# **vards User-Define**

# stoph Höger istoph.hoeger@tu-berlin.de 02.2013 - 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

Your deck.js Presentation

### **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!

Your deck.js Presentation

### 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.

<pre>n = 1, n = 0: instream(m.c.h_outflow) = m,.c.h_outflow; instream(m,.c.h_outflow) = m,.c.h_outflow; instream(m,.c.h_outflow) = m,.c.h_outflow; instream(m,.c.h_outflow) = m.s.c.h_outflow; // Additional equation to be generated c.h_outflow = m.s.c.h_outflow; n = 0, M = 2: // Additional equation to be generated c.h_outflow = instream(cl.h_outflow); c2.h_outflow = instream(cl.h_outflow); c3.m_flow.max &lt;&lt; 0 for all j = 1:N with j &lt;&gt; i and c.m_flow.max &lt;&lt;= 0 for j in 1:N and i &lt;&gt; j and m,.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m,.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop iff m,.c.m_flow.min &gt;= 0 for all j = 1:N and c.m_flow.max &lt;&lt;= 0 for all j = 1:N and k&lt;&gt; q then c.,.m_flow.max &lt;&lt;= 0 for all k = 1:M and k&lt;&gt; q then c.,.m_flow.max &lt;= 0 for all k = 1:M and k&lt;&gt; q then c.,.h_outflow = 0; else m_q = (sum(max(-m,.c.m_flow,0) for j in 1:N) + mum(positiveMax(cm,.m_flow,0) for j in 0:N, j, m,.c.h_outflow) + (sum(positiveMax(cm,.c.m_flow,0) for j in 0:N, j, j, c.h_outflow) + mum(positiveMax(cm,.c.m_flow,0) for j in 0:N, j, j, j, c.h_outflow) + mum(positiveMax(cm,.c.m_flow,0) for j in 0:N, j, j, j, c.h_outflow) + sum(positiveMax(cm,.c.m_flow,0) for j in 0:N, j, j, j, c.h_outflow) + mum(positiveMax(cm,.c.m_flow,0) for j in 0:N, j, j, c.h_outflow) + sum(positiveMax(cm,.c.m_flow,0) for j in 0:N, j, j, j, c.h_outflow) + sum(positiveMax(cm,.c.m_flow,0) for j in 0;N, j, j,</pre>	N - 1 M - 0:	
<pre>Instrument(====================================</pre>	inStream(m.c.h	outflow) = m.c.b outflow:
<pre>N = 2, M = 0: instream(m,c.h_outflow) = m<sub>2</sub>.c.h_outflow; instream(m,c.h_outflow) = instream(c<sub>1</sub>,h_outflow); // Additional equation to be generated c<sub>1</sub>.h_outflow = m<sub>1</sub>.c.h_outflow); // Additional equation to be generated c1.h_outflow = instream(c2.h_outflow); // Additional equation to be generated c1.h_outflow = instream(c2.h_outflow); c2.h_outflow = instream(c2.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c2.h_outflow)))/ (sum(positiveMax(c_m,flow,a)) * instream(c2.h_outflow)))/ (sum(positiveMax(c_m,flow,a)); for j in 1:N and i &lt;&gt; j and m<sub>3</sub>.c.m_flow.min &lt; 0, for k in 1:N and i &lt;&gt; j and m<sub>3</sub>.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m<sub>3</sub>.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m<sub>3</sub>.c.m_flow.max &lt;&lt; 0 for all j = 1:N and c_1.m_flow.max &lt;&lt; 0 for all k = 1:M and k &lt;&gt; q then c_1.h_outflow = 0; else s_q = (Sum(max(cm,c.m.flow,0)) for i in 1:N) + sum(max(c_1.m_flow,0) for i in 1:N</pre>		energy without one and the
<pre>instream(m,c.h_outflow) = m,.c.h_outflow; instream(m,c.h_outflow) = m,.c.h_outflow; N = 1, W = 1: instream(m,c.h_outflow) = instream(c,.h_outflow); // Additional equation to be generated c.h_outflow = m,c.h_outflow; N = 0, M = 2: // Additional equation to be generated cl.h_outflow = instream(cl.h_outflow); All other cases: if m,c.