3.1.28)	*DictType* (Section 3.1.28)	*BoolType* (Section 3.1.14)

For the subset over sets, the element types of both set types must be the same. For subset over dictionaries, both key types must be the same and both value types must be the same.

Subtraction operator

Left type	Right type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
IntType (Section 3.1.46)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
ListType (Section 3.1.50)	*ListType* (Section 3.1.50)	*ListType* (Section 3.1.50)
SetType (Section 3.1.73)	*SetType* (Section 3.1.73)	*SetType* (Section 3.1.73)
DictType (Section 3.1.28)	*DictType* (Section 3.1.28)	*DictType* (Section 3.1.28)

For subtraction over sets and lists, the element types of all the set types must be the same. For subtraction over dictionaries, the key types must all be the same and the value type must all be the same.

EObject
- *ChiObject* (Section 3.1.20)
- *Expression* (Section 3.1.36)
- *BinaryExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **left** [1] : *Expression*
Left-hand sub-expression of the binary expression.

cont **operator** [1] : *BinaryOp*
Operator of the binary expression.

cont **right** [1] : *Expression*
Right-hand sub-expression of the binary expression.

3.1.12 BinaryOp (class)

Extra class for attaching position information to a binary operator.

EObject
└ *ChiObject* (Section 3.1.20)
└ *BinaryOp*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **op** [1] : *BinaryOperators*
Binary operator value.

3.1.13 BoolLiteral (class)

Boolean literal value.

- **BoolLiteral.type** The type of the boolean literal is a *BoolType* (Section 3.1.14).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *BoolLiteral*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

attr **value** [1] : *EBoolean*
Value of the boolean literal.

3.1.14 BoolType (class)

Type denoting a boolean value.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *BoolType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.15 BreakStatement (class)

The break statement aborts execution of the inner loop. Execution continues with the statement directly following the loop.

- **BreakStatement.usage** A break statement may only be used inside a for loop (*ForStatement.body* (Section 3.1.39)) or while loop (*WhileStatement.body* (Section 3.1.97)).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement* (Section 3.1.76)
└ *BreakStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.16 CallExpression (class)

Expression denoting application of a function definition *FunctionDeclaration* (Section 3.1.40) or instantiation of a process definition *ProcessDeclaration* (Section 3.1.59).

EObject
- *ChiObject* (Section 3.1.20)
- *Expression* (Section 3.1.36)
- *CallExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **arguments** [0..*] : *Expression*
Expressions (one for each formal parameter of the function declaration or the process declaration) denoting the values of the parameters of the application.

- **CallExpression.parameterCount** The number of argument expressions must be equal to the number of formal parameters of the function or process type of the *CallExpression.function* (Section 3.1.16) attribute.
- **CallExpression.parameterTypes** The type of each argument expression must be equal to the type of its corresponding formal parameter of the function type or process type of the *CallExpression.function* (Section 3.1.16) attribute.

cont **function** [1] : *Expression*
Expression denoting the function to call or the process to instantiate. Often this is a *FunctionReference* (Section 3.1.41) or a *ProcessReference* (Section 3.1.61), but other forms are allowed too, for example a variable.

- **CallExpression.funcType** The type of the function expression must be a *FunctionType* (Section 3.1.42) or a *ProcessType* (Section 3.1.62).

cont **name** [0..1] : *Expression*
Optional expression expressing the name of the process instance.

- **CallExpression.nullName** The name expression (*CallExpression.name* (Section 3.1.16)) must be `null` if *CallExpression.type* (Section 3.1.16) is a function (*FunctionType* (Section 3.1.42).
- **CallExpression.nameType** If the attribute *CallExpression.name* (Section 3.1.16) is not `null`, it must be an expression of type *StringType* (Section 3.1.79) or of type *IntType* (Section 3.1.46).

3.1.17 ChannelExpression (class)

Not yet described.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Expression* (Section 3.1.36)
 ⌞ *ChannelExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **elementType** [1] : *Type*
Not yet described.

3.1.18 ChannelOp (class)

Extra class to attach a position to the channel operations.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *ChannelOp*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **ops** [1] : *ChannelOps*
Channel operations.

3.1.19 ChannelType (class)

Data type denoting a channel.

EObject

└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *ChannelType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **elementType** [1] : *Type*
Data type of data communicated over the channel. The *VoidType* (Section 3.1.96) means that no data is communicated (these are called 'synchronization channels').

cont **operations** [0..1] : *ChannelOp*
Allowed operations on the channel value.

3.1.20 ChiObject (class)

Base class for all classes with an optional position.

EObject
└ *ChiObject*

Direct derived classes: *BinaryOp* (Section 3.1.12), *ChannelOp* (Section 3.1.18), *CreateCase* (Section 3.1.25), *Declaration* (Section 3.1.26), *DictionaryPair* (Section 3.1.30), *Expression* (Section 3.1.36), *IfCase* (Section 3.1.43), *MatrixRow* (Section 3.1.52), *Name* (Section 3.1.55), *SelectCase* (Section 3.1.69), *Statement* (Section 3.1.76), *TupleField* (Section 3.1.84), *Type* (Section 3.1.86), *UnaryOp* (Section 3.1.90), *Unwind* (Section 3.1.93), *VariableDeclaration* (Section 3.1.94)

cont **position** [0..1] : *Position*
Position of the construct in the source file.

3.1.21 CommunicationStatement (abstract class)

Base class for communication (send and receive) actions.

- **CommunicationStatement.notInFunction** The communication statement may not be used in a function body (*FunctionDeclaration.statements* (Section 3.1.40)).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement* (Section 3.1.76)
└ *CommunicationStatement*

Direct derived classes: *ReceiveStatement* (Section 3.1.66), *SendStatement* (Section 3.1.71)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **channel** [1] : *Expression*
Expression evaluating to the channel communicated on.

- **CommunicationStatement.channelType** The type of the *CommunicationStatement.channel* (Section 3.1.21) expression must be a channel type (*ChannelType* (Section 3.1.19)).

cont **data** [0..*] : *Expression*
For *ReceiveStatement* (Section 3.1.66), addressables that store the communicated data if the channel is not a synchronization channel (See Section 2.1 for the constraints on addressable expressions). For *SendStatement* (Section 3.1.71), the expression evaluating to the value communicated over the channel (if not a synchronization channel).

- **CommunicationStatement.synchronization** If the channel element type is *VoidType* (Section 3.1.96), the data part must be empty.

3.1.22 ConstantDeclaration (class)

Declaration of a name for a constant value.

- **ConstantDeclaration.type** Type of the constant declaration must be equal to the type of the value.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Declaration* (Section 3.1.26)
└ *ConstantDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **type** [1] : *Type*
Data type of the value.

cont **value** [1] : *Expression*
Expression denoting the value.

- **ConstantDeclaration.constant** Value of the expression may not change, it may only refer to literals, operators, and other constant values.
- **ConstantDeclaration.nocycle** A constant value may not (indirectly) depend on itself.

