

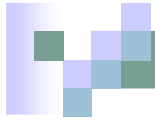
GenGED vs AToM³

Creating a visual DEVS modeling environment

Presented by Denis Dubé

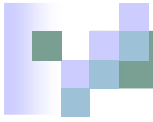
March 24, 2005





Overview

- **Introduction to DEVS**
 - **Why, what, and how**
- Round 1: Basic Diagram Editor
- Round 2: The Visual Modeling Environment
- Round 3: Generating PyDEVS code
- Conclusion and Future Work



What is DEVS?

■ Discrete Event System Specification

■ Highlights:

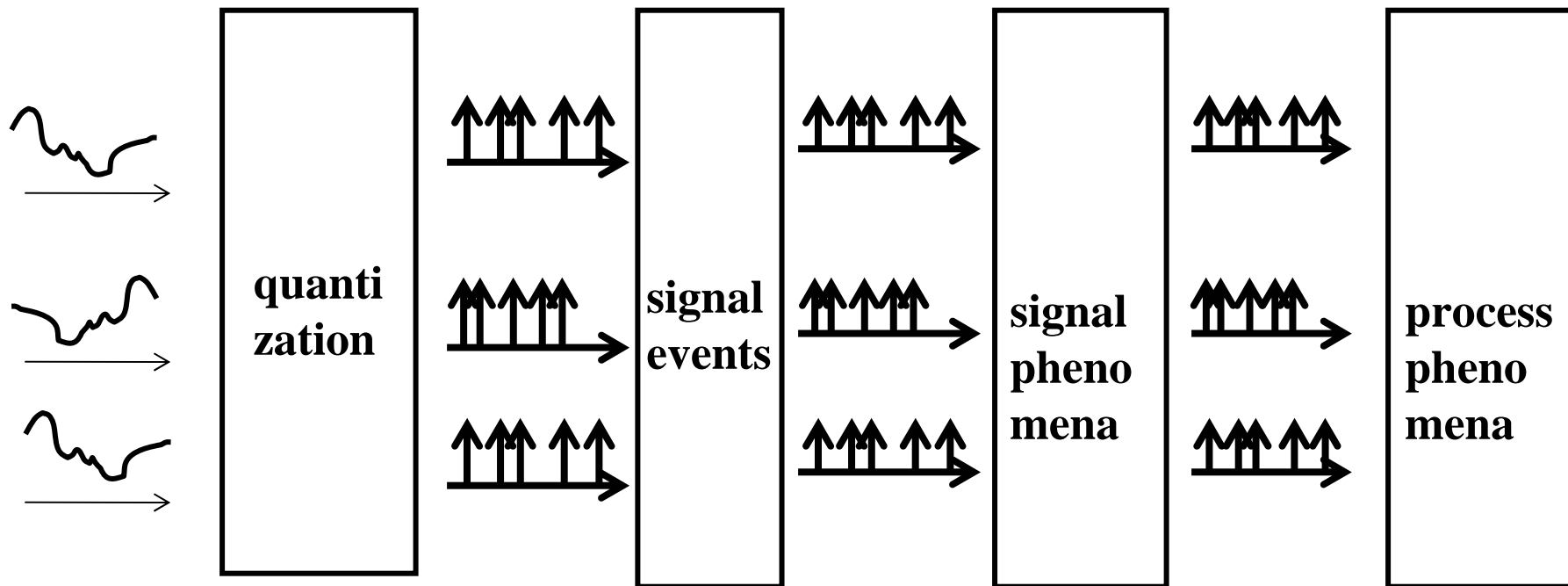
- Based on a **formal modeling and simulation** framework
- Derived from **mathematical dynamical system theory**
- Supports **hierarchical**, modular construction
- Supports discrete event approximation of continuous systems



What Lockheed uses DEVS for

- **Critical Mobile Target**
- **Global Positioning System III**
- **Arsenal Ship**
- **Coast Guard Deep Water**
- **Space Operations Vehicle**
- **Common Aero Vehicle**
- **Joint Composite Tracking Network**
- **Integrated System Center**
- **Space Based Laser**
- **Space Based Discrimination**
- **Missile Defense (Theater / National)**

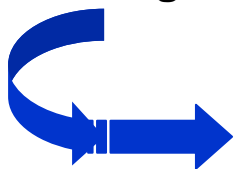
DEVS in control of steel production



Large Scale:

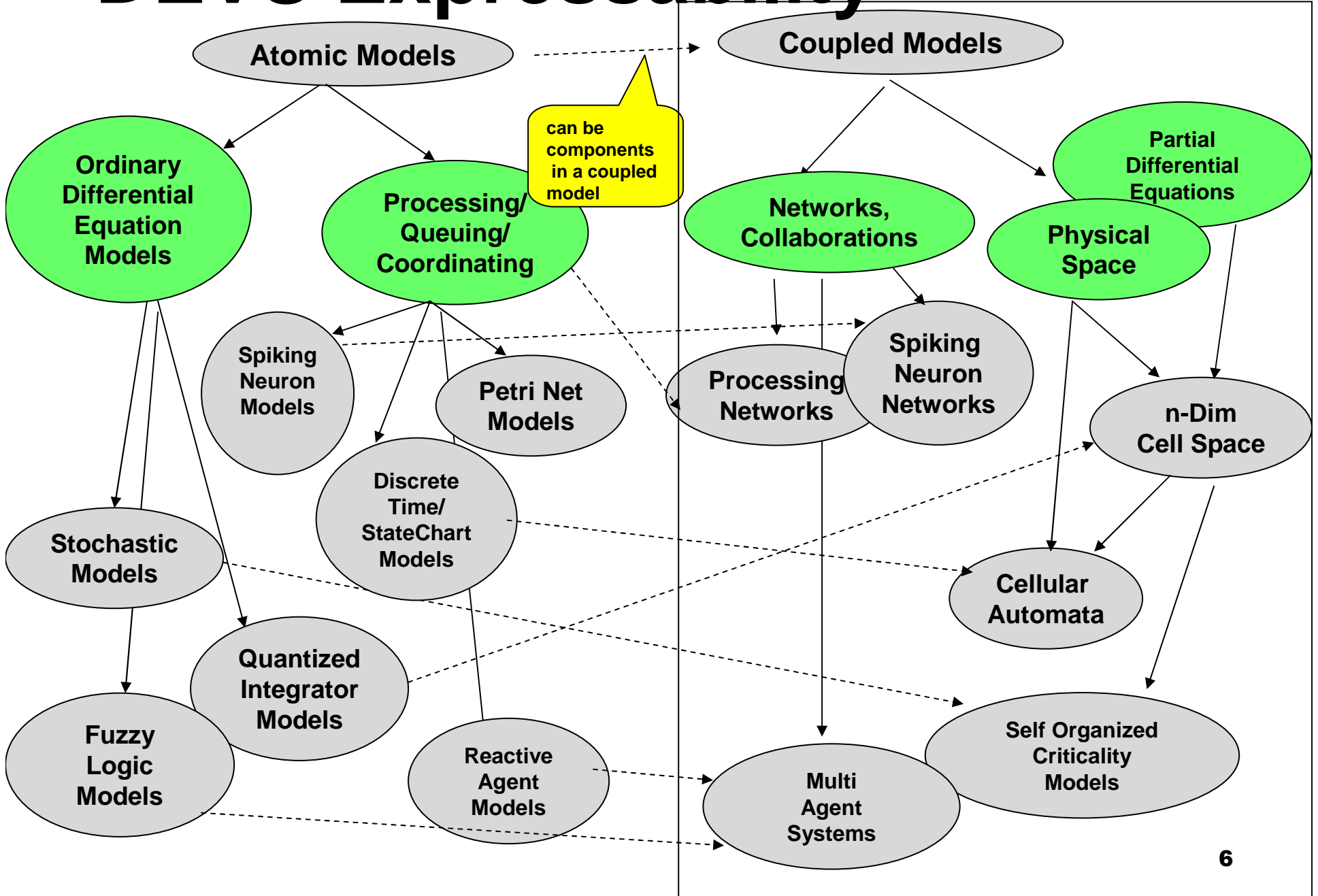
- Conceptual model contains 25,000 objects for 33 goals, 27 tasks, etc.
- Approximately 400,000 lines of code.
- 14 man-years: 6 knowledge engineers and 12 experts

