Modelica Change Proposal MCP-0021
Component Iterators
Status: In Development
2016-12-13, version v3, #1848

Proposed Changes to the Modelica Language Specification
Version 3.3 Revision 1

Table of Contents

Chapter 6

Interface or Type Relationships

Chapter 7

Inheritance, Modification, and Redeclaration

Chapter 8

Equations

Chapter 9

Connectors and Connections

Chapter 10

Arrays

10.1 Array Declarations

10.2 Flexible Array Sizes

10.3 Built-in Array Functions

10.4 Vector, Matrix and Array Constructors

10.4.1 Array Constructor with Iterators

An expression:

```
"{" expression for iterators "}"
```

or

```
array "(" expression for iterators ")"
```

is an array constructor with iterators. The expressions inside the iterators of an array constructor shall be vector expressions. They are evaluated once for each array constructor, and is evaluated in the scope immediately enclosing the array constructor.

For an iterator:

```
IDENT in array_expression
```

the loop-variable, `IDENT`, is in scope inside expression in the array construction. The loop-variable may hide other variables, as in for-clauses. The loop-variable has the same type as the type of the elements of array_expression; and can be simple type as well as a record type. The loop-variable will have the same type for the entire loop - i.e. for an array_expression {1,3.2} the iterator will have the the type of the type-compatible expression (Real) for all iterations. For deduction of ranges, see Section **Error! Reference source not found.**; and for using types as range see Section **Error! Reference source not found.**.

For an iterator:

```
IDENT in class NAME
```

the `NAME` must be (using normal lexical lookup) the name of a `model`, `block`, `connector`, `record`, or `operator record`, `type` specialized class. The loop-variable, `IDENT`, is resolved to every instance of class `NAME` that is on the same or a lower level as the parent of the component containing the iterator expression. *[For example if the iterator expression is in a model `A` and the simulation contains a component of `A` with component path `top.x.a` or `top.x.a[:]` only components in top.x are considered; and if the component-path is `top.x[n].a` only components in `top.x[1]` are considered for `top.x[1].a` and only `top.x[2]` for `top.x[2].a`. If the component path for the model `A` is `a` all components are considered. The reason for this is to allow graphical composition where one sub-component use this and other sub-components are considered.]* If no instance of class `NAME` is present, the constructed array has zero dimension. The full path name of the actual instance can be inquired with the built-in function `IDENT.getInstanceName`().

[*Example:*

```
record Observation "Data that shall be observed"
  constant String name;
  parameter Real m;
  parameter Real v2[3];
  Real           r2;
end Observation;

model ObservedModel "Model that produces the data"
  parameter Real p=2;
  parameter Real v[3] = {-1,2.5,6};
  Real    r;
```

```
  Real    w[3];
  Boolean b;
  Integer i;
  …
end ObservedModel;

model Submodel
  ObservedModel c1(p=3, v={1,-4,8});
  ObservedModel c2;
end Submodel;

model MyObs
  Integer i2[:]      = {c.i+3 for c in class ObservedModel};
  Observation obs[:] = {Observation(m=c.p, v2=c.v, r2=c.r, name=c.getInstanceName())
                                   for c in class ObservedModel};
  Integer i3[:]      = {c.i for c in class ClassNotUsedInMyModel};
end MyObs;

model MyModel1
  Submodel s1;
  Submodel s2;

  MyObs observer;

end MyModel1;

model MyModel2
  MyModel1 myModel1;
  Submodel s3;
end MyModel2;
```

In every iteration of the for loop, the iterator variable c is a reference to an instance of class ObservedModel present in MyModel1. Therefore, the above model is equivalent to the following expanded form:

```
model MyObs_Expanded
  Integer i2[:];
  Observation obs[:];
  Integer i3[:]
end MyObs_Expanded;

model MyModel1_Expanded
  Submodel s1;
  Submodel s2;

  MyObs_Expanded observer(
       i2= {s1.c1.i+3, s1.c2.i+3,  s2.c1.i+3, s2.c2.i+3},
       obs={Observation(m=s1.c1.p,  v2=s1.c1.v,  r2=s1.c1.r,
                                   name="MyModel2_Expanded.myModel1.s1.c1"),
                       Observation(m=s1.c2.p,  v2=s1.c2.v,  r2=s1.c2.r,
                                   name=" MyModel2_Expanded.myModel1..s1.c2"),
                       Observation(m=s2.c1.p,  v2=s2.c1.v,  r2=s2.c1.r,
                                   name=" MyModel2_Expanded.myModel1.s2.c1"),
                       Observation(m=s2.c2.p,  v2=s2.c2.v,  r2=s2.c2.r,
                                   name=" MyModel2_Expanded.myModel1.s2.c2")}),
        i3=zeros(0));
end MyModel1_Expanded;

model MyModel2_Expanded
  MyModel1_Expanded myModel1;
  Submodel          s3;
end MyModel2_Expanded;
```

]

10.4.2 Array Concatenation

10.4.3 Vector Construction

10.5 Array Indexing

Chapter 11

Statements and Algorithm Sections

Chapter 12

Functions

Chapter 13

Packages

Chapter 14

Overloaded Operators

Chapter 15

Stream Connectors

Chapter 16

Synchronous Language Elements

Chapter 17

State Machines

Chapter 18

Annotations

Chapter 19

Unit Expressions

Chapter 20

The Modelica Standard Library

Appendix A

Glossary

Appendix B

Modelica Concrete Syntax

```
...
 for_equation :
   for for_indices loop
      { equation ";" }
   end for

 for_statement :
   for for_indices loop
      { statement ";" }
   end for

 for_indices :
   for_index {"," for_index}

 for_index:
   IDENT [ in expression  ] |
   IDENT [ in class NAME ]
```

Appendix C

Modelica DAE Representation

Appendix D

Derivation of Stream Equations