/ards User-Define
ations

stoph Höger
istoph.hoeger@tu-berlin.de

)2.2013 - Openmodelica Workshop 2

Agenda

- ## What is a user-defined relation?

  Short overview and problem description.

- ## How can we get user-defined relations?

  What *semantic* problems need to be solved inside an implementation.

- ## What works today?

  Relations as first-class citizens.

- ## What next?

  Defining relation-resolution

Problem Scope

Current situation

- ## Models are composed by relations.

  Usually `connect`, but also `branch`, `transition` (so called Equation Operators)

- ## The global set of all relations implies a set of equations.

  Actually, they are computed according to some algorithms defined in the language spec.

- ## This process may introduce some symbols

  E.g. `isRoot`, they can only be used in equations!

The Specification's Issue With Connection Semantics

- ## Five Different connection semantics do already exist.

  These are:
  Potential/Flow/Stream/Overconstrained/Transitions
  More are likely to come.

- ## Partially defined in *pseudo-code* (stream)

  This is, in general, a good thing. Unfortunately they cannot be expressed in actual, valid Modelica.

- ## Partially defined in *algorithmically*

  E.g. overconstrained connectors. Again, this is a good thing.

[*Based on the above requirements, the following implementation is recommended:*

```
N = 1, M = 0:
 inStream(m1.c.h_outflow) = m1.c.h_outflow;

N = 2, M = 0:
 inStream(m1.c.h_outflow) = m2.c.h_outflow;
 inStream(m2.c.h_outflow) = m1.c.h_outflow;

N = 1, M = 1:
 inStream(m1.c.h_outflow) = inStream(c1.h_outflow);
 // Additional equation to be generated
 c1.h_outflow = m1.c.h_outflow;

N = 0, M = 2:
 // Additional equation to be generated
 c1.h_outflow = inStream(c2.h_outflow);
 c2.h_outflow = inStream(c1.h_outflow);

All other cases:
 if mj.c.m_flow.min >= 0 for all j = 1:N with j <> i and
     ck.m_flow.max <= 0 for all k = 1:M
 then
   inStream(mi.c.h_outflow) = mi.c.h_outflow;
 else
   si = sum(max(-mj.c.m_flow,0) for j in cat(1,1:i-1, i+1:N) +
        sum(max( ck.m_flow  ,0) for k in 1:M);
   inStream(mi.c.h_outflow) =
    (sum(positiveMax(-mj.c.m_flow,si)*mj.c.h_outflow) +
     sum(positiveMax(ck.m_flow,si)    *inStream(ck.h_outflow)))/
    (sum(positiveMax(-mj.c.m_flow,si)) +
     sum(positiveMax(ck.m_flow,si)))
                                for j in 1:N and i <> j and mj.c.m_flow.min < 0,
                                for k in 1:M             and ck.m_flow.max > 0

 // Additional equations to be generated
 for q in 1:M loop
   if mj.c.m_flow.min >= 0 for all j = 1:N and
       ck.m_flow.max <= 0 for all k = 1:M and k <> q
   then
      cq.h_outflow = 0;
   else
      sq = (sum(max(-mj.c.m_flow,0) for j in 1:N) +
            sum(max( ck.m_flow  ,0) for k in cat(1,1:q-1, q+1:M)));
      cq.h_outflow = (sum(positiveMax(-mj.c.m_flow,sq)* mj.c.h_outflow) +
                      sum(positiveMax(ck.m_flow,sq)    * inStream(ck.h_outflow)))/
                     (sum(positiveMax(-mj.c.m_flow,sq)) +
                      sum(positiveMax(ck.m_flow,sq)))
                                  for j in 1:N and mj.c.m_flow.min < 0,
```

The User's/Implementor's Issue With Connection Semantics

Hard to *implement*

Have you ever tried to just understand the concept of stream in one day as a programmer?

-

Hard to *extend*

You want a new kind of connection? Go write a Specification chapter!

-

Hard to *test*

Is there a contradiction between e.g. the stream semantics and some of the newer additions to the language? Who knows?

The Implementor's Solution

- ## Describe this process by code

  Ideally, an engineer should do it, not a programmer!

- ## (Specification benefit)

  Ship this code as a core library.

What do we need?

Steps needed

- ## Lift variables to the object level.

  A `Real` is not just a number, but an *unknown*! It can be evaluated like anything else.

- ## Lift relations to the object level.

  Relations are basically named records. Records are (hopefully) already implemented in $TOOL

- ## Compute equations out of the relation-sets

  How to do this leaves many degrees of freedom.

Variables as objects

```
 1 model X
 2   Real x,y;
 3   equation
 4
 5   if x <> y then
 6     x = y;
 7   end if;
 8
 9   x = 1;
10 end X;
11
```

Equality can be computed on variables, since they are globally named!

OpenModelica's current answer

```
Warning: In component , in
relation x <> y, <> on
Reals is only allowed
inside functions.
```

But in fact, we know that they are different:

```
 1 model X
 2   Real x,y;
 3   equation
 4
 5   if getInstanceName() + "x" <> get
 6     x = y;
 7   end if;
 8
 9   x = 1;
10 end X;
11
```

Relations as objects

```
connect(x,y);
```

Ideally, we could represent a relation as a special data structure in Modelica:

```
1 relation connect
2   Real left;
3   Real right;
4 end connect;
5
```

Syntax compatible extension

```
connect(x,y);
```

Unfortunately, we do not have that currently.

```
1 record PotentialConnection
2   Real left;
3   Real right;
4   annotation(relation(label="connec
5 end PotentialConnection;
6
```

This record is like a model, except it cannot be constructed implicitly.