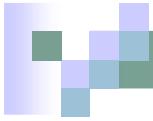
One advantage of DEVS is *compactness*: 50,000 reduction in data volume



Effective analysis and control of the behavior of blast furnaces at high resolution

DEVS Expressability





DEVs notation

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \text{ta} \rangle$$

where

X : set of input values

S : set of states

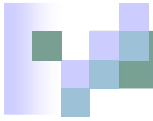
Y : set of output values

δ_{int} : Internal transition function

δ_{ext} : External transition function

λ : Output Function

ta : Time advance function



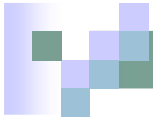
How DEVS works

- System is in State s

- If no external event (δ_{ext}) occurs, the system stays in s for the time period given by the time advance function: $ta(s)$
- After $ta(s)$ time, $e=ta(s)$, system outputs $\lambda(s)$

- If an external event (δ_{ext}) occurs the new state is determined by x (input value), current state s , and e

- e = how long the system was in that state



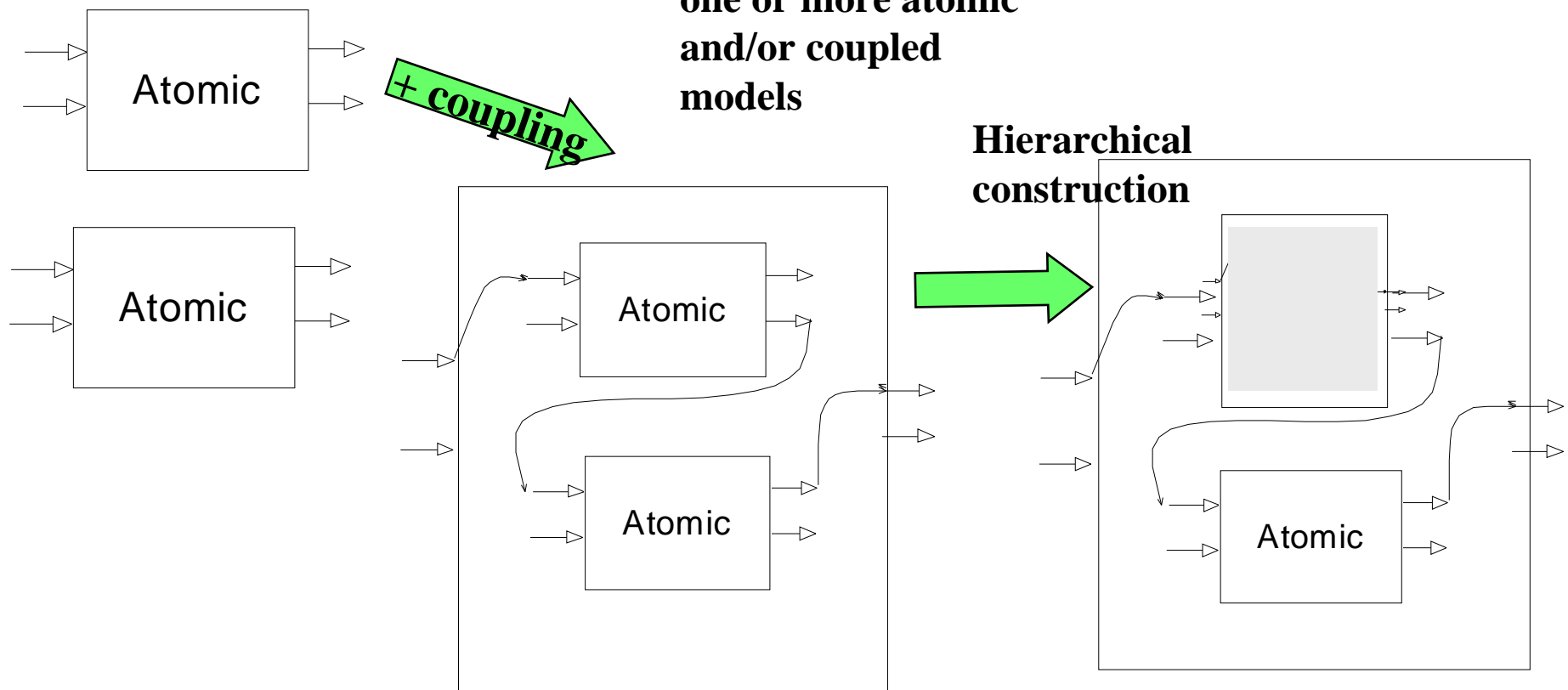
How DEVS works

- Internal transitions generate **output**
 - System states in state “s” for time t_a before making internal transition and generating output
- External transitions do **not** generate **output**
 - Response to external input

DEVS Hierarchical Modular Composition

Atomic: lowest level model,
contains structural dynamics
-- model level modularity

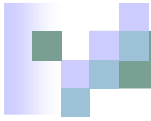
Coupled: composed of
one or more atomic
and/or coupled
models



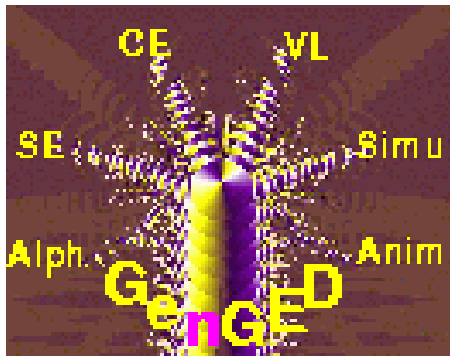


Overview

- Introduction to DEVS
- **Round 1: Basic Diagram Editor**
 - **Modeling tool review**
 - Implementation
- Round 2: The Visual Modeling Environment
- Round 3: Generating PyDEVS code
- Conclusion and Future Work