3.1.23 ConstantReference (class)

Reference of a constant by its name in an expression.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *ConstantReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

ref **constant** [1] : *ConstantDeclaration*
Referenced *ConstantDeclaration* (Section 3.1.22) in the expression.

- **ConstantReference.scope** The name of the referenced constant should refer to the same declaration in the current scope.

3.1.24 ContinueStatement (class)

Abort execution of the body of the inner loop, and continue with the next iteration. For *WhileStatement* (Section 3.1.97) loops, execution continues with evaluation of the while condition. For *ForStatement* (Section 3.1.39) loops, execution continues with computing the next value of the iteration variables.

- **ContinueStatement.usage** A continue statement may only be used inside a for loop (*ForStatement.body* (Section 3.1.39)) or while loop (*WhileStatement.body* (Section 3.1.97)).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement* (Section 3.1.76)
└ *ContinueStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.25 CreateCase (abstract class)

One of the cases in a *run* or *start* statement.

EObject
└ *ChiObject* (Section 3.1.20)
└ *CreateCase*

Direct derived classes: *IteratedCreateCase* (Section 3.1.47), *ProcessInstance* (Section 3.1.60)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.26 Declaration (abstract class)

Declaration at the global level of a Chi specification.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Declaration*

Direct derived classes: *BehaviourDeclaration* (Section 3.1.10), *ConstantDeclaration* (Section 3.1.22), *EnumDeclaration* (Section 3.1.32), *TypeDeclaration* (Section 3.1.87)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name*
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

3.1.27 DelayStatement (class)

Statement to pass some time.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Statement* (Section 3.1.76)
 ⌞ *DelayStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **length** [1] : *Expression*
Expression denoting the length of the delay. Should be non-negative at run-time.

- **DelayStatement.type** Type of the length expression must be a *RealType* (Section 3.1.65) or an *IntType* (Section 3.1.46).

3.1.28 DictType (class)

Data type denoting a dictionary.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Type* (Section 3.1.86)
 ⌞ *DictType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **keyType** [1] : *Type*
Type of the keys of the dictionary type.

- **DictType.keyType** Type of the keys may not be *VoidType* (Section 3.1.96).

cont **valueType** [1] : *Type*
Type of the values of the dictionary type.

- **DictType.valueType** Type of the values may not be *VoidType* (Section 3.1.96).

3.1.29 DictionaryExpression (class)

Expression denoting a value with a dictionary type.

- **DictionaryExpression.type** Type of the expression must be *DictionaryExpression* (Section 3.1.29).
- **DictionaryExpression.keyType** Type of the key-part of each dictionary-pair must be the same as the *DictType.keyType* (Section 3.1.28) type.
- **DictionaryExpression.valueType** Type of the value-part of each dictionary-pair must be the same as the *DictType.valueType* (Section 3.1.28) type.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *DictionaryExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **pairs** [0..*] : *DictionaryPair*
Key/value pairs of the dictionary expression.

- **DictType.keyValues** Value of each pair must be unique.

3.1.30 DictionaryPair (class)

Key/value pair of a dictionary value.

EObject
└ *ChiObject* (Section 3.1.20)
└ *DictionaryPair*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **key** [1] : *Expression*
Expression denoting the key of the pair.

cont **value** [1] : *Expression*
Expression denoting the value of the pair.

3.1.31 DistributionType (class)

Data type denoting a distribution.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *DistributionType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **resultType** [1] : *Type*
Data type of sampled values of the distribution.

3.1.32 EnumDeclaration (class)

Not yet described.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Declaration* (Section 3.1.26)
└ *EnumDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **values** [1..*] : *EnumValue*
Not yet described.

3.1.33 EnumTypeReference (class)

Not yet described.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Type* (Section 3.1.86)
 ⌞ *EnumTypeReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

ref **type** [1] : *EnumDeclaration*
Not yet described.

3.1.34 EnumValue (class)

A value in an enumeration type.

- **EnumValue.unique** The name of each value must be unique globally, to ensure a proper mapping of its values back to the correct enum type.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Name* (Section 3.1.55)
 ⌞ *EnumValue*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **name** [1] : *ChiIdentifier* (inherited from *Name*)
Name contained by the class.

3.1.35 EnumValueReference (class)

Reference to the name of a enum type.

- **EnumValueReference.type** The data type of the reference is equal to the enum type to which the referenced enum value belongs.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Expression* (Section 3.1.36)
 ⌞ *EnumValueReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

ref **value** [1] : *EnumValue*
Reference to the referenced enum value.

3.1.36 Expression (abstract class)

Object denoting a value.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Expression*

Direct derived classes: *BaseFunctionReference* (Section 3.1.9), *BinaryExpression* (Section 3.1.11), *BoolLiteral* (Section 3.1.13), *CallExpression* (Section 3.1.16), *ChannelExpression* (Section 3.1.17), *ConstantReference* (Section 3.1.23), *DictionaryExpression* (Section 3.1.29), *EnumValueReference* (Section 3.1.35), *FieldReference* (Section 3.1.37), *ListExpression* (Section 3.1.49), *MatrixExpression* (Section 3.1.51), *Number* (Section 3.1.56), *ProcessReference* (Section 3.1.61), *ReadCallExpression* (Section 3.1.63), *RealNumber* (Section 3.1.64), *SetExpression* (Section 3.1.72), *SliceExpression* (Section 3.1.74), *StringLiteral* (Section 3.1.78), *TimeLiteral* (Section 3.1.81), *TupleExpression* (Section 3.1.83), *UnaryExpression* (Section 3.1.89), *UnresolvedReference* (Section 3.1.91), *VariableReference* (Section 3.1.95)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type*
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

3.1.37 FieldReference (class)

Reference expression the field of a record.

- **FieldReference.type** Type of the field reference is equal to the type of the referenced field.
- **FieldReference.visible** The field must be reachable from the current scope.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *FieldReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

ref **field** [1] : *TupleField*
Referenced field.

3.1.38 FileType (class)

Data type of a stream of data from or to the operating system.

Note: There is no fixed type associated with the data at the stream. Also, a stream may be opened for reading, for writing, or both. Trying to use a stream in the 'wrong' way results in undefined behaviour.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *FileType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.39 ForStatement (class)

Ierative loop statement. Values from the source are iteratively assigned to variables of the statement, and the body is executed for each assignment.

Note that the body may contain *BreakStatement* (Section 3.1.15) or *ContinueStatement* (Section 3.1.24) objects, which cause partial execution of the body. Also, execution of a *ReturnStatement* (Section 3.1.67) causes termination of the body as well as the loop.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Statement* (Section 3.1.76)
 ⌞ *ForStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **body** [1..*] : *Statement*
Statements executed after each assignment of values to the variables.

cont **source** [1] : *Expression*
Expression denoting a sequence of values to assign to the variables.