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and cc.m_flow.max &lt;= 0 for all j = 1:N with j &lt;&gt; i and cc.m_flow.max &lt;= 0 for all k = 1:M then instream(m,c.h_outflow) = m,c.h_outflow; else (sum(mosiliveMax(-m_flow,a)) for j in cat(1,1:i-1, i+1:N) + sum(max(-m_c.m_flow,a)) for k in 1:M); instream(m,c.h_outflow) = m,c.h_outflow) + sum(mosiliveMax(-m_flow,a)) * instream(c,h_outflow)))/ (sum(posiliveMax(-m_flow,a)) * instream(c,h_outflow)))/ (sum(posiliveMax(-m_flow,a)) + sum(posiliveMax(-m_flow,a)) + sum(posiliveMax(-m_flow,a)) = instream(c,h_outflow)))/ (hditional equations to be generated for q in 1:M loop if m,c.m_flow.max &lt;= 0 for all j = 1:N and cc.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then .c.h_outflow = 0; else else sum(max(-m_fc.m.flow,a)) for k in cat(1,1:q-1, q+1:M))); cm, for gin 1:N) = sum(max(-m_flow,a)) for k in 1:N) + sum(posiliveMax(-m_fc.m_flow,a)) * instream(c_h,outflow))/ (sum(mosiliveMax(-m_fc.m_flow,a)) * instream(c_h,outflow))/ (sum(max(c_h,m_flow,a)) for k in cat(1,1:q-1, q+1:M))); cm,h_outflow = 0; else sum(max(-m_fc.m_flow,a)) * instream(c_h,outflow))/ (sum(max(c_h,m_flow,a),m_flow,a)) * instream(c_h,outflow))/ (sum(posiliveMax(m_flow,m_flow,a), * instream(c_h,outflow))/ (sum(posiliveMax(m_flow,m_flow,m_flow,a)) * instream(c_h,outflow) + (sum(posiliveMax(m_flow,m_flow,m_flow,a)) * instream(c_h,outflow))/ (sum(posiliveMax(m_flow,m_flow,m_flow,m_flow,a)) * instream(c_h,outflow))/ (sum(posiliveMax(m_flow,m_</pre>	N = 2, M = 0:	
<pre>Instream (m, c.h_outflow) = m, c.h_outflow; N = 1, W = 1: instream (m, c.h_outflow) = instream(c, h_outflow); // Additional equation to be generated c.h_outflow = m, c.h_outflow; N = 0, M = 2: // Additional equation to be generated cl.h_outflow = instream(cl.h_outflow); c2.h_outflow = instream(cl.h_outflow); c3.h_outflow = instream(cl.h_outflow); c3.h_outflow = m, c.h_outflow); si = sum (max(-m, c.m_flow, n) for j in cat(1,1:i-1, i+1:N) + sum (max(-m, c.m_flow, n) for j in cat(1,1:i-1, i+1:N) + sum (max(-m, c.m_flow, n) for k in 1:N); instream (m, c.h_outflow) = (sum (positiveMax(-m, flow, n)) for j in cat(1,1:i-1, i+1:N) + sum (positiveMax(-m, flow, n)) + sum (positiveMax(-m, flow, n)) + sum (positiveMax(-m, flow, n))) for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m_j.c.m_flow.max &lt;= 0 for all j = 1:N and c_i.m_flow.max &lt;= 0 for all j = 1:N and c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.h_outflow = 0; else s_q = (Sum(max(-m, c.m.flow, 0)) for k in cat(1,1:q-1, q+1:N))); c_i.h_outflow = 0; else s_q = (Sum(max(-m, c.m.flow, 0)) for k in cat(1,1:q-1, q+1:N))); c_i.h_outflow = 0; else s_q = (Sum(max(-m, c.m.flow, 0)) for k in cat(1,1:q-1, q+1:N))); c_i.h_outflow = (Sum(positiveMax(-m, c.m. flow, m)) * instream(c_i.h_outflow))/ (Sum(max(-c_i.m_flow, m) for f in 1:N) + sum(positiveMax(-m, flow, m) * instream(c_i.h_outflow)))/ (Sum(max(-c_i.m_flow, m) for f in 1:N) + sum(positiveMax(-m, flow, m) * instream(c_i.h_outflow)))/ (Sum(max(-c_i.m_flow, m) for f in 1:N) + sum(positiveMax(-m, flow, m) * instream(c_i.h_outflow)))/ (Sum(max(-c_i.m_flow, m) for f in 0; for 0; m 1; flow, m) * instream(c_i.h_outflow))/ (Sum(positiveMax(-m, flow, m) * instream (c_i.h_outflow))/ (Sum(positiveMax(-m, flow, m) * instream (c_i.h_outflow))/ (Sum(positiveMax(-m, flow, m) * instream (c_i.h_outflow))/ (Sum(positiveMax(-m, flow, m) * instream(c_i.h_outflow))/ (Sum(positiveMax(-m, flow, m) * instream(c_i.h_outfl</pre>	inStream(m1.c.h	outflow) = m2.c.h_outflow;