Modeling tools

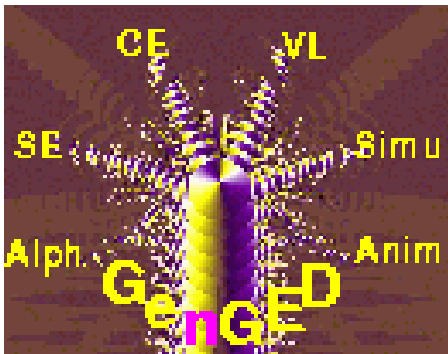


- Generation of Graphical Environments for Design



- A Tool for Multi-formalism and Meta-Modeling

Implementations

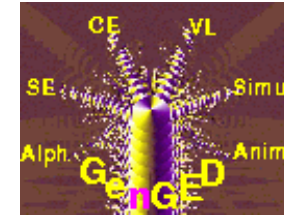


- **Java** but the PARCON constraints handler is in **Objective C**, thus GenGED works properly only on **Linux** (with **libc5**, such as the extinct species **Red Hat 4.0**) & **Solaris**

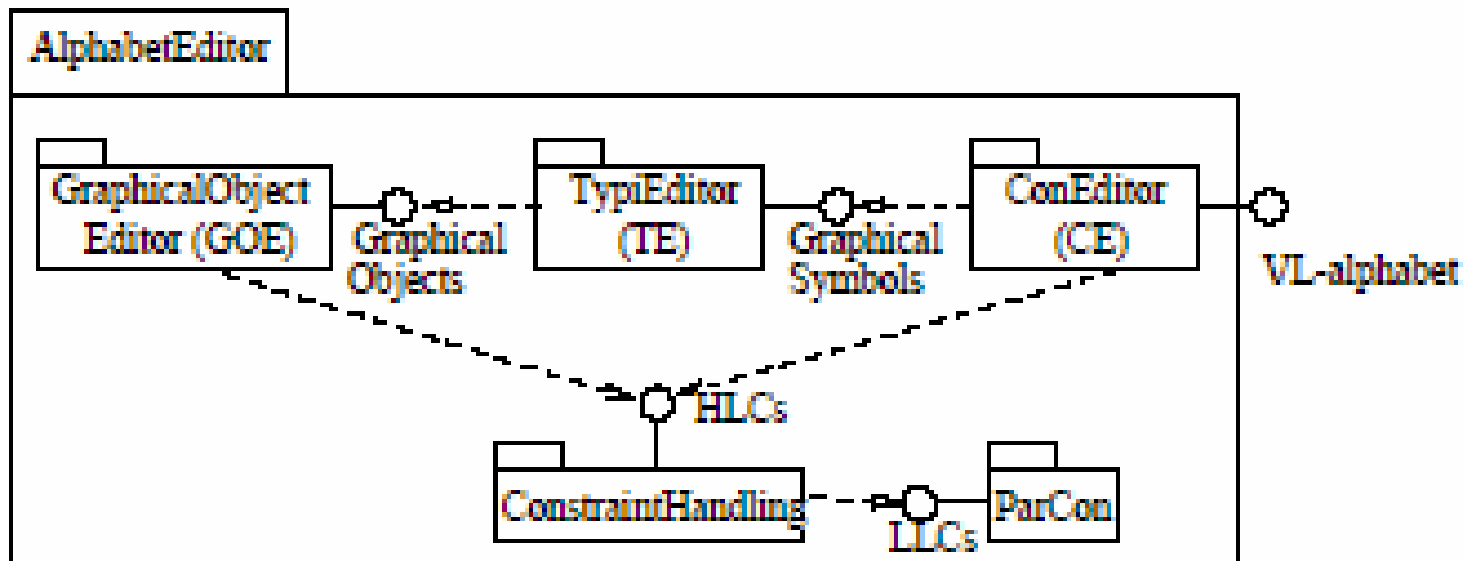


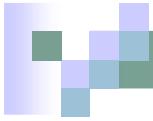
- **Python 2.3** and **Tcl/Tk 8.3** (or better), completely **platform independent** (in theory)

Creating a formalism

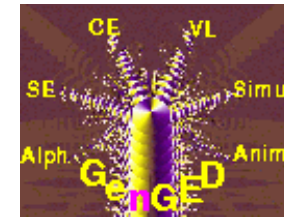


- Graphical Object Editor (draw visual icons)
- TypiEditor (map icons to semantic objects)
- ConEditor (connect semantic objects)

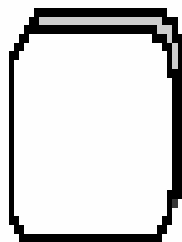




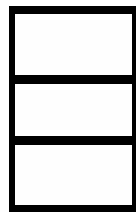
Alphabet editor: GOE



- **Primitive objects:** rectangles, circles, arrows, etc.
- **Composite** of primitive objects linked via graphical constraints



CD



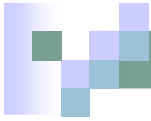
Class



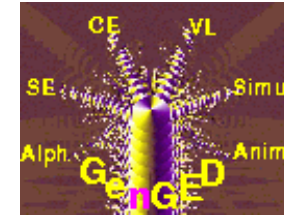
Ass



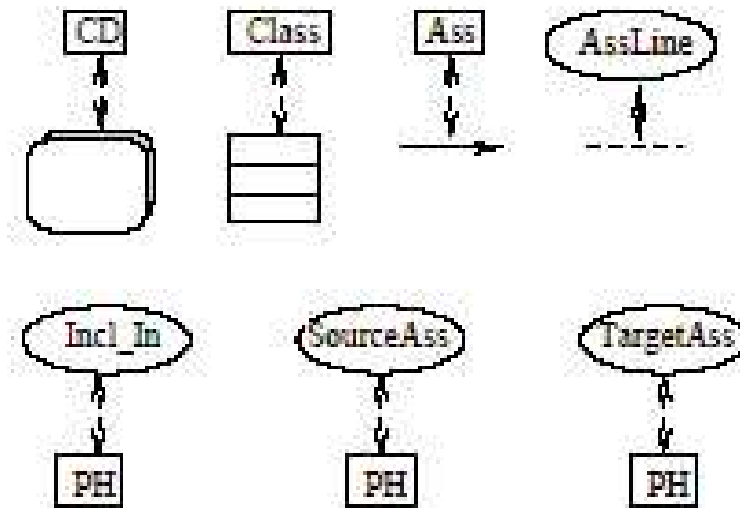
AssLine



Alphabet editor: TypiEditor



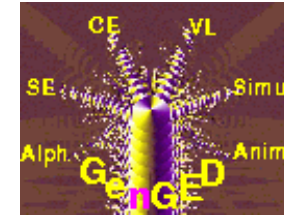
- Mapping to graph nodes/edges of:
 - Graphical Objects
 - Place holders (non-visual)



