Infrastructure for DEVS Modelling and Experiment

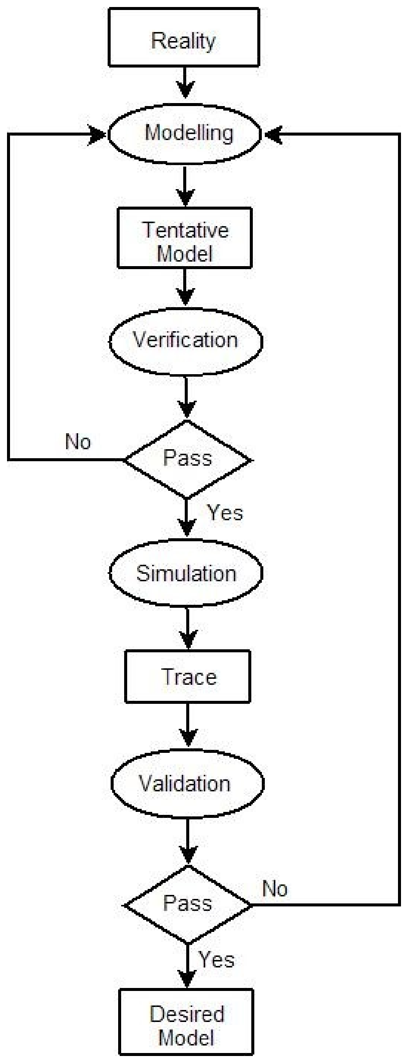
Hongyan Song
August 2006

Modelling, Simulation, and Design Lab
School of Computer Science
McGill University

Motivations and Purposes

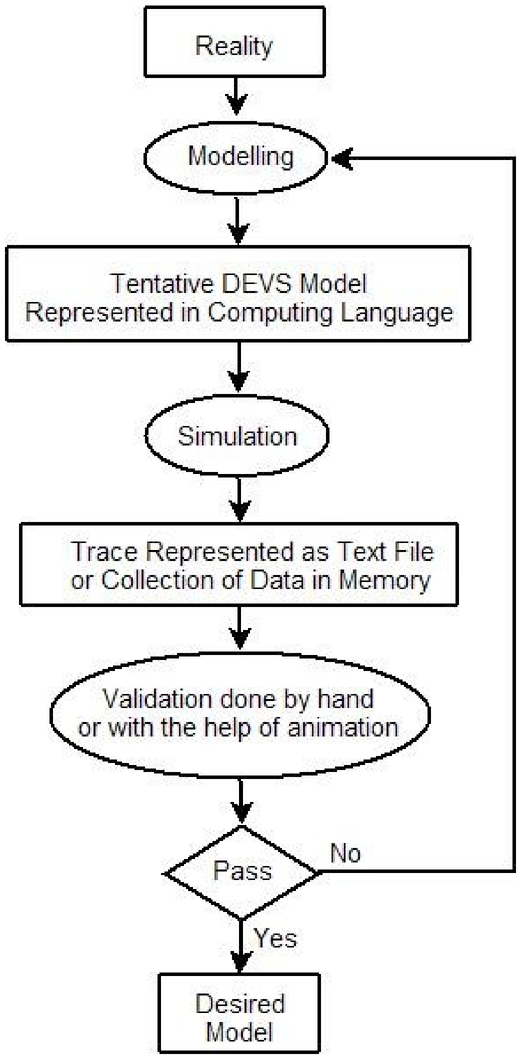
- Facilitate the Process of DEVS Modelling
- Promote DEVS Standardization and Application

The Modelling Process



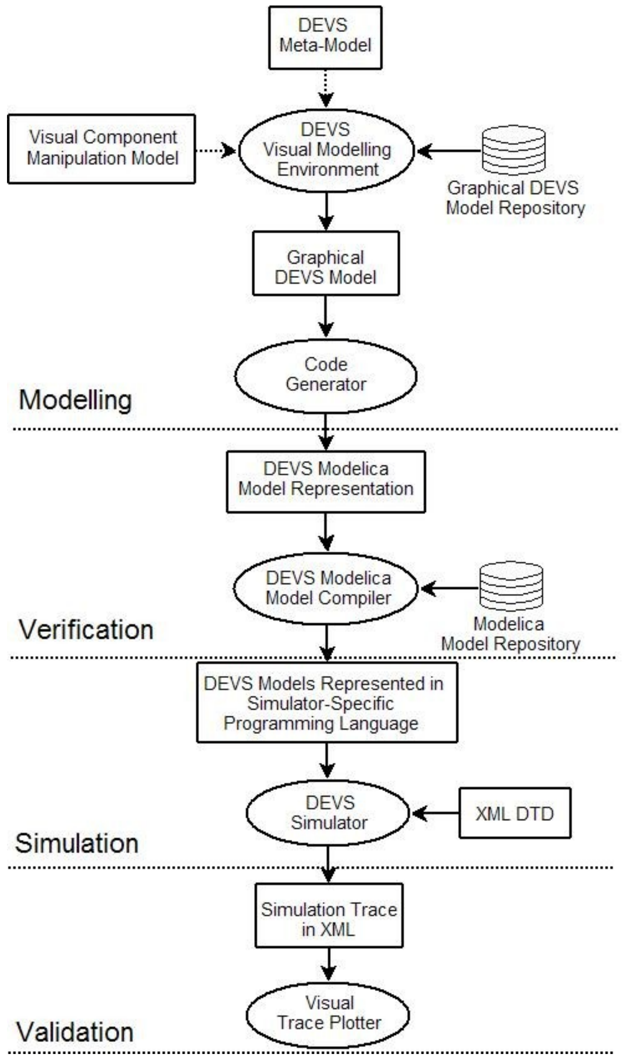
- Modelling
  - Build a tentative model
- Verification
  - Make sure a model is syntatically correct to a specific formalism
- Simulation
  - Execute the model with experimental data to generate the model behaviour
- Validation
  - Make sure the model is semantics correct