- **ForStatement.sourceType** The type of the source expression must be a *ListType* (Section 3.1.50), *SetType* (Section 3.1.73), or a *DictType* (Section 3.1.28).

cont **variables** [1..*] : *VariableDeclaration*
Sequence of variables to assign at the start of each iteration. For the purpose of type correctness, the variable list may be considered to be a list of identifier expressions where Section 2.1 must be applied to.

- **ForStatement.variablesType** If the type of the source expression (*ForStatement.source* (Section 3.1.39)) is a *ListType* (Section 3.1.50) or *SetType* (Section 3.1.73), the type of the iterated variable must be the element type (*ListType.elementType* (Section 3.1.50) or *SetType.elementType* (Section 3.1.73)). If the type of the source expression is a dictionary (*DictType* (Section 3.1.28)), the type of the variables is a record with the key type and the value type of the dictionary.
- **ForStatement.uniqueVariables** Names of variables of the statement must be unique to each other as well as in the scope.

3.1.40 FunctionDeclaration (class)

Definition of a user-defined function.

- **FunctionDeclaration.bodyReturn** All exit points in the statements of a function declaration must end with a *ReturnStatement* (Section 3.1.67).

- **FunctionDeclaration.dataOnly** Formal parameters and local variables must not contain timers.
- **FunctionDeclaration.noTimeAccess** Expressions in the statements of a a function definition may not use *TimeLiteral* (Section 3.1.81).
- **FunctionDeclaration.noCreate** *FunctionDeclaration.statements* (Section 3.1.40) may not have objects of the *RunStatement* (Section 3.1.68) class.
- **FunctionDeclaration.noDelay** *FunctionDeclaration.statements* (Section 3.1.40) may not have objects of the *DelayStatement* (Section 3.1.27) class.
- **FunctionDeclaration.noCommunication** The communication statement may not be used in a function body (*FunctionDeclaration.statements* (Section 3.1.40)).

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Declaration* (Section 3.1.26)
 ∟ *BehaviourDeclaration* (Section 3.1.10)
 ∟ *FunctionDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **statements** [1..*] : *Statement* (inherited from *BehaviourDeclaration*)
Body of statements of the definition.

cont **variables** [0..*] : *VariableDeclaration* (inherited from *BehaviourDeclaration*)
Formal parameters and local variables of the definition. Both kinds of variables are read/write, as parameters are always call-by-value.

cont **returnType** [1] : *Type*
Type of the value returned by the function.

- **FunctionDeclaration.typeOfReturnValue** The type of the returned value is not *VoidType* (Section 3.1.96).

3.1.41 FunctionReference (class)

Reference to a user-defined function.

- **FunctionReference.type** The type of a function reference is a *FunctionType* (Section 3.1.42), where its return type (*FunctionType.resultType* (Section 3.1.42)) is equal to the return type of the referenced function, and the list of parameter types (*FunctionType.parameterTypes* (Section 3.1.42)) must match with the types of the formal parameters of the referenced function.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Expression* (Section 3.1.36)
 ⌞ *BaseFunctionReference* (Section 3.1.9)
 ⌞ *FunctionReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

ref **function** [1] : *FunctionDeclaration*
Referenced function.

- **FunctionReference.inScope** The referenced function must be visible in the current scope.

3.1.42 FunctionType (class)

Data type of a function.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Type* (Section 3.1.86)
 ⌞ *ProcessType* (Section 3.1.62)
 ⌞ *FunctionType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **parameterTypes** [0..*] : *Type* (inherited from *ProcessType*)
Data types of the formal parameters.

cont **resultType** [1] : *Type*
Type of the result value of an application of a function with this function signature.

- **FunctionType.resultType** The result type a function does not hjave type *VoidType* (Section 3.1.96).

3.1.43 IfCase (class)

One case to try in an if statement.

EObject
└ *ChiObject* (Section 3.1.20)
└ *IfCase*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **body** [1..*] : *Statement*
Statements to execute if the condition is empty or holds.

cont **condition** [0..1] : *Expression*
Expression that decides whether or not to execute the body of the object.

- **IfCase.boolCondition** If the condition is not empty (`null`), it must have type *BoolType* (Section 3.1.14).

3.1.44 IfStatement (class)

Selection statement.

Execution of the statement means sequentially testing whether the *IfCase.condition* (Section 3.1.43) holds. If it does not hold, the next case is tried. If the condition is absent or it does hold, the body associated with the condition is executed. Then the if statement terminates (the remaining *IfCase* (Section 3.1.43) objects are not tested nor executed).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement* (Section 3.1.76)
└ *IfStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **cases** [1..*] : *IfCase*
Sequence of if cases to test and potentially execute.

- **IfStatement.lastCase** The *IfCase.condition* (Section 3.1.43) may not be empty, except for the last case. (This is then considered to be an 'else' branch.)

3.1.45 InstanceType (class)

Data type of an instantiated (running) process.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *InstanceType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.46 IntType (class)

Data type of an integer number.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *IntType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.47 IteratedCreateCase (class)

A case in a *run* or *start* statement that must be expanded at run-time to a number of process instances.

EObject
└ *ChiObject* (Section 3.1.20)

└ *CreateCase* (Section 3.1.25)
└ *IteratedCreateCase*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **instances** [1..*] : *CreateCase*
Parameterized process instance to instantiate on each iteration.

cont **unwinds** [1..*] : *Unwind*
Sequence of loops to expand for the create case.

3.1.48 IteratedSelectCase (class)

Case in a *select* statement that must be expanded at tun-time to a number of conditions to wait on.

EObject
└ *ChiObject* (Section 3.1.20)
└ *SelectCase* (Section 3.1.69)
└ *IteratedSelectCase*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **body** [1..*] : *Statement* (inherited from *SelectCase*)
Sequence of statements to execute if the select case is chosen.

cont **guard** [0..1] : *Expression* (inherited from *SelectCase*)
Optional guard expression that should hold for the case to be chosen.

- **SelectCase.guardType** If the guard is present (i.e. not `null`), the type of the guard expression should be *BoolType* (Section 3.1.14).

cont **unwinds** [1..*] : *Unwind*
Sequence of loops to expand for the select case.

3.1.49 ListExpression (class)

Expression denoting a list value.

- **ListExpression.type** The data type of a list expression is a *ListType* (Section 3.1.50).

- **ListExpression.elements** The element type of its list data type (*ListType.elementType* (Section 3.1.50)) must be the same as the type of each of the element value expressions.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *ListExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **elements** [0..*] : *Expression*
Ordered list of element values.

3.1.50 ListType (class)

Data type of a list value.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *ListType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **elementType** [1] : *Type*
Data type of the elements of a list value.

ref **initialLength** [0..1] : *Expression*
Length of the list at initialization.

- **ListType.nonZeroInitialLength** The *ListType.initialLength* (Section 3.1.50) can only be non-zero for for element data types that have an initial value.

Define which data types have an initial value.

Define when a list can have an non-zero initial length.