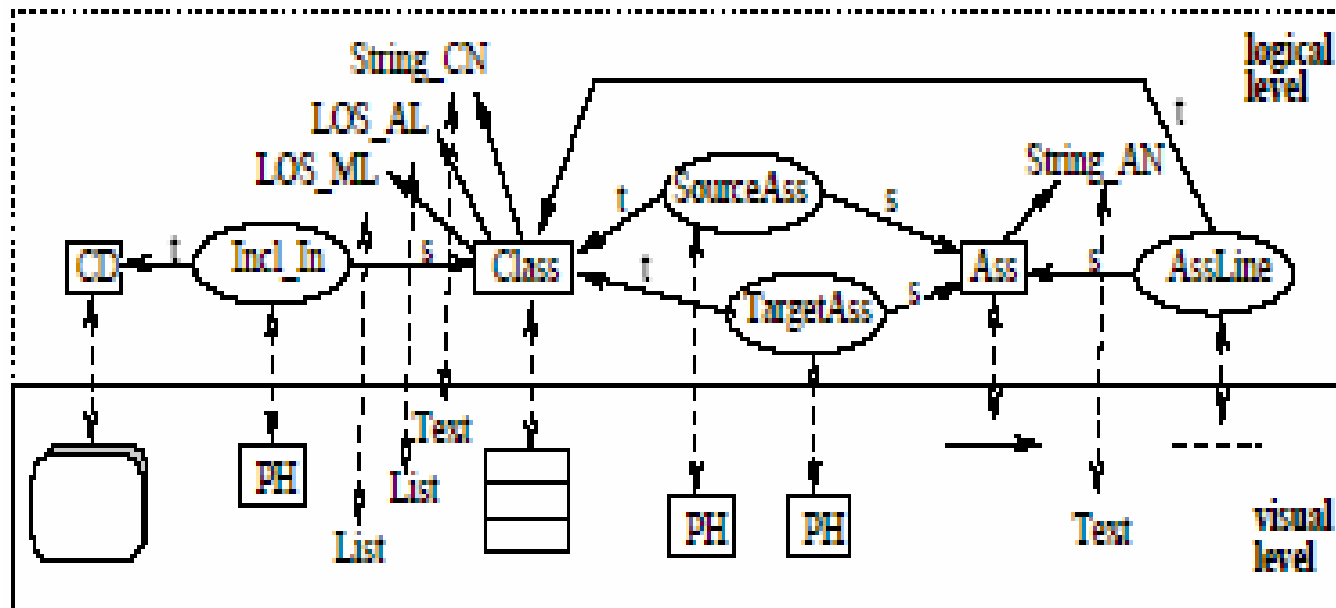
- Creation of **attribute data types** by instantiating built-in data types



Alphabet editor: ConEditor



- **Attribution mode:** map nodes/edges with one or more data types
- **Link mode:** source and target definition for edges





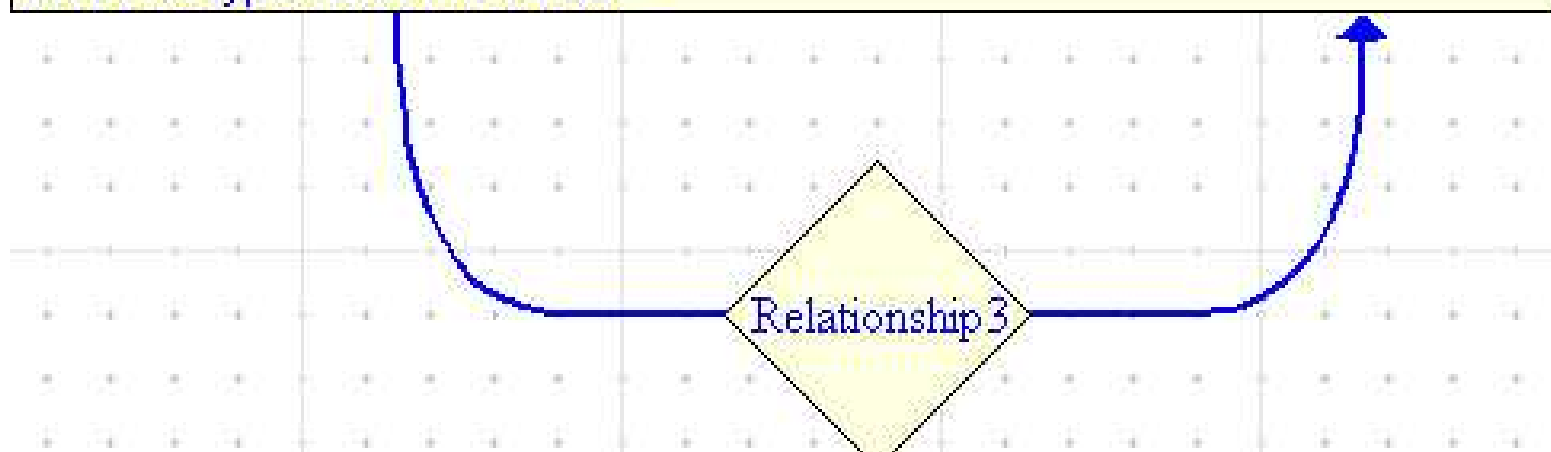
Creating a formalism

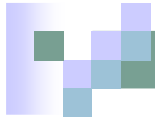


■ Entity Relationship

Entity3

```
name type=String init.value=Entity_  
Graphical_Appearance type=Appearance init.value=graph_class0.py  
cardinality type=List init.value=  
attributes type=List init.value=  
Constraints type=List init.value=  
Actions type=List init.value=
```

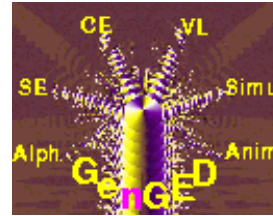




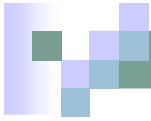
Overview

- Introduction to DEVS
- **Round 1: Basic Diagram Editor**
 - Modeling tool review
 - **Implementation**
- Round 2: The Visual Modeling Environment
- Round 3: Generating PyDEVS code
- Conclusion and Future Work