Current DEVS Modelling Process


Reality
Modelling
Tentative DEVS Model Represented in Computing Language
Simulation
Trace Represented as Text File or Collection of Data in Memory
Validation done by hand or with the help of animation
Pass
No
Yes
Desired Model

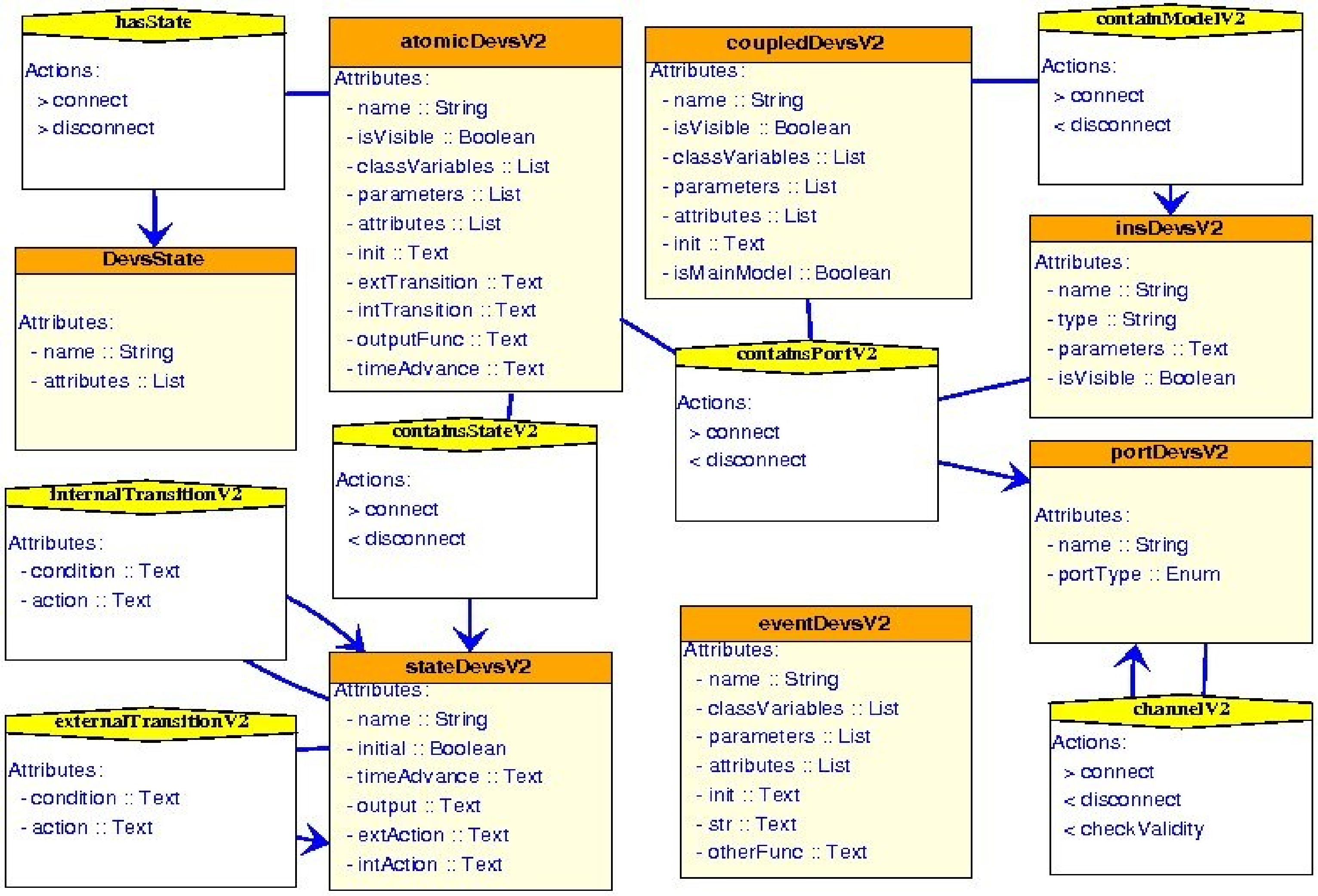
- No visual modelling tools
- Models represented in programming languages
- No automatic model verification tools
- Simulation trace is in memory or in text file
- No specific tool for DEVS trace plotting