<pre>N = 1, M = 1: instream(n,c.h_outflow) = instream(c,h_outflow); // Additional equation to be generated c,h_outflow = m,c.h_outflow; N = 0, M = 2: // Additional equation to be generated c1.h_outflow = instream(c2.h_outflow); c2.h_outflow = instream(c2.h_outflow); c2.h_outflow = instream(c1.h_outflow); c2.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); s1.f m,c.m_flow.max &lt;= 0 for all k = 1:M instream(m,c.h_outflow) = m,c.h_outflow; else s1 = sum(max(-m,c.m_flow,0) for j in cat(1,1:i-1, i+1:N) + isum(mositiveMax(-m_iflow,a)); instream(m,c.h_outflow) = instream(c_i,h_outflow)))/ (sum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); sum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(positiveMax(-m_iflow,a)); isum(max(= 0 for all j = 1:N and c_i.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m,i.c.m_flow.max &lt;= 0 for all j = 1:N and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_i.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c</pre>	instream(m <sub>2</sub> .c.h	outflow) = m1.c.h_outflow;
<pre>instream(m,c.h_outflow) = instream(c,h_outflow); // Additional equation to be generated c_1.h_outflow = m_1.c.h_outflow; N = 0, M = 2: // Additional equation to be generated c1.h_outflow = instream(c.h_outflow); All other cases: if m_ic.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and c_t.m_flow.min &gt;= 0 for all k = 1:M then instream(m_i.c.h_outflow) = m_i.c.h_outflow; else si = sum(max(-m_ic.m_flow,o) for j in cat(1,1:i-1, i+1:N) + sum(max(o_t.m_flow, a) for k in 1:N); instream(m_i.c.h_outflow) = (sum(positiveMax(-m_i.c.m_flow,a))*m_i.c.h_outflow) + sum(positiveMax(c_m_flow,a)) for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m_i.c.m_flow.min &gt;= 0 for all j = 1:N and c_t.m_flow.max &lt;= 0 for all j = 1:N and c_t.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_t.m_flow.max &lt;= 0</pre>	N = 1, M = 1:	
<pre>// Additional equation to be generated c_i,h_outflow = m,c.h_outflow; N = 0, M = 2: // Additional equation to be generated ci.h_outflow = instream(c2.h_outflow); c2.h_outflow = instream(c2.h_outflow); c2.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c3.h_outflow = instream(c1.h_outflow); c1.h_outflow = instream(c2.h_outflow); c1.h_outflow = instream(c2.h_outflow); c1.h_outflow = instream(c2.h_outflow); c1.h_outflow = instream(c2.h_outflow); c1.h_outflow = instream(c2.h_outflow); c1.h_outflow = 0; c1.h_outflow = 0; c1.h_outflow</pre>	inStream(m1.c.h	_outflow) = inStream(c1.h_outflow);
<pre>c<sub>1</sub>,h_outflow = m<sub>1</sub>,c.h_outflow; N = 0, M = 2: // Additional equation to be generated cl.h_outflow = inStream(cl.h_outflow); All other cases: if m<sub>1</sub>,c.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and c.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and c.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and c.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and c.m_flow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and c.m_flow.max &lt;= 0 for all s = 1:M then inStream(m<sub>1</sub>.c.h_outflow) = m<sub>1</sub>.c.h_outflow; else (sum(mositiveMax(-m_flow, a)) for k in 1:M); inStream(m_c.ch_outflow) = (n, c.h_outflow) + sum(positiveMax(-m_flow, a)) * inStream(ck_houtflow)))/ (sum(positiveMax(-m_flow, a))) + sum(positiveMax(-m_flow, a))) + for j in 1:N and i &lt;&gt; j and m<sub>2</sub>.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m<sub>2</sub>.c.m_flow.min &lt; 0, for k in 1:N and i &lt;&gt; j and m<sub>2</sub>.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m<sub>2</sub>.c.m_flow.max &lt;= 0 for all j = 1:N and c.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then s = (sum(max(-m_1,c.m.flow, 0) for k in cat(1,1:q-1, q+1:M))); s um(positiveMax(-m_1,c.m_flow, a)) * inStream(c_n,h_outflow))/ (sum(max(c_n,m_flow, c,m_flow, m_1) * and flow, a) * m_cth.outflow) + sum(positiveMax(c,m_flow, m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(mositiveMax(c,m_flow, m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_flow, m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_flow, m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_n,flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_n,flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_n,flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_n,flow, m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_n,flow,m_n) * inStream(c_n,h_outflow))/ (sum(positiveMax(c,m_n</pre>	// Additional e	quation to be generated