3.1.51 MatrixExpression (class)

Expression denoting a matrix literal (with operators '+' and '·' over real values).

- **MatrixExpression.Rowlength** Each row in *MatrixExpression.rows* (Section 3.1.51) must have the same number of elements.
- **MatrixExpression.type** The type of the matrix expression must be a *MatrixType* (Section 3.1.53) where the number of rows (*MatrixType.rowSize* (Section 3.1.53)) matches with the number of values in *MatrixExpression.rows* (Section 3.1.51), and the number of columns (*MatrixType.columnSize* (Section 3.1.53)) matches with the length of each row,

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *MatrixExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **rows** [1..*] : *MatrixRow*
Class for storing a single row in the matrix.

3.1.52 MatrixRow (class)

A row of elements in a matrix literal.

EObject
└ *ChiObject* (Section 3.1.20)
└ *MatrixRow*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **elements** [1..*] : *Expression*
A expression denoting a single value in a matrix.

- **MatrixRow.type** The type of each element must be *RealType* (Section 3.1.65).

3.1.53 MatrixType (class)

The type of a matrix.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *MatrixType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **columnSize** [1] : *Expression*
Number of columns in the matrix value.

- **MatrixType.columnDimensionIsInt** Type of the column dimension expression must be *IntType* (Section 3.1.46).
- **MatrixType.columnDimensionValue** Value must be a constant (and at least one, as expressed by other constraints).

cont **rowSize** [1] : *Expression*
Number of rows in the matrix value.

- **MatrixType.rowDimensionIsInt** Type of the row dimension expression must be *IntType* (Section 3.1.46).
- **MatrixType.rowDimensionValue** Value must be a constant (and at least one, as expressed by other constraints).

3.1.54 ModelDeclaration (class)

Toplevel declaration. Defines the experiment being performed. It is allowed to have several model declarations in a Chi specification.

- **ModelDeclaration.noReturn** The *BehaviourDeclaration.statements* (Section 3.1.10) reference may not contain a *ReturnStatement* (Section 3.1.67).

EObject
└ *ChiObject* (Section 3.1.20)

└ *Declaration* (Section 3.1.26)
└ *BehaviourDeclaration* (Section 3.1.10)
└ *ModelDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **statements** [1..*] : *Statement* (inherited from *BehaviourDeclaration*)
Body of statements of the definition.

cont **variables** [0..*] : *VariableDeclaration* (inherited from *BehaviourDeclaration*)
Formal parameters and local variables of the definition. Both kinds of variables are read/write, as parameters are always call-by-value.

3.1.55 Name (class)

Extra class to attach a position to a name.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Name*

Direct derived classes: *EnumValue* (Section 3.1.34)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **name** [1] : *ChiIdentifier*
Name contained by the class.

3.1.56 Number (class)

Unsigned number expression literal.

- **ChiNumber.type** Type of the number literal must be a *IntType* (Section 3.1.46).

EObject
└ *ChiObject* (Section 3.1.20)

└ *Expression* (Section 3.1.36)
└ *Number*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

attr **value** [1] : *ChiNumber*
String expressing the value of the natural number.

3.1.57 PassStatement (class)

Empty statement, ends immediately, and has no side effects.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement* (Section 3.1.76)
└ *PassStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.58 Position (class)

Position interval in a source file.

Reuse position metamodel documentation.

Position should have a filename attached.

Position should not have startOffset and endOffset?

EObject
└ *Position*

Direct derived classes: none

attr **endColumn** [1] : *EInt*
Column number of the character behind the interval.

attr **endLine** [1] : *EInt*
Line number of the character behind the interval.

attr **endOffset** [1] : *EInt*
Offset of the character behind the interval in the file.

attr **source** [0..1] : *EString*
Source file name of the position.

attr **startColumn** [0..1] : *EInt*
Column number of the first character in the interval.

attr **startLine** [1] : *EInt*
Line number of the first character in the interval.

attr **startOffset** [1] : *EInt*
Offset of the first character of the interval in the file.

3.1.59 ProcessDeclaration (class)

Parameterized definition of a process.

- **ProcessDeclaration.noReturn** The *BehaviourDeclaration.statements* (Section 3.1.10) reference may not contain a *ReturnStatement* (Section 3.1.67).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Declaration* (Section 3.1.26)
└ *BehaviourDeclaration* (Section 3.1.10)
└ *ProcessDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **statements** [1..*] : *Statement* (inherited from *BehaviourDeclaration*)
Body of statements of the definition.

cont **variables** [0..*] : *VariableDeclaration* (inherited from *BehaviourDeclaration*)
Formal parameters and local variables of the definition. Both kinds of variables are read/write, as parameters are always call-by-value.

3.1.60 ProcessInstance (class)

Single instantiated process in a *CreateCase* (Section 3.1.25).

EObject
└ *ChiObject* (Section 3.1.20)
└ *CreateCase* (Section 3.1.25)
└ *ProcessInstance*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **call** [1] : *Expression*
Instantiation expression.

- **ProcessInstance.type** The type of instantiation expression should be a *InstanceType* (Section 3.1.45).

cont **var** [0..1] : *Expression*
Variable for assigning the process instance to.

- **ProcessInstance.varType** The type of the *ProcessInstance.var* (Section 3.1.60) attribute expression must be of type *InstanceType* (Section 3.1.45).

3.1.61 ProcessReference (class)

Reference to a process declaration.

- **ProcessReference.type** Type of the process reference must be a *ProcessType* (Section 3.1.62) class, with matching formal parameeters.
- **ProcessReference.scope** The name of the reference must refer to the referenced process declaration in the scope of the expression.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *ProcessReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

ref **process** [1] : *ProcessDeclaration*
Reference to the referenced process declaration.

- **ProcessReference.processNotNull** For type-checked chi models, the reference should not be `null`.

Check that the not-null requirement is also stated with other references.

3.1.62 ProcessType (class)

Data type of a process declaration.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *ProcessType*

Direct derived classes: *FunctionType* (Section 3.1.42)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **parameterTypes** [0..*] : *Type*
Data types of the formal parameters.

3.1.63 ReadCallExpression (class)

Function application of reading a value from an input stream (often a file).

- **ReadCallExpression.resultType** The type of the expression is the same as the *ReadCallExpression.type* (Section 3.1.63) attribute.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *ReadCallExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **file** [0..1] : *Expression*
Not yet described.

cont **loadType** [1] : *Type*
Type of the read data.

3.1.64 RealNumber (class)

Unsigned real number expression literal.

- **ChiRealNumber.type** Type of the real number literal must be a *RealType* (Section 3.1.65).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *RealNumber*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

attr **value** [1] : *ChiRealNumber*
String describing the value of the unsigned real number.

3.1.65 RealType (class)

Data type of the real nmubers.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Type* (Section 3.1.86)
 ∟ *RealType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.66 ReceiveStatement (class)

Perform receive operation on a communication channel.