The Architecture of The Infrastructure

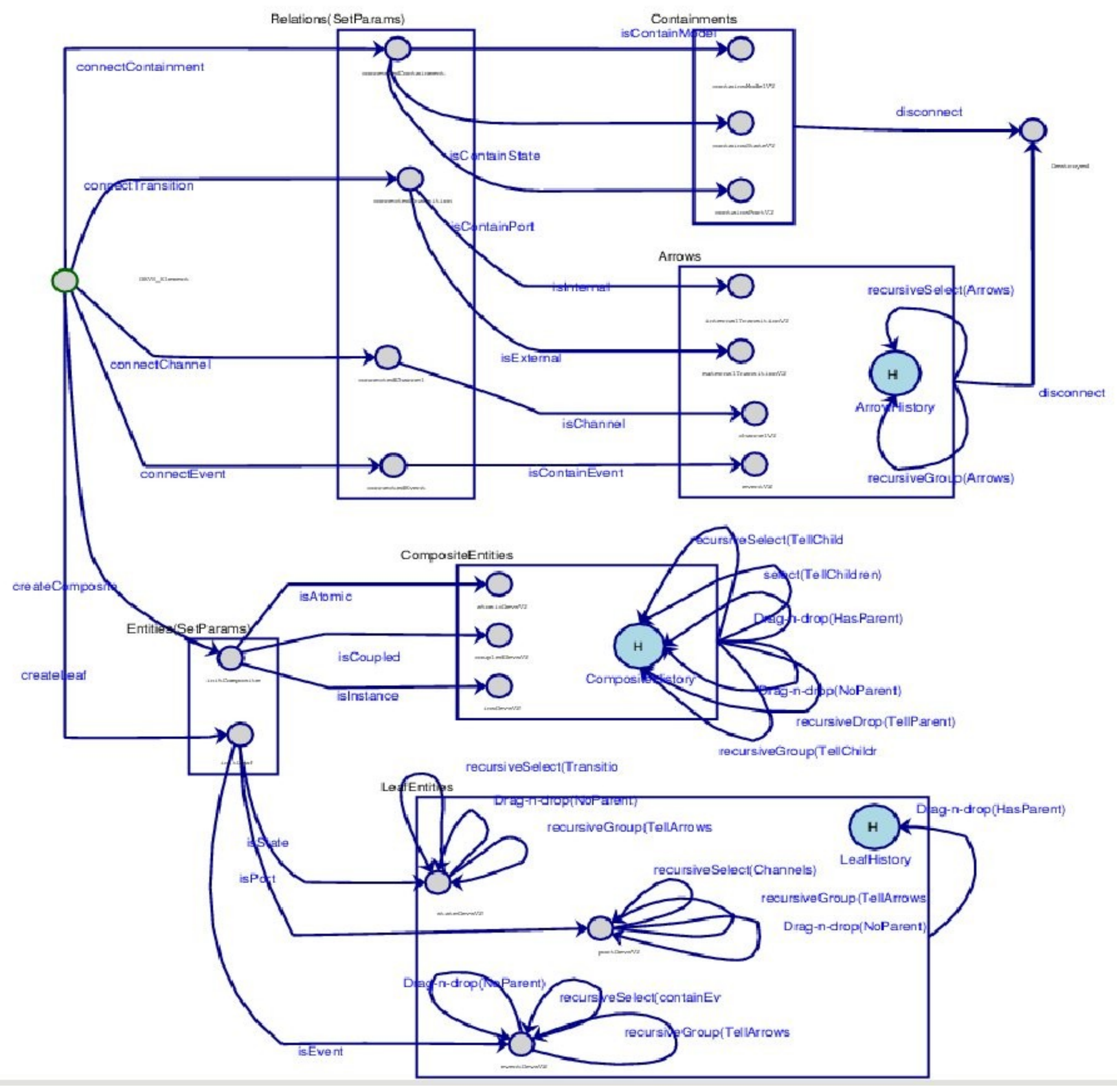

DEVS Meta-Model
Visual Component Manipulation Model
DEVS Visual Modelling Environment
Graphical DEVS Model Repository
Graphical DEVS Model
Code Generator
Modelling
DEVS Modelica Model Representation
DEVS Modelica Model Compiler
Modelica Model Repository
Verification
DEVS Models Represented in Simulator-Specific Programming Language
DEVS Simulator
XML DTD
Simulation
Simulation Trace in XML
Visual Trace Plotter
Validation