<pre>N = 0, N = 2: // Additional equation to be generated clh_outflow = inStream(clh_outflow); c2h_outflow = inStream(clh_outflow); c2h_outflow = inStream(clh_outflow); c1m_otflow.min &gt;= 0 for all j = 1:N with j &lt;&gt; i and then c<sub>1</sub>m_flow.max &lt;= 0 for all k = 1:N then inStream(match_outflow) = m_i.c.h_outflow; else si = sum(max(c_n.c.m_flow,o) for j in cat(l,l:i-1, i+1:N) + sum(max(c_n.m_flow, o) for k in 1:N); inStream(ma,c.h_outflow) = (sum(positiveMax(c_n.m_flow,o)) *inStream(c_h_outflow)))/ (sum(positiveMax(c_n.m_flow,o)) *inStream(c_h_outflow)))/ (sum(positiveMax(c_n.m_flow,o))) sum(positiveMax(c_n.m_flow,o))) for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for k in 1:N and c_n.m_flow.min &gt;= 0 for all j = 1:N and c_n.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_n.l_outflow = 0; else s_q = (Sum(max(cm_j.c.m.flow,0)) for j in 1:N) + sum(mositiveMax(c_m,c.m.flow,m) * an(ch,outflow)))/ (sum(max(c_n.m_flow,m) for j in 1:N) + sum(max(c_n.m_flow,m) for j in 1:N) +</pre>	c1.h_outflow = n	n1.c.h_outflow;
<pre>// Additional equation to be generated cl.h.outflow = instream(cl.h.outflow); c2.h_outflow = instream(cl.h_outflow); c3.h_outflow = instream(cl.h_outflow); c3.h_outflow.max &lt; 0 for all j = 1:N with j ⇔ i and c.m_flow.max &lt; 0 for all k = 1:M then si = sum(max(m_n.c.m_flow,0) for j in cat(1,1:i-1, i+1:N) + sum(max(c,m_flow,0) for k in cat(1,1:i-1, i+1:N) + sum(max(c,m_flow,0) for k in 1:N); instream(m_n.c.m_outflow) = m_i.c.h_outflow) + sum(positiveMax(cm_n.c.m_flow,s_1)*m_i.c.h_outflow))// (uum(positiveMax(cm_n.c.m_flow,s_1))* sum(positiveMax(cm_n.c.m_flow,s_1)) + sum(positiveMax(cm_n.c.m_flow,s_1)) + sum(positiveMax(cm_n.c.m_flow,s_1)) + sum(positiveMax(cm_n.c.m_flow,s_1)) + sum(positiveMax(cm_n.c.m_flow,s_1)); for j in 1:N and c_{a.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m_j.c.m_flow.min &gt;= 0 for all j = 1:N and c_{a.m_flow.max &lt;= 0 for all k = 1:M and k ⇔ q then c.m_flow.max &lt;= 0 for all j = 1:N and c_{a.m_flow.max &lt;= 0 fo</pre>	N = 0, M = 2:	
<pre>cl.h_outflow = inStream(c2.h_outflow); c2.h_outflow = inStream(c1.h_outflow); All other cases: if m,c.m_flow.max &lt;= 0 for all j = 1:N with j &lt;&gt; i and</pre>	// Additional e	quation to be generated
<pre>c2.h_outflow = inStream(c1.h_outflow); All other cases: if m; c.m_flow.min &gt;= 0 for all j = 1:N with j ⇔ i and c, m_flow.max &lt;= 0 for all k = 1:M then Stream(m,c.h_outflow) = m; c.h_outflow; else is = sum(max(-m,c.m_flow,0) for j in cat(1,1:i-1, i+1:N) + sum(max(-c,.m_flow,0) for j in cat(1,1:i-1, i+1:N) + sum(max(-c,.m_flow,0) for k in 1:N); inStream(m,-c.h_outflow) = (sum(positiveMax(-m,c.m_flow,s)) *inStream(c,.h_outflow)))/ (sum(positiveMax(-m,c.m_flow,s)) *inStream(c,.h_outflow)))/ (sum(positiveMax(-m,c.m_flow,s)) * sum(cositiveMax(-m,c.m_flow,s)) * for j in 1:N and i ⇔ j and m; c.m.flow.min &lt; 0, for j in 1:N and c,.m_flow.max &gt;= 0 for all j = 1:N and c,.m_flow.min &gt;= 0 for all j = 1:N and c,.m_flow.max &lt;= 0 for all j = 1:N and c,.m_flow.max &lt;= 0 for all i = 1:N and k ⇔ q then .c_1h_outflow = 0; else sq = (sum(max(-m,c.m.flow,0) for j in 1:N) + sum(mox(c,.m_flow,.m) for k in cat(1,1:q-1, q+1:N))); c_q.h_outflow = (sum(positiveMax(-m,c.m.flow,s)) * inStream(c,h_outflow))/ (sum(positiveMax(-m,c.m.flow,s)) * inStream(c,h_outflow)))/ (sum(positiveMax(-m,c.m.flow,s)) * inStream(c,h_outflow)))/ (sum(positiveMax(-m,c.m.flow,s)) * inStream(c,h_outflow))/ (sum(positiveMax(-m,c.m.flow,s)) * inStream(</pre>	c1.h_outflow =	inStream(c2.h_outflow);