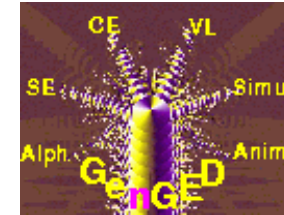
Implementation



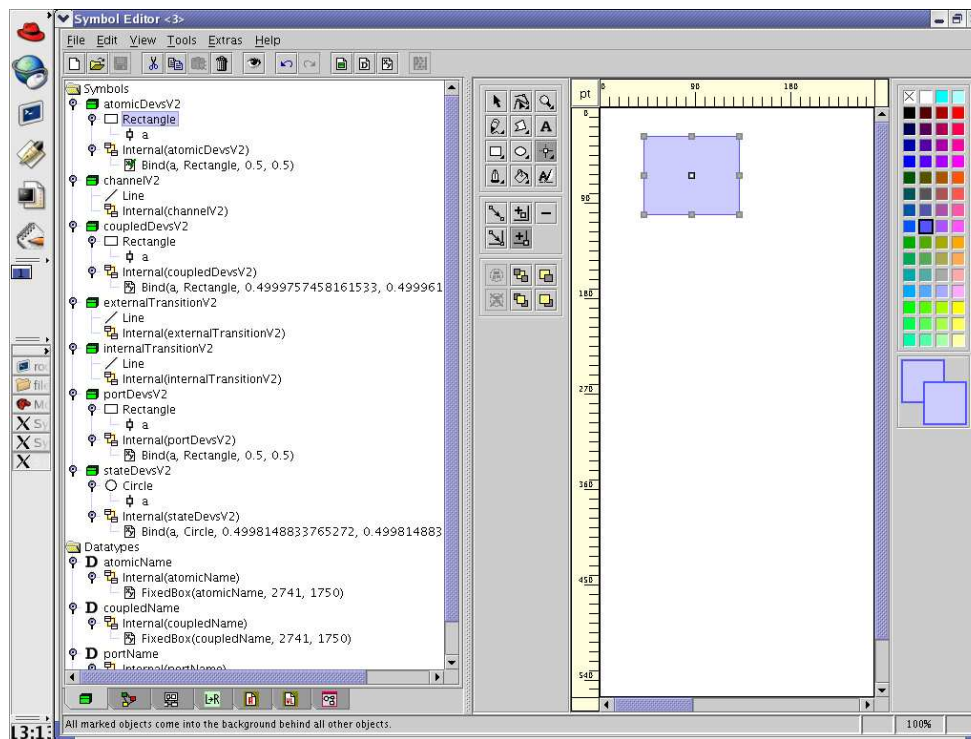
- Goal: Draw diagram with following components:
 - Coupled DEVS (rectangle + name attribute)
 - Atomic DEVS (rectangle + name attribute)
 - States (circle + name attribute)
 - Ports (square + name attribute)
 - Inside relationship → Coupled with Coupled
 - Inside relationship → Atomic with Coupled
 - Inside relationship → State with Atomic
 - Arrow relationship → External & internal transitions
 - Arrow relationship → Channels (between ports)



Alphabet editor



- Graphical Object Editor and TypiEditor?
- Update: Now it's a Symbol Editor



Symbol Editor <3>

File Edit View Tools Extras Help

Symbols

- atomicDevsV2
 - Rectangle
 - a
 - Internal(atomicDevsV2)
 - Bind(a, Rectangle, 0.5, 0.5)
- channelV2
 - Line
 - Internal(channelV2)
- coupledDevsV2
 - Rectangle
 - a
 - Internal(coupledDevsV2)
 - Bind(a, Rectangle, 0.4999757458161533, 0.499961)
- externalTransitionV2
 - Line
 - Internal(externalTransitionV2)
- internalTransitionV2
 - Line
 - Internal(internalTransitionV2)
- portDevsV2
 - Rectangle
 - a
 - Internal(portDevsV2)
 - Bind(a, Rectangle, 0.5, 0.5)
- stateDevsV2
 - Circle
 - a
 - Internal(stateDevsV2)
 - Bind(a, Circle, 0.4998148833765272, 0.499814883)

Datatypes

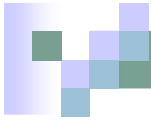
- D atomicName
 - Internal(atomicName)
 - FixedBox(atomicName, 2741, 1750)
- D coupledName
 - Internal(coupledName)
 - FixedBox(coupledName, 2741, 1750)
- D portName
 - Internal(portName)

pt 0 90 180

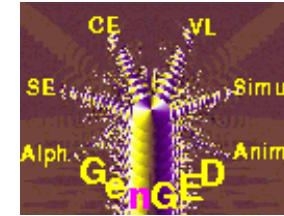
0 90 180 270 360 450 540

100%

All marked objects come into the background behind all other objects.