- Modelling
  - Visual modelling environment
  - Graphical models to models represented in modelling language
- Verification
  - Model compiler
  - Check syntax of modelling language, modelling formalism and generate programming language specific models
- Simulation
  - Standardized XML trace
  - Trace from one simulator plotted by different tools, one tool used by many simulators
- Validation
  - Visual trace plotter specific for DEVS

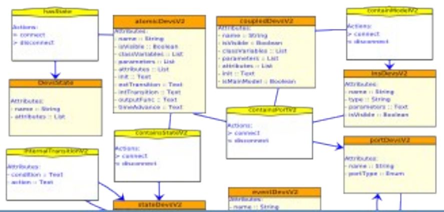
DEVS Meta-Model

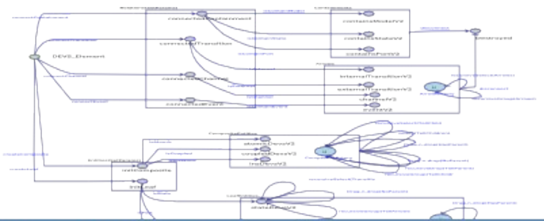


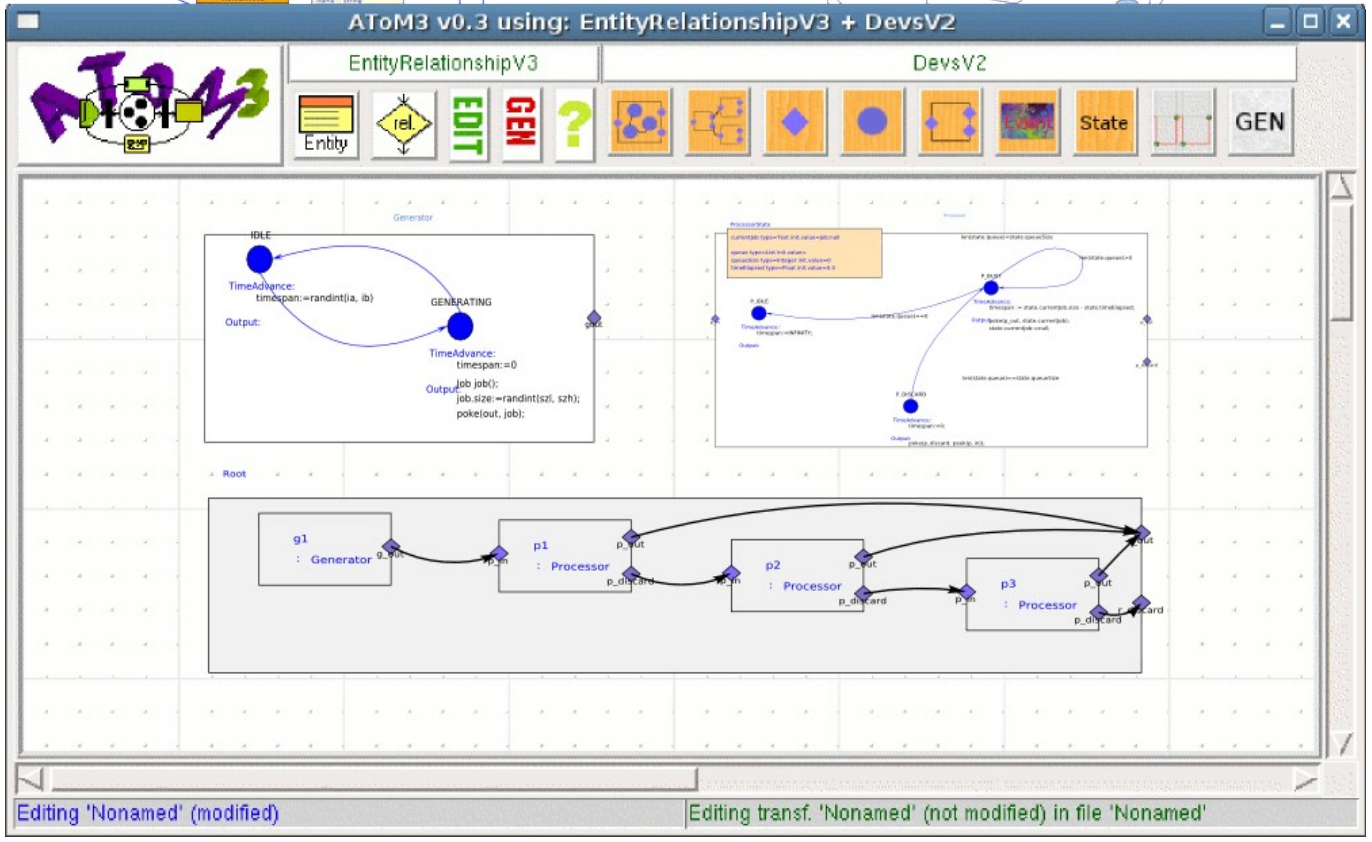
GUI Model of the Visual Environment



Visual Modelling Environment and Visual Models







AToM3 v0.3 using: EntityRelationshipV3 + DevsV2

Editing 'Nonamed' (modified)