<pre>All other cases: if m,c.m_flow.max &lt;= 0 for all j = 1:N with j &lt;&gt; i and</pre>	c2.h_outflow =	inStream(c1.h_outflow);
<pre>int characterized by the set of the se</pre>	All other cases	
<pre>then</pre>	if m.c.m flow.	nin >= 0 for all i = 1:N with i <> i and
<pre>then ' instream(m_i.c.h_outflow) = m_i.c.h_outflow; else instream(m_i.c.h_outflow) = m_i.c.h_outflow; else si = sum(max(-m_i.c.m_flow,o) for j in cat(1,1:i-1, i+1:N) + sum(max(o_c.m_flow,o) for k in 1:N); instream(m_i.c.h_outflow) = (sum(positiveMax(-m_i.c.m_flow,a_i)*m,c.h_outflow) + sum(positiveMax(-m_i.c.m_flow,a_i)) + sum(positiveMax(-m_i.c.m_flow,a_i)); sum(positiveMax(-m_i.c.m_flow,a_i)) + isum(positiveMax(-m_i.c.m_flow,a_i)); for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j k in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j k in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m_i.c.m_flow.max &lt;= 0 for all j = 1:N and</pre>	ck.m flow.	$nax \le 0$ for all $k = 1:M$
<pre>instream(m_c.c.h_outflow) = m_t.c.h_outflow; else si = sum(max(-m_m,c.m_flow,0) for j in cat(1,1:i-1, i+1:N) + sum(max(-c_t,m_flow,0) for j in cat(1,1:i-1, i+1:N) + sum(nositiveMax(-m_t,c.m_flow,s) + instream(c_t,h_outflow) + sum(positiveMax(-m_t,m_tlow,s) + instream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_tlow,s)) + sum(positiveMax(-m_t,m_tlow,s))) + sum(positiveMax(-m_t,m_tlow,s))) (// Additional equations to be generated for q in 1:N loop if m_t.c.m_flow.min &gt;= 0 for all j = 1:N and c_t,m_flow.min &gt;= 0 for all j = 1:N and c_t,m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_t,h_outflow = 0; else s_ = (sum(max(-m_t,c.m_flow,0) for j in 1:N) + sum(positiveMax(-m_t,c.m_flow,s), * inStream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_flow,s), * inStream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_flow,s), * inStream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_flow,s), * inStream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_flow,s), * inStream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_flow,s), * inStream(c_t,h_outflow)))/ (sum(positiveMax(-m_t,m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_t,m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_t,m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_t,m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_tlow,s), * inStream(c_t,h_tlow)))/ (sum(positiveMax(-m_tlow)))/ (sum(max(-m_tlow,s)) * instream(c_t,h</pre>	then	
<pre>else</pre>	inStream(mi.c	.h outflow) = m_i.c.h outflow;
<pre>si = sum(max(-m,c.m.flow,0) for j in cat(1,1:i-1, i+1:N) + sum(max( c<sub>k</sub>, m.flow, 0) for j in (at(1,1:i-1, i+1:N) + sum(positiveMax(c<sub>k</sub>, m.flow,s<sub>1</sub>) *m,c.h.outflow) + sum(positiveMax(c<sub>k</sub>, m.flow,s<sub>1</sub>) *m,c.h.outflow)))/ (sum(positiveMax(c<sub>k</sub>, m.flow,s<sub>1</sub>)) + sum(positiveMax(c<sub>k</sub>, m.flow,s<sub>1</sub>)))</pre>	else	