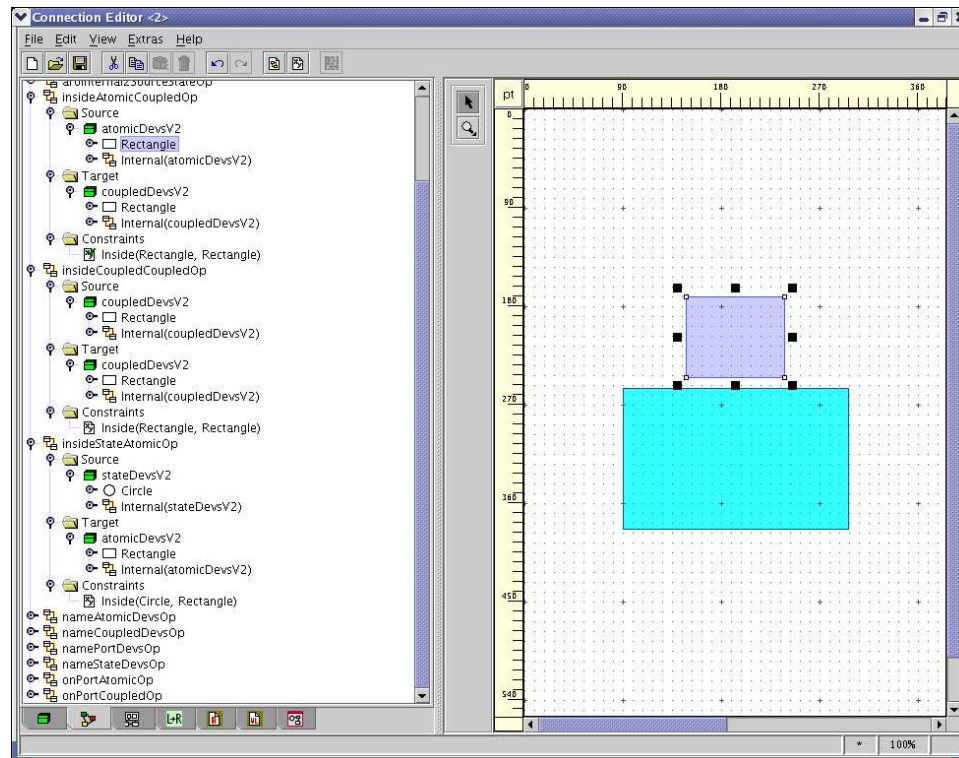


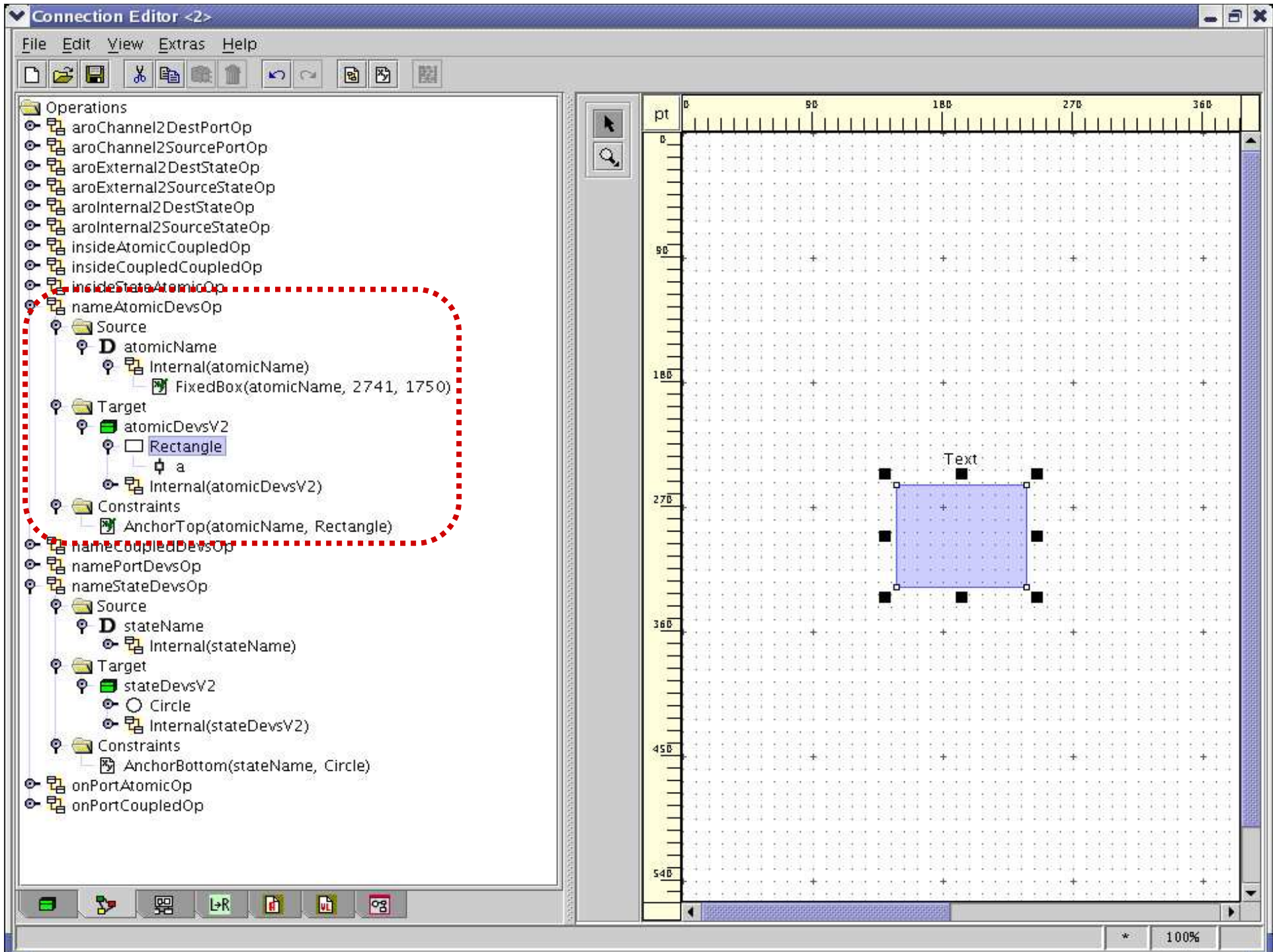
Alphabet editor

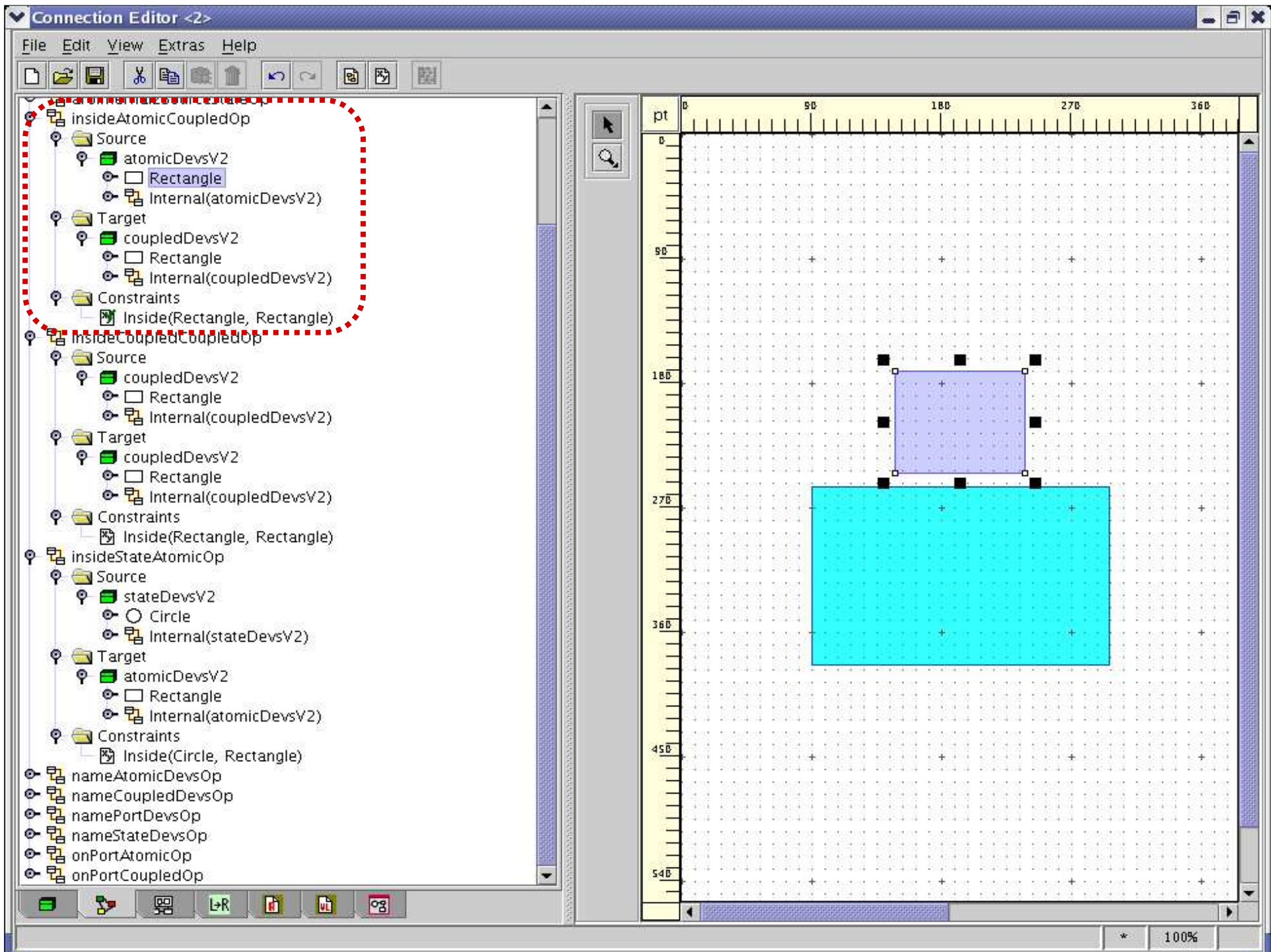


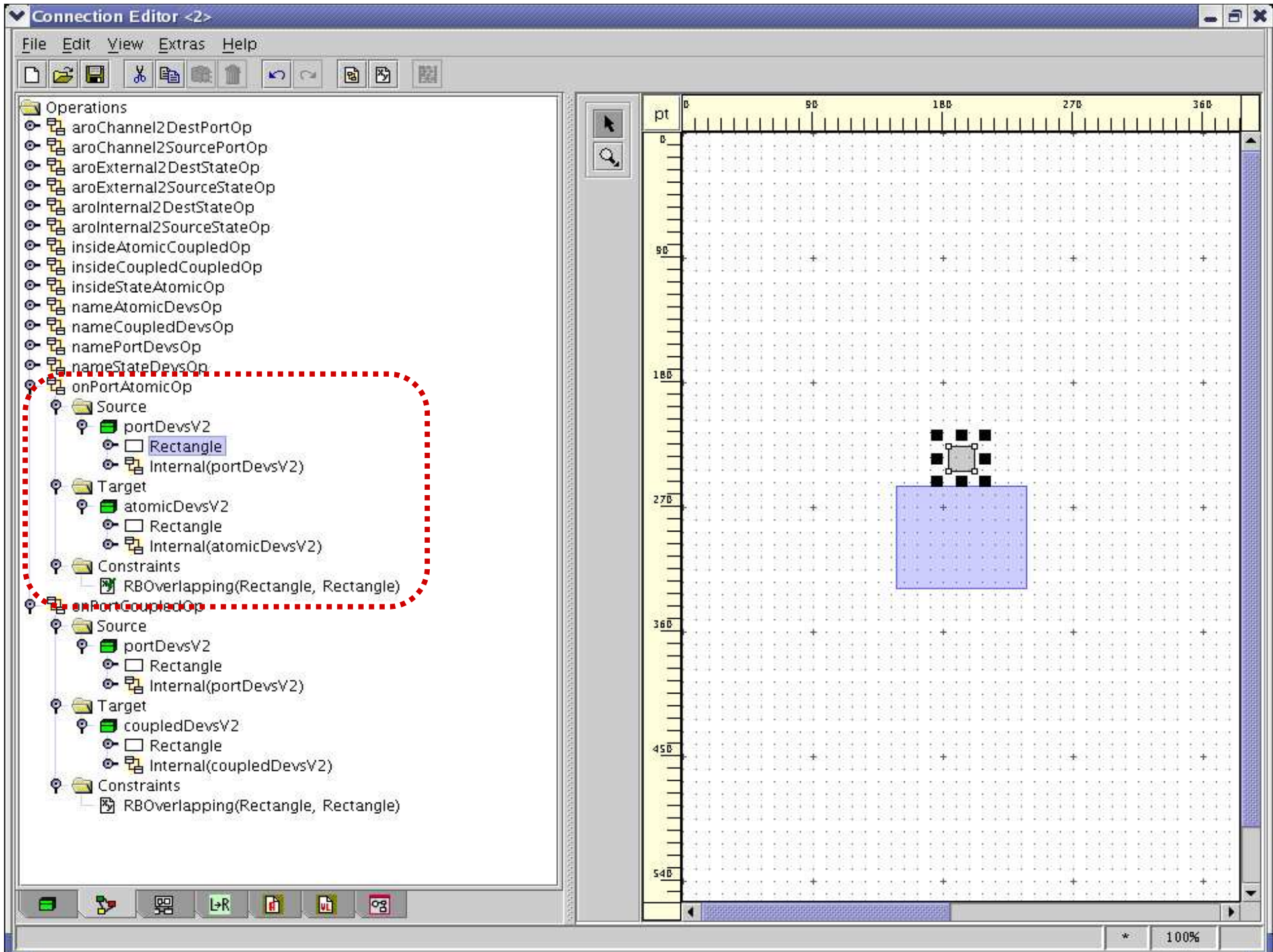
■ Connection Editor

- Specify relationship between different entities









Connection Editor <2>

File Edit View Extras Help

Operations

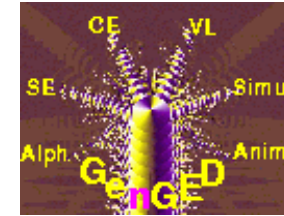
- aroChannel2DestPortOp
- aroChannel2SourcePortOp
- aroExternal2DestStateOp
 - Source
 - externalTransitionV2
 - Line
 - Internal(externalTransitionV2)
 - Target
 - stateDevsV2
 - Circle
 - Internal(stateDevsV2)
 - Constraints
 - AnchorEndPointOnBorder(Line, Circle)
- aroExternal2SourceStateOp
 - Source
 - externalTransitionV2
 - Line
 - Internal(externalTransitionV2)
 - Target
 - stateDevsV2
 - Circle
 - Internal(stateDevsV2)
 - Constraints
 - AnchorStartPointOnBorder(Line, Circle)
- aroInternal2DestStateOp
- aroInternal2SourceStateOp
- insideAtomicCoupledOp
- insideCoupledCoupledOp
- insideStateAtomicOp
- nameAtomicDevsOp
- nameCoupledDevsOp
- namePortDevsOp
- nameStateDevsOp
- onPortAtomicOp
- onPortCoupledOp