Editing transf. 'Nonamed' (not modified) in file 'Nonamed'

Modelica Model Representation

```
class GeneratorState

  Generator.SeqStates
    seqState(start=Generator.SeqStates.G_IDEL);

end GeneratorState;


class Generator

  extends AtomicDEVS;

  parameter Integer ia=0;

  parameter Integer ib=0;

  parameter Integer szl=0;

  parameter Integer szh=0;

  parameter String name="a";

  output DevsPort g_out;

  GeneratorState state();

  type SeqStates = enumeration(G_IDLE,
    G_GENERATING);
```

```
function intTransition

 algorithm

   if( state.seqState==SeqStates.G_IDLE ) then

     state.seqState := SeqStates.G_GENERATING;

   elseif(state.seqState==SeqStates.G_GENERATING)
    then

      state.seqState := SeqStates.G_IDLE;

   end if;

 end intTransition;

 function outputFnc

     ......

 end outputFnc;

 function timeAdvance

  ......

 end timeAdvance;

end Generator;
```

Python DEVS Representation - 1

```
class GeneratorState:

  def __init__(self):

    self.seqState = Generator.G_IDLE


  def __str__(self):

    strRep = ''

    strRep = strRep + "\nseqState: " +
   str(self.seqState)

    return strRep
```

```
def toXML(self):

  strRep = ''

  strRep = strRep + "\n<attribute
 category=\"P\">"

  strRep = strRep +
 "\n\t<name>seqState</name>"

  strRep = strRep +
 "\n\t<type>Generator.SeqStates</type>"

  strRep = strRep +
 "\n\t<value>"+str(self.seqState)+"</value>"

  strRep = strRep + "\n</attribute>"

  return strRep
```

Python DEVS Representation - 2

```
class Generator( AtomicDEVS ):

  G_IDLE = 'G_IDEL'

  G_GENERATING = 'G_GENERATING'

  def __init__(self, ia, ib, szl, szh, name):

    AtomicDEVS.__init__(self, name)

    self.ia = ia

    self.ib = ib

    self.szl = szl

    self.szh = szh

    self.name = name

    self.g_out = self.addOutPort("g_out")

    self.state = GeneratorState()
```

```
def intTransition( self ):

  if(self.state.seqState ==
 Generator.G_IDLE):

    self.state.seqState =
 Generator.G_GENERATING

  elif(self.state.seqState ==
 Generator.G_GENERATING):

    self.state.seqState = Generator.G_IDLE

  return self.state

def outputFnc( self ):

  evt = None

  if(self.state.seqState ==
 Generator.G_GENERATING):

    evt = Job(self.szl, self.szh)

    self.poke(self.g_out, evt)

def timeAdvance( self ):

  .....
```