<pre>sum(max( c<sub>k</sub>.m_flow ,0) for k in 1:M); inStream (m,.ch_outflow) = (sum(positiveMax(-m,.c.m_flow,s_1)*m,.ch_outflow) + sum(positiveMax(-m,.c.m_flow,s_1)*m,ch_outflow)))/ (sum(positiveMax(-m,.c.m_flow,s_1)) + sum(positiveMax(-m,.c.m_flow,s_1)) + for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0, for k in 1:N and i &lt;&gt; j and m_j.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N loop if m_j.c.m_flow.max &lt;= 0 for all j = 1:N and c_k.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c_k.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then sum(max( c_k.m_flow, 0) for j in 1:N) + sum(max(c_k.m_flow, 0) for j in 1:N) + sum(positiveMax(c_k.m_flow, m_k, c.m_flow, b, m_k, c.h, outflow) + (sum(positiveMax(c_k.m_flow, m_k) * inStream(c_k.h_outflow)))/ (sum(positiveMax(c_k.m_flow, m_k) * inStream(c_k.h_outflow)))/ (sum(positiveMax(c_k.m_flow, m_k) * inStream(c_k.h_outflow)))/ (sum(positiveMax(c_k.m_k flow, m_k) * inStream(c_k.h_outflow)))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.m_k) * inStream(c_k.h_outflow))/ (sum(positiveMax(c_k.m_k) * inStream(c_k.m_k) * inStream(c_k.m_k) * inStream(c_k.m_k) * inStream(c_k.m_k) *</pre>	si = sum(max(	-mj.c.m_flow,0) for j in cat(1,1:i-1, i+1:N) +
<pre>instream(m.c.m_outflow) = iw,si,'m,.c.h_outflow) + sum(positiveMax(c_k.m_flow,si,' 'in,.c.h_outflow)))/ (sum(positiveMax(c_k.m_flow,si,'))+ sum(positiveMax(c_k.m_flow,si,'))+ sum(positiveMax(c_k.m_flow,si,'))+ for a sint sint sint sint sint sint sint sint</pre>	sum (max (	c <sub>k</sub> .m_flow ,0) for k in 1:M);
<pre>(sum (positiveMax(-m,.c.m_flow,a)'m,.c.m_outflow) + sum (positiveMax(-m,.flow,a))'m,.c.m_outflow))/ (sum (positiveMax(-m,.flow,a)) + sum (positiveMax(-m_,.fow,flow,a)) + sum (positiveMax(-m_,.fow,flow,a)) + for j in 1:N and i &lt;&gt; j and mj.c.m_flow.min &lt; 0, for c in 1:N and i &lt;&gt; j and mj.c.m_flow.min &lt; 0, for c in 1:N and i &lt;&gt; j and mj.c.m_flow.max &gt; 0 // Additional equations to be generated for q in 1:N boop if mj.c.m_flow.min &gt;= 0 for all j = 1:N and c.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then c.m_flow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then sum (max( c.m_flow, 0) for j in 1:N) + sum (max( c.m_flow, 0) for j in 1:N) + c.h.outflow = (sum [positiveMax(-m,.c.m_flow,a)] * inStream (c.h.outflow))/ (sum (positiveMax(-m, flow, m_flow,a)) * inStream (c.h.outflow))/ (sum (positiveMax(-m, flow, m_flow,a)) * inStream (c.h.outflow))/</pre>	inStream(mi.c	h_outflow) =
<pre>sum(positiveMax(c_k,m_riow,si) ~ inStream(c_k,n_outflow)))/ (sum(positiveMax(c_k,m_riow,si)) + sum(positiveMax(c_k,m_riow,si))) for j in 1:N and i &lt;&gt; j and m_j.c.m_flow.min &lt; 0,</pre>	(sum (positiv	emax(-m,.c.m_flow,si)*m,.c.h_outflow) +
<pre>(usun (Dollivemax ("m., c.m. flow, n)) / ' sum (positivemax (c.m. flow, n)) / ' for j in 1:N and i &lt;&gt; j and m., c.m. flow.min &lt; 0,     for q in 1:N loop     if m., c.m. flow.min &gt;= 0 for all j = 1:N and         c., m_flow.mix &lt;= 0 for all j = 1:N and         c., m_flow.mix &lt;= 0 for all k = 1:N and k &lt;&gt; q         th         c., n_flow.max &lt;= 0 for all j = 1:N and         c., n_flow.mix &lt;= 0 for all j = 1:N and         c., n_flow.mix &lt;= 0 for all j = 1:N and         c., n_flow.max &lt;= 0 for all i = 1:N and k &lt;&gt; q         th         c., n_flow.max &lt;= 0 for all j = 1:N and         c., n_flow.max &lt;= 0 for all i = 1:N and k &lt;&gt; q         th         c., n_flow.max &lt;= 0 for j in 1:N +         sum (max(c.m., c.m. flow, 0) for j in 1:N) +         sum (max(c.m., c.m. flow, 0) for j in 1:N) +         sum (max(c.m., c.m. flow, 0) for j in 1:N) +         sum (max(c.m. flow, m, flow, m, j) * m.c.h.outflow +         sum (positiveMax(c.m., c.m. flow, m, j) * inStream(c., h_outflow)))/         (sum (mositiveMax(m., m. flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream(c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream (c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream (c., h_outflow)))/         (sum (positiveMax(m., m, flow, m, j) * inStream (c., h_outflow)))/         (sum (positiveMax(m., m, flow))) /         (sum (positiveMax(m, flow))) /         (sum</pre>	sum (positiv	eMax(C <sub>k</sub> .m_IIOW,S <sub>1</sub> ) ^LINSUPERIM(C <sub>k</sub> .n_OUTIIOW)))/