Evaluate relations like records

```
model Switch
  parameter Boolean open;
  OnePort lhs;
  OnePort rhs;
  Ground ground;
  equation
  connect(lhs.p, if open then grour
  connect(rhs.n, if open then grour
end Switch;

```

```
[/tmp/X.mo:7:18-7:20:writable] Error:
No viable alternative near token: if
```

Equivalent rewrite

```
 1 model Switch
 2   parameter Boolean open;
 3   OnePort lhs;
 4   OnePort rhs;
 5
 6   Ground ground;
 7   equation
 8   if open then
 9     connect(lhs.p, rhs.n);
10     connect(rhs.n, lhs.p);
11   else
12     connect(lhs.p, ground.p);
13     connect(rhs.n, ground.p);
14   end if;
15 end Switch;
16
```

They should be the same!

Demo: What works now

Simple Example

```
model SimpleCircuit1
  StepVoltage source;
  Resistor R1(R=10);
  Capacitor C1(C=5);
  Inductor I1(I=5);
  Ground ground;
  equation
  connect(source.p, R1.n);
  connect(R1.p, C1.n);
  connect(C1.p, I1.n);
  connect(I1.p, source.n);
  connect(source.n, ground.p);
end SimpleCircuit1;

```

Calculating Connected Sets

```
 1   function connected_sets
 2     input PotentialConnectRelation
 3     output Sets sets;
 4     protected
 5     SimpleGraph graph;
 6     ConnectivityInspector connectiv
 7   algorithm
 8     graph := SimpleGraph(relations
 9     for r in relations loop
10       graph.addVertex(r.left);
11       graph.addVertex(r.right);
12       graph.addEdge(r.left, r.right
13     end for;
14     connectivity := ConnectivityIns
15     sets := Sets(Array(connectivity
16   end connected_sets;
17
```

(Using External Java functions)

Calculating Equations

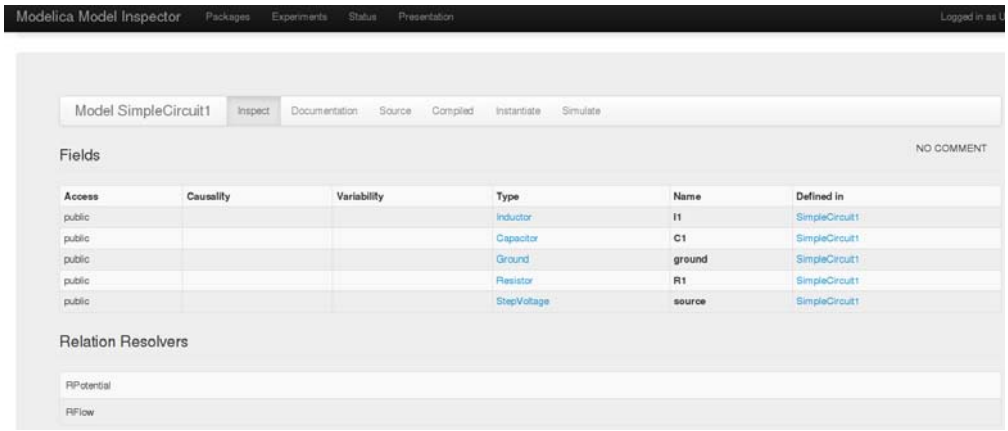
```
1   /* JPotentialConnectRelation is t
2      this needs to come out of an a
3   outer PotentialConnectRelation[:]
4
5   equation
6   for set in connected_sets(JPotent
7     for i in 1:(length(set) - 1) lo
8       set[i+1] = set[i];
9     end for;
10  end for;
11
```

(More or less Modelica)

Result (no flows yet)

- .R1.n.u = .source.p.u
- .R1.p.u = .C1.n.u
- .I1.n.u = .C1.p.u
- .source.n.u = .ground.p.u
- .I1.p.u = .source.n.u
- ...

There would have been a demo...



Calculating Equations differently

```
1   /* JPotentialConnectRelation is t
2      this needs to come out of an a
3   outer PotentialConnectRelation[:]
4
5   equation
6   for set in connected_sets(JPotent
7     for i in 1:(length(set) - 1) lo
8       set[i+1] = (2 - 1) * set[i];
9     end for;
10  end for;
11
```

(Equivalent formulation)

Result (no flows yet)

- (-1,000000 * .R1.n.u) + (1,000000 * .source.p.u) = 0,000000
- (-1,000000 * .C1.n.u) + (1,000000 * .R1.p.u) = 0,000000
- (-1,000000 * .I1.n.u) + (1,000000 * .C1.p.u) = 0,000000
- (-1,000000 * .I1.p.u) + (1,000000 * .ground.p.u) = 0,000000
- (-1,000000 * .ground.p.u) + (1,000000 * .source.n.u) = 0,000000
- ...

Future

Resolution of relations

- ## Create equations based on computation.

  This is actually, what models usually are good for.

- ## Make general purpose data structures available

  This requires a better FFI

- ## Define a calling convention.

  This requires a lot of discussions.