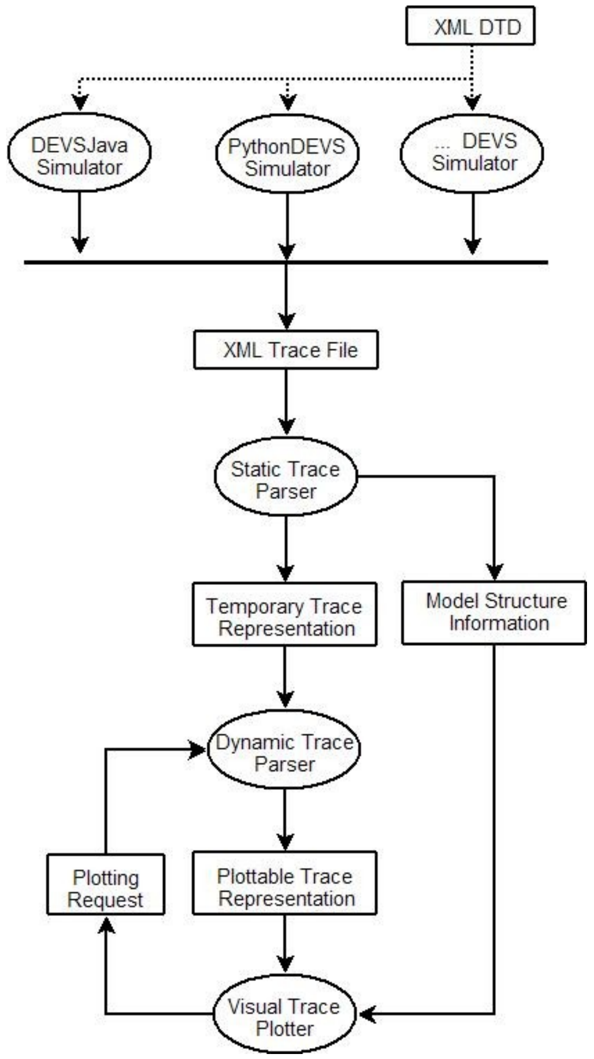
XML DTD for Simulation Trace

1. <!ELEMENT trace (event+)>
2. <!ELEMENT event (model, time, kind, port*, state)>
3. <!ELEMENT model (#PCDATA)>
4. <!ELEMENT time (#PCDATA)>
5. <!ELEMENT kind (IN|EX|#PCDATA)>
6. <!ELEMENT port (message)>
7. <!ELEMENT message (#PCDATA)>
8. <!ELEMENT state (attribute+)>
9. <!ELEMENT attribute (name, type, value+)>
10. <!ELEMENT name (#PCDATA)>
11. <!ELEMENT type (#PCDATA)>
12. <!ELEMENT value (#PCDATA| attribute)*>
13. <!ATTLIST port name CDATA #IMPLIED>
14. <!ATTLIST port category (I|O) #REQUIRED>
15. <!ATTLIST attribute category (P|C | PC | CC) #REQUIRED>

Simulation Trace in XML

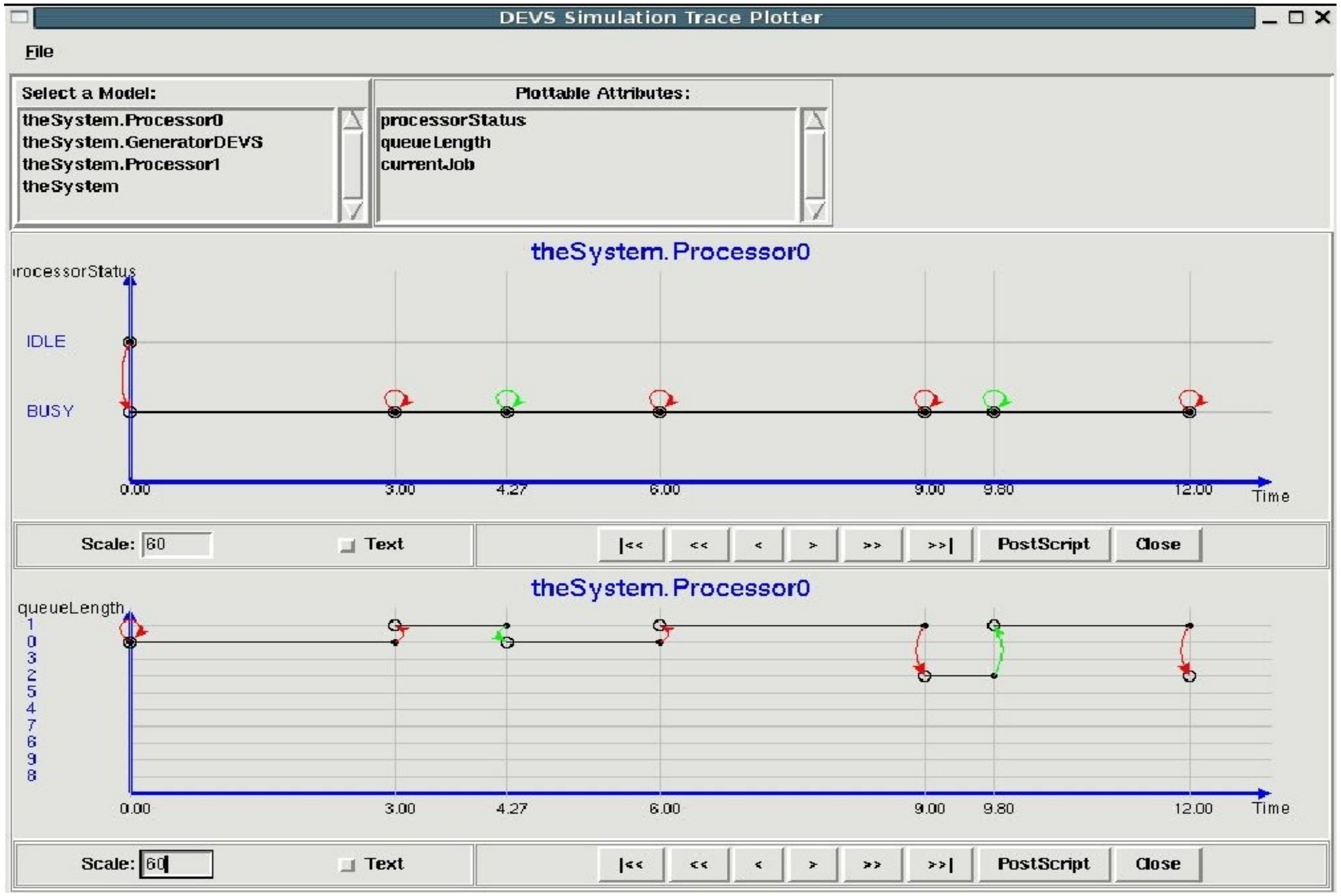
```
<event>
 <model>RootExperiment.p1</model>
 <time>9.0</time>
 <kind>EX</kind>
 <port name="p_in" category="I">
   <message>id: 2 size: 8</message>
 </port>
 <state>
  <attribute category="C">
   <name>currentJob</name>
   <type>Job</type>
   <value>
     <attribute category="P">
       <name>id</name>
       <type>Integer</type>
       <value>1</value>
     </attribute>
     <attribute category="P">
      <name>size</name>
      <type>Integer</type>
      <value>5</value>
     </attribute>
    </value>
   </attribute>
   <Attribute>
    ......
   </Attribute>
   .....
  </state>
 </event>
```