$\label{eq:construction} \begin{tabular}{lllllllllllllllllllllllllllllllllll$	sum (positiv	eMax(c, m flow s.)))
<pre>for k in 1:M and c<sub>k</sub>,m_flow.max &gt; 0 // Additional equations to be generated for q in 1:M loop if m<sub>1</sub>.c.m_flow.min &gt;= 0 for all j = 1:N and</pre>	oum (poorer	for i in 1:N and i <> i and m.c.m flow.min < 0.
<pre>// Additional equations to be generated for q in 1:M loop if m,c.m_flow.min &gt;= 0 for all j = 1:N and</pre>		for k in 1:M and ck.m_flow.max > 0
<pre>for q in 1:M loop if m, c.m_flow.min &gt;= 0 for all j = 1:N and</pre>	// Additional e	quations to be generated
<pre>if m_n.c.m_flow_min &gt;= 0 for all j = 1:N and</pre>	for g in 1:M lo	OD
<pre>`c_k.m_filow.max &lt;= 0 for all k = 1:M and k &lt;&gt; q then     cq.h_outflow = 0; else sq = (sum(max(-m_j.c.m_flow,0) for j in 1:N) +     sum(max(-c_k.m_flow, 0) for k in cat(1,1:q-1, q+1:M))); cq.h_outflow = (sum(positiveMax(m_j.c.m_flow,sq) * m,c.h.outflow) +     sum(positiveMax(c_k.m_flow,sq) * inStream(c_k.h_outflow)))/     (gun (positiveMax(m_j.c.m_flow,sq) * inStream(c_k.h_outflow)))/     (gun (positiveMax(m_j.c.m_flow,sq) * inStream(c_k.h_outflow)))/     (gun (positiveMax(m_j.c.m_flow,sq) * inStream(c_k.h_outflow)))/     (gun (positiveMax(m_j.c.m_flow,sq) * inStream(c_k.h_outflow)))/ </pre>	if ma.c.m flor	$y.min \ge 0$ for all $j = 1:N$ and
<pre>then _ c_h outflow = 0; else _ c_h outflow = 0; else _ c_h outflow = 0; else _ c_h outflow = 0; sq = (sum(max(-c_n,c.m.flow,0) for j in 1:N) + sum(max(-c_n,m.flow,0) for k in cat(1,1;q-1, q+1:M))); c_q.h_outflow = (sum(positiveMax(-c_n,c.m.flow,s_q) * inStream(c_k.h_outflow)))/ (sum(positiveMax(-c_n,c.m.flow,s_q) * inStream(c_k.h_outflow)))/ </pre>	ck.m flo	x.max <= 0 for all $k = 1:M$ and $k <> q$
<pre>c_1,h_outflow = 0; else s_q = (sum(max(-m_j.c.m_flow,0) for j in 1:N) +</pre>	then	
<pre>else     aq = (sum(max(-m_j.c.m_flow,0) for j in 1:N) +     sum(max(-c_m.m_flow ,0) for k in cat(1,1:q-1, q+1:M)));     cq.h_outflow = (sum(positiveMax(-m_j.c.m_flow,sq) * m, c.h.outflow) +         sum(positiveMax(-m_j.c.m_flow,sq) * inStream(c_m.h_outflow)))/         (sum(positiveMax(-m_j.c.m_flow,sq) * inStream(c_m.h_outflow)))) </pre>	c <sub>q</sub> .h_outfl	w = 0;
<pre>sq = (sum(max(-m,c.m_tlow,0) for j ln l:N) + sum(max(-c_k,m_flow, 0) for k in cat(1,1:q-1, q+1:M))); cq.h_outflow = (sum(positiveMax(-m,c.m_flow,sq)*m,c.h_outflow) + sum(positiveMax(c_k,m_flow,sq) * inStream(c_k,h_outflow)))/ (sum(opsitiveMax(m,c,m_flow,cat) + inStream(c_k,h_outflow)))/</pre>	else	and the second
<pre>sum(max(c<sub>k</sub>,m_tLow , 0) Tor k ln cat(1,1:q-1, q+1:M))); c<sub>q</sub>,h_outflow = (sum(positiveMax(m,c,m, flow,ga)*m,c.h_outflow) + sum(positiveMax(m,c,m,flow,ga) * inStream(c<sub>k</sub>,h_outflow)))/ (sum(positiveMax(m,c,m,flow,ga) * inStream(c<sub>k</sub>,h_outflow)))/</pre>	$s_q = (sum)$	<pre>nax(-mj.c.m_flow,0) tor j in 1:N) +</pre>
<pre>cq.n_outriow = (sum(positivemax(~m,.c.m_irow_sq)~m,.c.n_outriow) +</pre>	Sum (	<pre>max( ck.m ilow , U) TOF K in cat(1,1:q-1, q+1:M))); w = (sum/positiveMax(-m c m flow c)) t m c h outflow);</pre>
(sum(positiveMax(-m,.c.m flow,s_)) +	cq.n_outin	<pre>cum(positiveMax(-mj.c.m_iiow,Sq)* mj.c.n_outriow) + cum(nositiveMax(c = flow s) + inStream(c = outflow)))/</pre>
I DOWNING DECEMBER I AND COMPANY AND TO DECEMBER IN T		(sum/positiveMax/-m_c_m_flow_c_)) +