- **ReceiveStatement.channelOp** The channel referenced by the *CommunicationStatement.channel* (Section 3.1.21) must allow a receive operation to take place.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Statement* (Section 3.1.76)
 ∟ *CommunicationStatement* (Section 3.1.21)
 ∟ *ReceiveStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **channel** [1] : *Expression* (inherited from *CommunicationStatement*)
Expression evaluating to the channel communicated on.

- **CommunicationStatement.channelType** The type of the *CommunicationStatement.channel* (Section 3.1.21) expression must be a channel type (*ChannelType* (Section 3.1.19)).

cont **data** [0..*] : *Expression* (inherited from *CommunicationStatement*)
For *ReceiveStatement* (Section 3.1.66), addressables that store the communicated data if the channel is not a synchronization channel (See Section 2.1 for the constraints on addressable expressions). For *SendStatement* (Section 3.1.71), the expression evaluating to the value communicated over the channel (if not a synchronization channel).

- **CommunicationStatement.synchronization** If the channel element type is *VoidType* (Section 3.1.96), the data part must be empty.

3.1.67 ReturnStatement (class)

Statement that ends execution of a function.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Statement* (Section 3.1.76)
 ⌞ *ReturnStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **values** [1..*] : *Expression*
Values returned by the function.

If the number of value expressions in a return statement is 1, the type of the value returned by the return statement is the same as the type of the value expression. For longer sequences of value expressions, the type is a record, where each field in the record has the same type as the type of the associated value expression.

- **ReturnStatement.Valuestype** The type of the return statement (as explained above) must be equal to the return type of the function that contains the return statement.

3.1.68 RunStatement (class)

Statement for executing one or more child processes, either just starting then or running them until completion.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Statement* (Section 3.1.76)
 ⌞ *RunStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **cases** [1..*] : *CreateCase*
Instantiated child processes.

attr **startOnly** [1] : *EBoolean*
Only start the child processes, do not wait until they are all finished.

3.1.69 SelectCase (class)

An alternative in a select statement.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *SelectCase*

Direct derived classes: *IteratedSelectCase* (Section 3.1.48)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **body** [1..*] : *Statement*
Sequence of statements to execute if the select case is chosen.

cont **guard** [0..1] : *Expression*
Optional guard expression that should hold for the case to be chosen.

- **SelectCase.guardType** If the guard is present (i.e. not `null`), the type of the guard expression should be *BoolType* (Section 3.1.14).

3.1.70 SelectStatement (class)

Selection of the next statement to execute from several alternatives based on guard expressions and ability to execute the next statement.

- **SelectStatement.notInFunction** The select statement may not be used in a function (*FunctionDeclaration.statements* (Section 3.1.40)).

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Statement* (Section 3.1.76)
 ⌞ *SelectStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **cases** [1..*] : *SelectCase*
Sequence of cases that belong to this select statement.

3.1.71 SendStatement (class)

Perform send operation on a communication channel.

- **SendStatement.channelOp** The channel referenced by the *CommunicationStatement.channel* (Section 3.1.21) must allow a send operation to take place.

EObject
- *ChiObject* (Section 3.1.20)
- *Statement* (Section 3.1.76)
- *CommunicationStatement* (Section 3.1.21)
- *SendStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **channel** [1] : *Expression* (inherited from *CommunicationStatement*)
Expression evaluating to the channel communicated on.

- **CommunicationStatement.channelType** The type of the *CommunicationStatement.channel* (Section 3.1.21) expression must be a channel type (*ChannelType* (Section 3.1.19)).

cont **data** [0..*] : *Expression* (inherited from *CommunicationStatement*)
For *ReceiveStatement* (Section 3.1.66), addressables that store the communicated data if the channel is not a synchronization channel (See Section 2.1 for the constraints on addressable expressions). For *SendStatement* (Section 3.1.71), the expression evaluating to the value communicated over the channel (if not a synchronization channel).

- **CommunicationStatement.synchronization** If the channel element type is *VoidType* (Section 3.1.96), the data part must be empty.

3.1.72 SetExpression (class)

Expression denoting a set value.

- **SetExpression.type** The type of a set expression is a *SetType* (Section 3.1.73).
- **SetExpression.elementsType** The type of each element expression of a set expression must be the same as the *SetType.elementType* (Section 3.1.73) attribute of its type.

EObject
- *ChiObject* (Section 3.1.20)

└ *Expression* (Section 3.1.36)
└ *SetExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **elements** [0..*] : *Expression*
Element expressions of the set.

3.1.73 SetType (class)

Data type of a set value.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *SetType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **elementType** [1] : *Type*
Type of the elements of the set.
Check that the void type gets excluded at the appropriate points

3.1.74 SliceExpression (class)

Take a slice of a list. First value in the slice is at the index given by *SliceExpression.start* (Section 3.1.74) (if omitted, 0 is taken). Iteratively add next values by incrementing the index by *SliceExpression.step* (Section 3.1.74) (if omitted, 1 is taken), until the index is equal or greater than the value expressed by *SliceExpression.end* (Section 3.1.74). If the end expression is omitted, it is equal to the length of the list.

EObject

└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *SliceExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **end** [0..1] : *Expression*
Upper limit of the slice. The list item with this index is not included in the slice.

- **SliceExpression.endType** If given, the type of the end expression must be *IntType* (Section 3.1.46) or *IntType* (Section 3.1.46).

cont **source** [1] : *Expression*
Source expression of the slice.

- **SliceExpression.sourceType** The source expression of a slice must have a *ListType* (Section 3.1.50) type.

cont **start** [0..1] : *Expression*
Lower index of the slice.

- **SliceExpression.endType** If given, the type of the end expression must be *IntType* (Section 3.1.46) or *IntType* (Section 3.1.46).

cont **step** [0..1] : *Expression*
Increment of the index.

- **SliceExpression.endType** If given, the type of the end expression must be *IntType* (Section 3.1.46).

3.1.75 Specification (class)

Class denoting a complete Chi specification.

EObject
└ *Specification*

Direct derived classes: none

cont **declarations** [0..*] : *Declaration*
Global declarations of the specification.

- **Specification.namesUnique** Each name of its declarations (*Specification.declarations* (Section 3.1.75)) must be unique.
- **Specification.hasModel** A specification must have at least one *ModelDeclaration* (Section 3.1.54).

3.1.76 Statement (abstract class)

Abstract base class of a statement.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Statement*

Direct derived classes: *AssignmentStatement* (Section 3.1.8), *BreakStatement* (Section 3.1.15), *CommunicationStatement* (Section 3.1.21), *ContinueStatement* (Section 3.1.24), *DelayStatement* (Section 3.1.27), *ForStatement* (Section 3.1.39), *IfStatement* (Section 3.1.44), *PassStatement* (Section 3.1.57), *ReturnStatement* (Section 3.1.67), *RunStatement* (Section 3.1.68), *SelectStatement* (Section 3.1.70), *TerminateStatement* (Section 3.1.80), *WhileStatement* (Section 3.1.97), *WriteStatement* (Section 3.1.98)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.77 StdLibFunctionReference (class)

Reference to a standard library function.