Design of the Trace Plotter



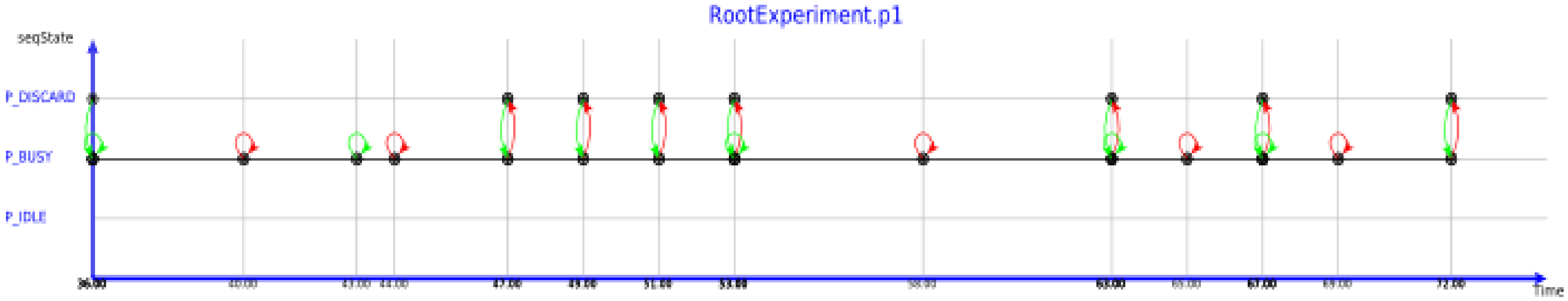
Visual Trace Plotter - 1

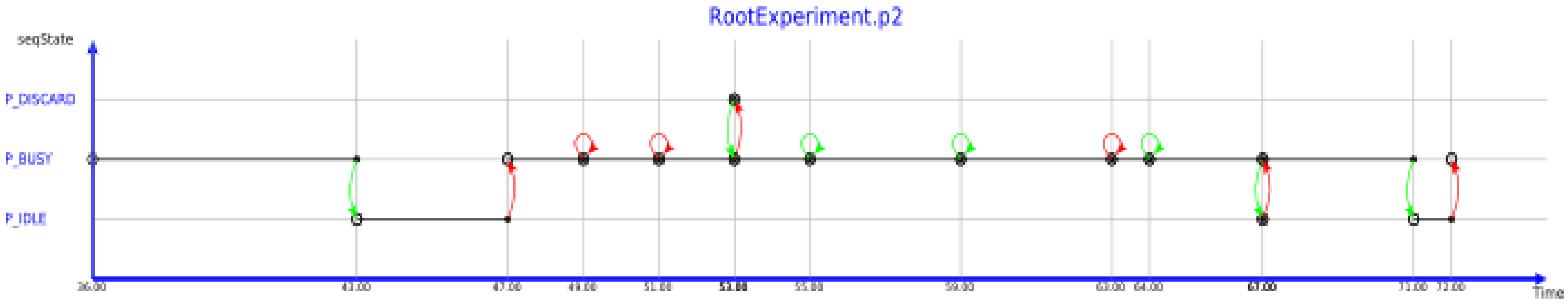
(Different Properties in the Same Model Instance)