### 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?

 $file:///C:/doc/modprod\%20 center/modprod\%20 workshop\%20 Feb\%202013/Presentati... \ 2013-02-04$ 

## 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:

### **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.



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

# Evaluate relations like records

```
model Switch
 1
2
     parameter Boolean open;
 3
     OnePort lhs;
 4
     OnePort rhs;
 5
     Ground ground;
 6
     equation
7
     connect(lhs.p, if open then grour
8
     connect(rhs.n, if open then grour
 9 end Switch;
10
```

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

### **Equivalent rewrite**

```
model Switch
 1
 2
     parameter Boolean open;
 3
     OnePort lhs;
 4
     OnePort rhs;
 5
 6
     Ground ground;
 7
     equation
     if open then
 8
       connect(lhs.p, rhs.n);
 9
10
       connect(rhs.n, lhs.p);
     else
11
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
 1
 2
     StepVoltage source;
 3
     Resistor R1(R=10);
 4
     Capacitor C1(C=5);
 5
6
     Inductor I1(I=5);
     Ground ground;
 7
     equation
 8
     connect(source.p, R1.n);
 9
     connect(R1.p, C1.n);
     connect(C1.p, I1.n);
10
     connect(I1.p, source.n);
11
12
     connect(source.n, ground.p);
13
   end SimpleCircuit1;
14
```

### 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
     end connected sets;
16
17
```

(Using External Java functions)

## **Calculating Equations**

```
/* JPotentialConnectRelation
 1
                                     is t
 2
3
        this needs to come out of an a
     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...

Model Sin	npleCircuit1 Inspect	Documentation Source Con	npiled instantiate Simulate		
Fields					NO CO
Access	Causality	Variability	Туре	Name	Defined in
public			inductor	11	SimpleCircuit1
public			Capacitor	C1	SimpleCircuit1
public			Ground	ground	SimpleCircuit1
public			Resistor	R1	SimpleCircutt
public			StepVoltage	source	SimpleCircuit1
public			StepVoltage	source	SimpleCircutt

# Calculating Equations differently

```
/* JPotentialConnectRelation is t
 1
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
         set[i+1] = (2 - 1) * set[i];
8
 9
       end for;
     end for;
10
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 \* .l1.n.u) + (1,000000 \* .C1.p.u) = 0,000000
- (-1,000000 \* .l1.p.u) + (1,000000 \* .ground.p.u) = 0,000000
- (-1,000000 \* .ground.p.u) + (1,000000 \* .source.n.u) = 0,000000

• ...

#### Future

 $file:///C:/doc/modprod\%20 center/modprod\%20 workshop\%20 Feb\%202013/Presentati... \ 2013-02-04$ 

# 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.