- **StdLibFunctionReference.type** Type of the reference must be the function type of the referenced function.

Math functions

Library function	Parameter types	Result type
Sign	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
Sign	*RealType* (Section 3.1.65)	*IntType* (Section 3.1.46)
Abs	*IntType* (Section 3.1.46)	*IntType* (Section 3.1.46)
Abs	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Exp	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Ln	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Log	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Sqrt	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Cbrt	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Floor	*RealType* (Section 3.1.65)	*IntType* (Section 3.1.46)
Ceil	*RealType* (Section 3.1.65)	*IntType* (Section 3.1.46)
Round	*RealType* (Section 3.1.65)	*IntType* (Section 3.1.46)
Sin	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Cos	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Tan	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Asin	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Acos	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Atan	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Sinh	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Cosh	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Tanh	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Asinh	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Acosh	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)
Atanh	*RealType* (Section 3.1.65)	*RealType* (Section 3.1.65)

Conversion functions

Library function	Parameter types	Result type
Int2Real	*IntType* (Section 3.1.46)	*RealType* (Section 3.1.65)
String2Bool	*StringType* (Section 3.1.79)	*BoolType* (Section 3.1.14)
Bool2String	*BoolType* (Section 3.1.14)	*StringType* (Section 3.1.79)
String2Int	*StringType* (Section 3.1.79)	*IntType* (Section 3.1.46)
Int2String	*IntType* (Section 3.1.46)	*StringType* (Section 3.1.79)
String2Real	*StringType* (Section 3.1.79)	*RealType* (Section 3.1.65)
Real2String	*RealType* (Section 3.1.65)	*StringType* (Section 3.1.79)

Container functions

Library function	Parameter types	Result type
Length	*ListType* (Section 3.1.50)	*IntType* (Section 3.1.46)
	SetType (Section 3.1.73)	*IntType* (Section 3.1.46)
	DictType (Section 3.1.28)	*IntType* (Section 3.1.46)
	StringType (Section 3.1.79)	*IntType* (Section 3.1.46)
Empty	*ListType* (Section 3.1.50)	*BoolType* (Section 3.1.14)
	SetType (Section 3.1.73)	*BoolType* (Section 3.1.14)
	DictType (Section 3.1.28)	*BoolType* (Section 3.1.14)
Pop	*ListType* (Section 3.1.50) of type t	type t, *ListType* (Section 3.1.50) of type t
	SetType (Section 3.1.73) of type t	type t, *SetType* (Section 3.1.73) of type t
	DictType (Section 3.1.28) of types k and v	type k, type v, *DictType* (Section 3.1.28) of types k and v
Max	*ListType* (Section 3.1.50) of type t	type t
	SetType (Section 3.1.73) of type t	type t
Min	*ListType* (Section 3.1.50) of type t	type t
	SetType (Section 3.1.73) of type t	type t
Take	*ListType* (Section 3.1.50), *IntType* (Section 3.1.46)	*ListType* (Section 3.1.50)
	StringType (Section 3.1.79), *IntType* (Section 3.1.46)	*StringType* (Section 3.1.79)
Drop	*ListType* (Section 3.1.50), *IntType* (Section 3.1.46)	*ListType* (Section 3.1.50)
	StringType (Section 3.1.79), *IntType* (Section 3.1.46)	*StringType* (Section 3.1.79)
Head	*ListType* (Section 3.1.50)	t_e
HeadReverse	*ListType* (Section 3.1.50)	t_e
Tail	*ListType* (Section 3.1.50)	*ListType* (Section 3.1.50)
TailReverse	*ListType* (Section 3.1.50)	*ListType* (Section 3.1.50)
Sort	*ListType* (Section 3.1.50), t_f	*ListType* (Section 3.1.50)
Insert	*ListType* (Section 3.1.50), t_f, t_e	*ListType* (Section 3.1.50)
Range	*IntType* (Section 3.1.46)	*ListType* (Section 3.1.50)
	IntType (Section 3.1.46), *IntType* (Section 3.1.46)	*ListType* (Section 3.1.50)
Matrix	*ListType* (Section 3.1.50), *IntType* (Section 3.1.46)	*MatrixType* (Section 3.1.53)

For the 'Length' function as well as the 'Empty' function, the element type of the container

parameter (all except the *StringType* (Section 3.1.79) is not important.

The 'Pop' function gets a value from a non-empty container (with any element type t), and returns the element and the modified container value.

The 'Max' and 'Min' functions return the biggest respectively smallest value in the list or set. These functions only work for types with ordered values, that is, for t is one of *IntType* (Section 3.1.46) or *RealType* (Section 3.1.65).

The 'Take' and 'Drop' functions on lists, and the 'Tail' and 'TailReverse' functions, take lists with any element type, and return lists with the same element type.

The 'Head' and 'HeadReverse' functions take lists of any element type t_e, and return elements.

The 'Sort' and 'Insert' functions accept lists of any element type t_e, and also return such lists. The t_f parameter is a compare function on two elements, returning a value of type *IntType* (Section 3.1.46). The value returned by the compare function should be negative if the first parameter is 'smaller' than the second parameter, zero if they are equal, and positive otherwise.

The 'Range' function constructs a list with values of the given interval. If only the (exclusive) upper limit is given, the interval starts at 0. Otherwise, the interval runs from the lower limit up to and excluding the upper limit.

The 'Matrix' function converts a list to a matrix type with a single row. The list must have a *RealType* (Section 3.1.65) element type. The second parameter exists for the purpose of type checking. It must evaluate to a compile-time constant, and indicates the length of the list.

Distribution functions

The following distribution functions construct distributions with various shapes and forms.

Library function	**Parameter types**	**Result type**
Constant	*BoolType* (Section 3.1.14)	*DistributionType* (Section 3.1.31) with *BoolType* (Section 3.1.14) result type.
	IntType (Section 3.1.46)	*DistributionType* (Section 3.1.31) with *IntType* (Section 3.1.46) result type.
	RealType (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Bernoulli	*RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *BoolType* (Section 3.1.14) result type.
Binomial	*RealType* (Section 3.1.65), *IntType* (Section 3.1.46)	*DistributionType* (Section 3.1.31) with *IntType* (Section 3.1.46) result type.
Geometric	*RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *IntType* (Section 3.1.46) result type.
Poisson	*RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *IntType* (Section 3.1.46) result type.
Uniform	*IntType* (Section 3.1.46), *IntType* (Section 3.1.46)	*DistributionType* (Section 3.1.31) with *IntType* (Section 3.1.46) result type.
	RealType (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.

Library function	**Parameter types**	**Result type**
Beta	*RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Erlang	*IntType* (Section 3.1.46), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Exponential	*RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Gamma	*RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
LogNormal	*RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Normal	*RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Triangle	*RealType* (Section 3.1.65), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Random	-	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
Weibull	*RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type.
SetSeed	*DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46)	*DistributionType* (Section 3.1.31)

For *StdLibFunctions.SetSeed* (Section 3.1.6), the result types of both distribution types must be the same.