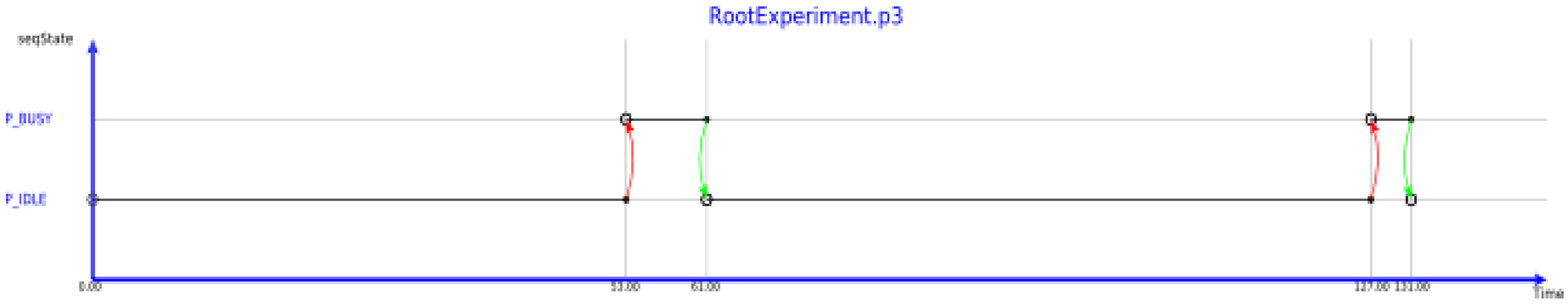


Visual Trace Plotter - 2

(Same Property in Different Model Instances)

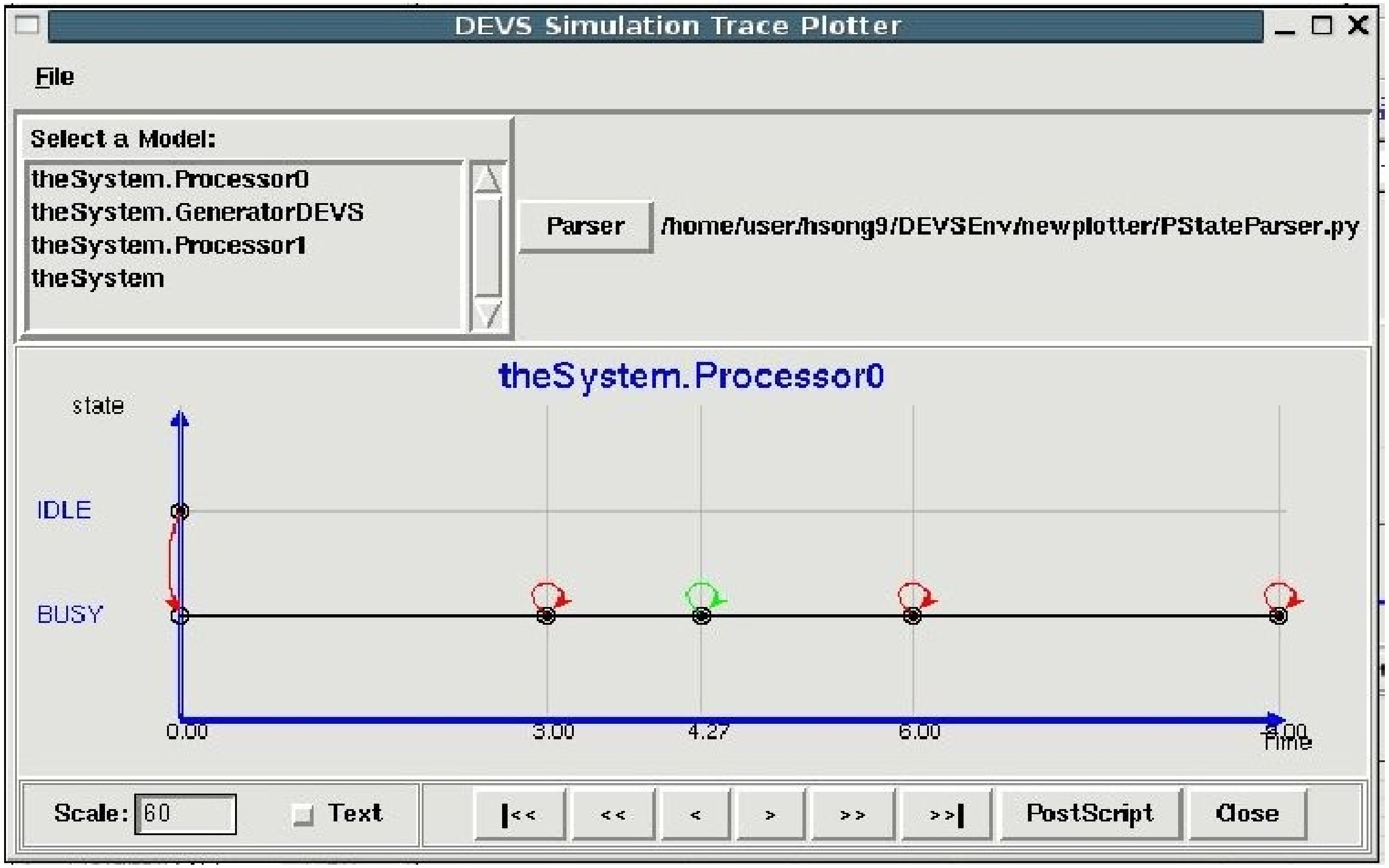






Visual Trace Plotter - 3

(User Customized State Parser)



Acknowledgements

Thanks !

Hans' advice and ideas on design and implementation the system

Denis' original DEVS meta-model and GUI model

Steven's muModelica compiler

Ernesto's help on system maintenance