Library function	**Parameter types**	**Result type**
Sample	*DistributionType* (Section 3.1.31) with *BoolType* (Section 3.1.14) result type	*DistributionType* (Section 3.1.31) with *BoolType* (Section 3.1.14) result type, *BoolType* (Section 3.1.14).
	DistributionType (Section 3.1.31) with *IntType* (Section 3.1.46) result type	*DistributionType* (Section 3.1.31) with *IntType* (Section 3.1.46) result type, *IntType* (Section 3.1.46).
	DistributionType (Section 3.1.31) with *RealType* (Section 3.1.65) result type	*DistributionType* (Section 3.1.31) with *RealType* (Section 3.1.65) result type, *RealType* (Section 3.1.65).

The *StdLibFunctions.SampleFunc* (Section 3.1.6) function (not to be confused with the '`sample`' unary operator *UnaryOperators.Sample* (Section 3.1.7)) takes a distribution, and returns a record with the updated distribution and the computed sample.

In case the above distribution creation functions do not suffice, the following functions can be used to compute samples using basic random generators (distributions created with *StdLibFunctions.Random* (Section 3.1.6)).

Each function takes one or more basic random generators, and the parameters of the distribution to compute. The answer is a record with the updated random generators (which should be kept and used again the next time), and the computed sample value. Some algorithms compute two sample values at the same time. Both these values should be used before computing a next batch of samples.

Library function	Parameter types	Result type
DrawBernoulli	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *BoolType* (Section 3.1.14))
DrawBeta	*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))
DrawBinomial	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *IntType* (Section 3.1.46)	(*DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46))
DrawErlang	*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))
DrawExponential	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))
DrawGamma	*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))
DrawGeometric	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46))
DrawLogNormal	*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65))
DrawNormal	*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65))
DrawPoisson	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65),	(*DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46))
DrawTriangle	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))
DrawUniform	*DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46), *IntType* (Section 3.1.46)	(*DistributionType* (Section 3.1.31), *IntType* (Section 3.1.46))
	DistributionType (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))
DrawWeibull	*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65), *RealType* (Section 3.1.65)	(*DistributionType* (Section 3.1.31), *RealType* (Section 3.1.65))

File functions

Library function i Parameter types	Result type	
Open	*StringType* (Section 3.1.79), *StringType* (Section 3.1.79)	*FileType* (Section 3.1.38)
Close	*FileType* (Section 3.1.38)	*BoolType* (Section 3.1.14)

The 'Open' function constructs a connection to a file. First parameter is the name of the file. Second parameter is the mode (`r` or `w`).

The 'Close' function denotes that all data has been read or written, and the connection may be dropped.

Instead of a file-name use a url-like notation to keep extensions possible?

Guard functions

Library function	Parameter types	Result type
Timeout	*VariableReference* (Section 3.1.95)	*BoolType* (Section 3.1.14)

The 'Timeout' function takes a parameter referencing a timer variable and gives `true` if the timer has timed out.

Timers are now much more useful, change the docs.

Check whether this text represents reality.

EObject
- *ChiObject* (Section 3.1.20)
- *Expression* (Section 3.1.36)
- *BaseFunctionReference* (Section 3.1.9)
- *StdLibFunctionReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

attr **function** [1] : *StdLibFunctions*
Referenced function.

3.1.78 StringLiteral (class)

Expression denoting a string value.

- **StringLiteral.type** The type of the string literal expression is a *StringType* (Section 3.1.79).

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Expression* (Section 3.1.36)
 ∟ *StringLiteral*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

attr **value** [1] : *EString*
Value of the string literal. The value is the literal value, no quotes around it, and no escaping of characters takes place in the value.

3.1.79 StringType (class)

Data type for string values.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Type* (Section 3.1.86)
 ∟ *StringType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.80 TerminateStatement (class)

Statement to terminate the execution of a Chi program.

Execution of this statement stops all execution activity of the Chi program (It is a global self-destruct.)

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Statement* (Section 3.1.76)
 ⌞ *TerminateStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.81 TimeLiteral (class)

Expression to refer to the current simulated time.

- **TimeLiteral.type** Type of the expression is *RealType* (Section 3.1.65).

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Expression* (Section 3.1.36)
 ⌞ *TimeLiteral*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

3.1.82 TimerType (class)

Type of a timer.

EObject
 ⌞ *ChiObject* (Section 3.1.20)
 ⌞ *Type* (Section 3.1.86)
 ⌞ *TimerType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.83 TupleExpression (class)

Expression denoting a tuple literal value.

- **TupleExpression.type** The type of a tuple expression is a *TupleType* (Section 3.1.85), with the type of its fields being equal to the types of the *TupleExpression.fields* (Section 3.1.83) expressions.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Expression* (Section 3.1.36)
└ *TupleExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **fields** [2..*] : *Expression*
Child expressions of a tuple expression.

3.1.84 TupleField (class)

Type of a single field in a tuple type.

EObject
└ *ChiObject* (Section 3.1.20)
└ *TupleField*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [0..1] : *Name*
Name of the field.

cont **type** [1] : *Type*
Type of the field.

3.1.85 TupleType (class)

Tuple type.

- **TupleType.unique** Field names of the fields should be unique within the tuple type.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *TupleType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **fields** [2..*] : *TupleField*
Available fields in the tuple type.

3.1.86 Type (abstract class)

Base class of all data types.

- **Type.nocycle** A data type may not refer (indirectly) to itself.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type*

Direct derived classes: *BoolType* (Section 3.1.14), *ChannelType* (Section 3.1.19), *DictType* (Section 3.1.28), *DistributionType* (Section 3.1.31), *EnumTypeReference* (Section 3.1.33), *FileType* (Section 3.1.38), *InstanceType* (Section 3.1.45), *IntType* (Section 3.1.46), *ListType* (Section 3.1.50), *MatrixType* (Section 3.1.53), *ProcessType* (Section 3.1.62), *RealType* (Section 3.1.65), *SetType* (Section 3.1.73), *StringType* (Section 3.1.79), *TimerType* (Section 3.1.82), *TupleType* (Section 3.1.85), *TypeReference* (Section 3.1.88), *UnresolvedType* (Section 3.1.92), *VoidType* (Section 3.1.96)

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.87 TypeDeclaration (class)

Declaration to attach a name to a data type.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Declaration* (Section 3.1.26)
└ *TypeDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **name** [1] : *Name* (inherited from *Declaration*)
Name of the declaration.

- **Declaration.name** Name of a declaration should be non-empty.

cont **type** [1] : *Type*
Type attached to the name.

3.1.88 TypeReference (class)

Reference to a type declaration.

- **TypeReference.type** Type of the type variable is equal to the type of the referenced type declaration.
- **TypeReference.nocycle** A type variable may not reference itself (neither directly nor indirectly).

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *TypeReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

ref **type** [1] : *TypeDeclaration*
Referenced type declaration o the type variable.

3.1.89 UnaryExpression (class)

Expression operation with one child expression.

EObject
- *ChiObject* (Section 3.1.20)
- *Expression* (Section 3.1.36)
- *UnaryExpression*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

cont **child** [1] : *Expression*
Child expression.

cont **operator** [1] : *UnaryOp*
Operation performed on the child expression value.

- **UnaryExpression.operatorArgument** The type of the unary operator argument must match with the type of the child expression (*Expression.type* (Section 3.1.36) of *UnaryExpression.child* (Section 3.1.89)).
- **UnaryExpression.resultType** The result type of the unary expression must match with the result type of the used operator.

The allowed child types and result types for each operator are listed in the tables below.

Inverse operator

Child type	Result type
BoolType (Section 3.1.14)	*BoolType* (Section 3.1.14)

Negate operator

Child type	Result type
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)

Sample operator

The sample operator takes a distribution as child (an expression with a *DistributionType* (Section 3.1.31) as type). The result type of the sample operator depends on the element type of the child type in the following way:

Child element type	Result type
BoolType (Section 3.1.14)	*BoolType* (Section 3.1.14)
IntType (Section 3.1.46)	*IntType* (Section 3.1.46)
RealType (Section 3.1.65)	*RealType* (Section 3.1.65)

- **UnaryExpression.sampleNotInFunction** The *UnaryOperators.Sample* (Section 3.1.7) may not be used in unary expressions (*UnaryExpression* (Section 3.1.89)) in a function declaration (*FunctionDeclaration* (Section 3.1.40)).

3.1.90 UnaryOp (class)

Extra class for attaching a position to a unary operator.

EObject
- *ChiObject* (Section 3.1.20)
- *UnaryOp*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **op** [1] : *UnaryOperators*
Unary operator contained in the class.

3.1.91 UnresolvedReference (class)

Reference to a named value which is not yet resolved.

- **UnresolvedReference.notInChecked** *UnresolvedReference* (Section 3.1.91) should not occur in type checked chi models.
- **UnresolvedReference.type** The *UnresolvedReference.type* (Section 3.1.91) should be `null`.

EObject
- *ChiObject* (Section 3.1.20)
- *Expression* (Section 3.1.36)
- *UnresolvedReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

attr **name** [1] : *ChiIdentifier*
Name given to the unresolved reference.

3.1.92 UnresolvedType (class)

Unresolved type reference.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Type* (Section 3.1.86)
└ *UnresolvedType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **name** [1] : *ChiIdentifier*
Name of the unresolved type reference.

- **UnresolvedType.notInChecked** *UnresolvedType* (Section 3.1.92) should not occur in type checked chi models.

3.1.93 Unwind (class)

Unwind loop.

EObject
└ *ChiObject* (Section 3.1.20)
└ *Unwind*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **source** [1] : *Expression*
Source expression to unwind.

- **Unwind.sourceType** The type of the *Unwind.source* (Section 3.1.93) expression must be an iterable container type, a *ListType* (Section 3.1.50), *SetType* (Section 3.1.73), or *DictType* (Section 3.1.28).

cont **variables** [1..*] : *VariableDeclaration*
Local variables of the unwind.

- **Unwind.variableNames** The names of the variables should be unique within an unwind.

What do we decide on var overloading/scoping rule?

3.1.94 VariableDeclaration (class)

Declaration of a variable or formal parameter.

EObject
└ *ChiObject* (Section 3.1.20)
└ *VariableDeclaration*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **initialValue** [0..1] : *Expression*
Optional initial value of the variable.

- **VariableDeclaration.initialValue** The initial value of a *VariableDeclaration* (Section 3.1.94) must be `null` if *VariableDeclaration.parameter* (Section 3.1.94) is true.
- **VariableDeclaration.initialValueType** If the initial value is not `null`, the type of the initial value expression must be equal to the type of the variable.

Timer variables are not allowed as parameter.

cont **name** [1] : *Name*
Name of the variable or formal parameter.

attr **parameter** [1] : *EBoolean*
Variable is a formal parameter.

cont **type** [0..1] : *Type*
Type of the variable.

3.1.95 VariableReference (class)

Expression referencing a variable.

- **VariableReference.type** Type of the variable reference is the same as the type of the referenced variable.
- **VariableReference.inScope** The referenced variable must be in the current scope.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Expression* (Section 3.1.36)
 ∟ *VariableReference*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **type** [0..1] : *Type* (inherited from *Expression*)
Type of the expression.

- **Expression.notNull** In type-checked models, *Expression.type* (Section 3.1.36) is never `null`.
- **Expression.noVoid** Type of an expression is never *VoidType* (Section 3.1.96).

ref **variable** [1] : *VariableDeclaration*
Referenced variable of the variable reference expression.

3.1.96 VoidType (class)

Data type without any values, used as data type for synchronization channels.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Type* (Section 3.1.86)
 ∟ *VoidType*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

3.1.97 WhileStatement (class)

The while statement repeatedly tests its condition, and if it holds, it executes its body. The repetition ends when the condition evaluates to `false`.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Statement* (Section 3.1.76)
 ∟ *WhileStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

cont **body** [1..*] : *Statement*
Sequence of statements to be executed when the condition of the while statement evaluates to `true`.

cont **condition** [1] : *Expression*
Condition of the while statement, used for testing whether the body of the while statement should be executed.

- **WhileStatement.ConditionType** Type of the condition expression must be a *BoolType* (Section 3.1.14).

3.1.98 WriteStatement (class)

Statement for producing output onto an output stream.

EObject
 ∟ *ChiObject* (Section 3.1.20)
 ∟ *Statement* (Section 3.1.76)
 ∟ *WriteStatement*

Direct derived classes: none

cont **position** [0..1] : *Position* (inherited from *ChiObject*)
Position of the construct in the source file.

attr **addNewline** [1] : *EBoolean*
Append a newline after writing the data.

attr **toFile** [1] : *EBoolean*
Output stream is a file (rather than a terminal).

cont **values** [1..*] : *Expression*

Sequence of values to write. If *WriteStatement.toFile* (Section 3.1.98) holds, the first value is the stream to write to. Values of type string are written unchanged, other data values are converted to string first.

Note: Although an implementation should attempt to output a representative string for a value, some information may get lost during the conversion.

- **WriteStatement.fileValue** If *WriteStatement.toFile* (Section 3.1.98) holds, the type of the first value should be of type *FileType* (Section 3.1.38).

Chapter 4

Legal

Bibliography

[1] Eclipse Foundation. Eclipse Foundation Project Handbook. https://www.eclipse.org/projects/handbook/#starting-incubation.

[2] Contributors to the Eclipse Foundation. Eclipse Supervisory Control Engineering Toolkit (Eclipse ESCET). https://eclipse.dev/escet.