pt 0 90 180 270 360

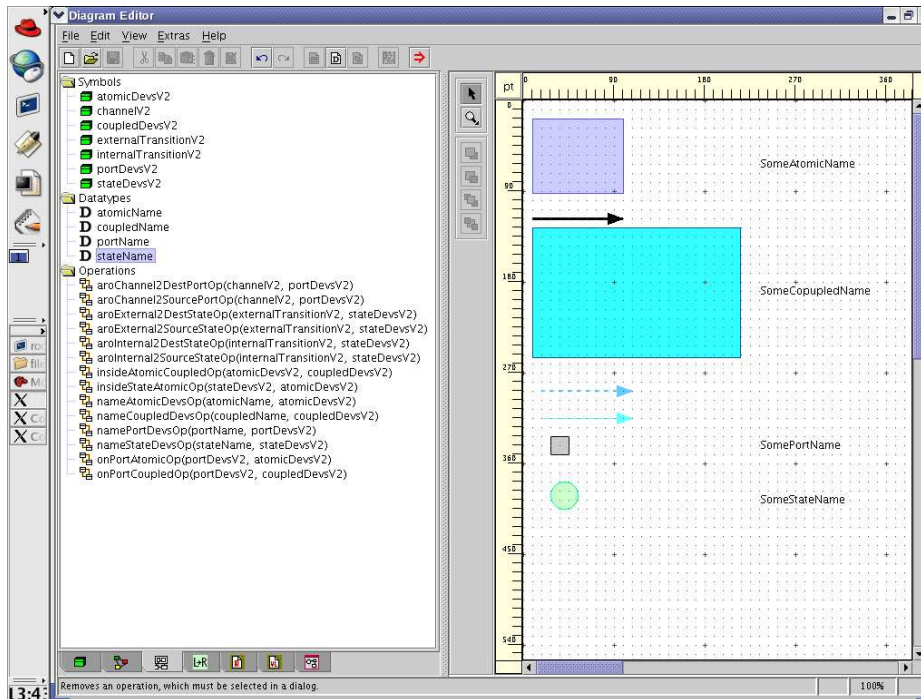
0 90B 180B 270B 360B 450B 540B

13:31 Brings object from memory into the work area. * 100%

Diagram Editor



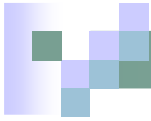
- With our Alphabet defined, we can now generate a diagram editor to test our prototype
 - NOTE: We already have layout at this point!



1. Double click on Symbols or Datatypes and they appear on the canvas

2. Double click on operations, the source, and target to establish connections between Symbols and Datatypes or Symbols and Symbols (with layout constraints)

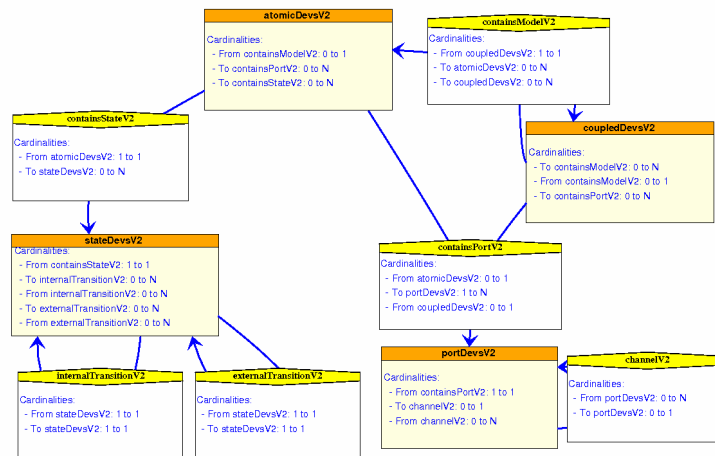
- **Problem:** In our prototype, an arrow relationship requires a total of 6 double clicks to connect its front and back ends to other entities!



Entity Relationship



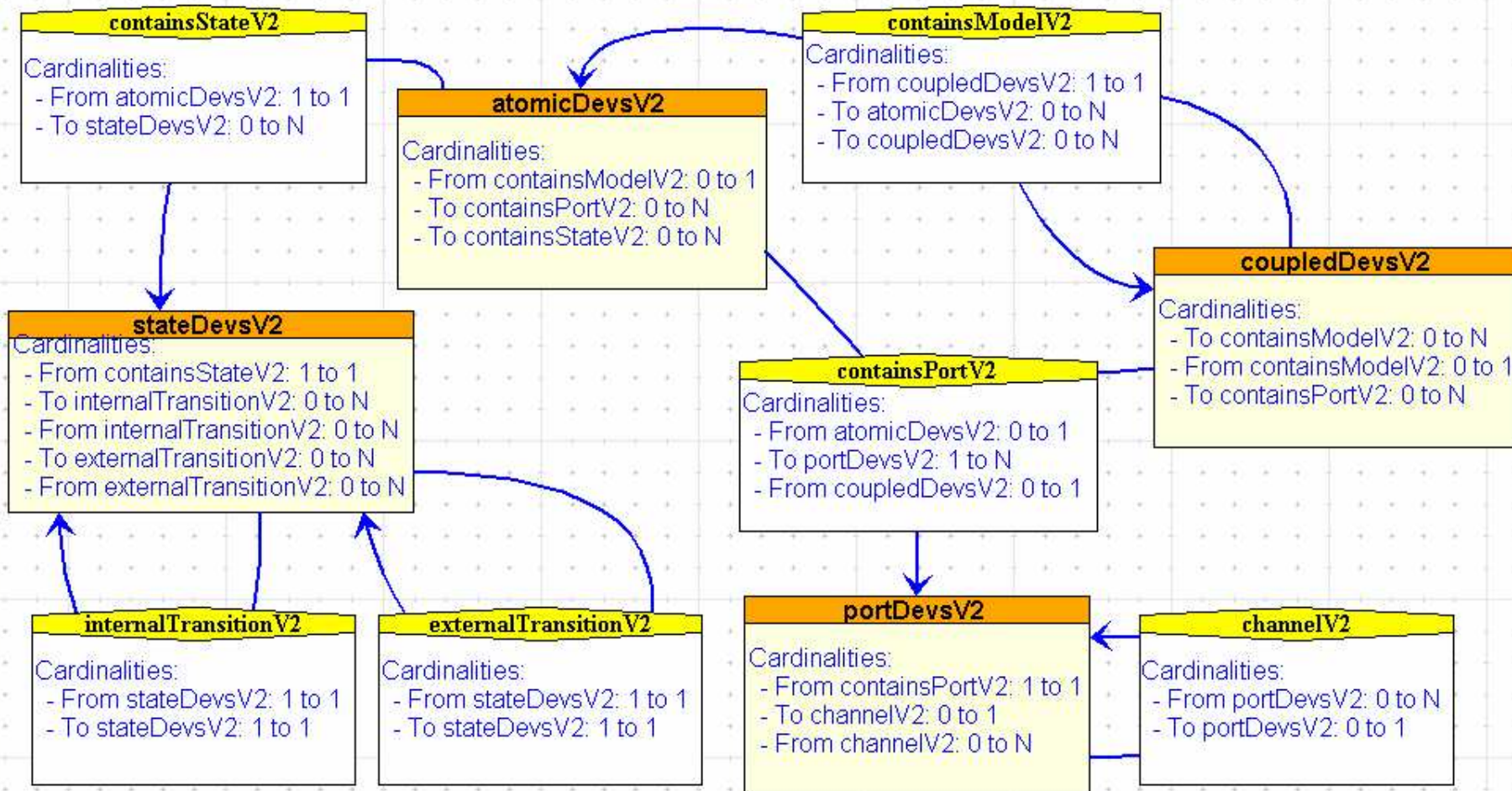
- Startup AToM3 with the default formalism, Entity Relationship version 3
 - Specify Entities, Relationships, and Cardinalities






EntityRelationshipV3

Entity rel. EDIT GEN

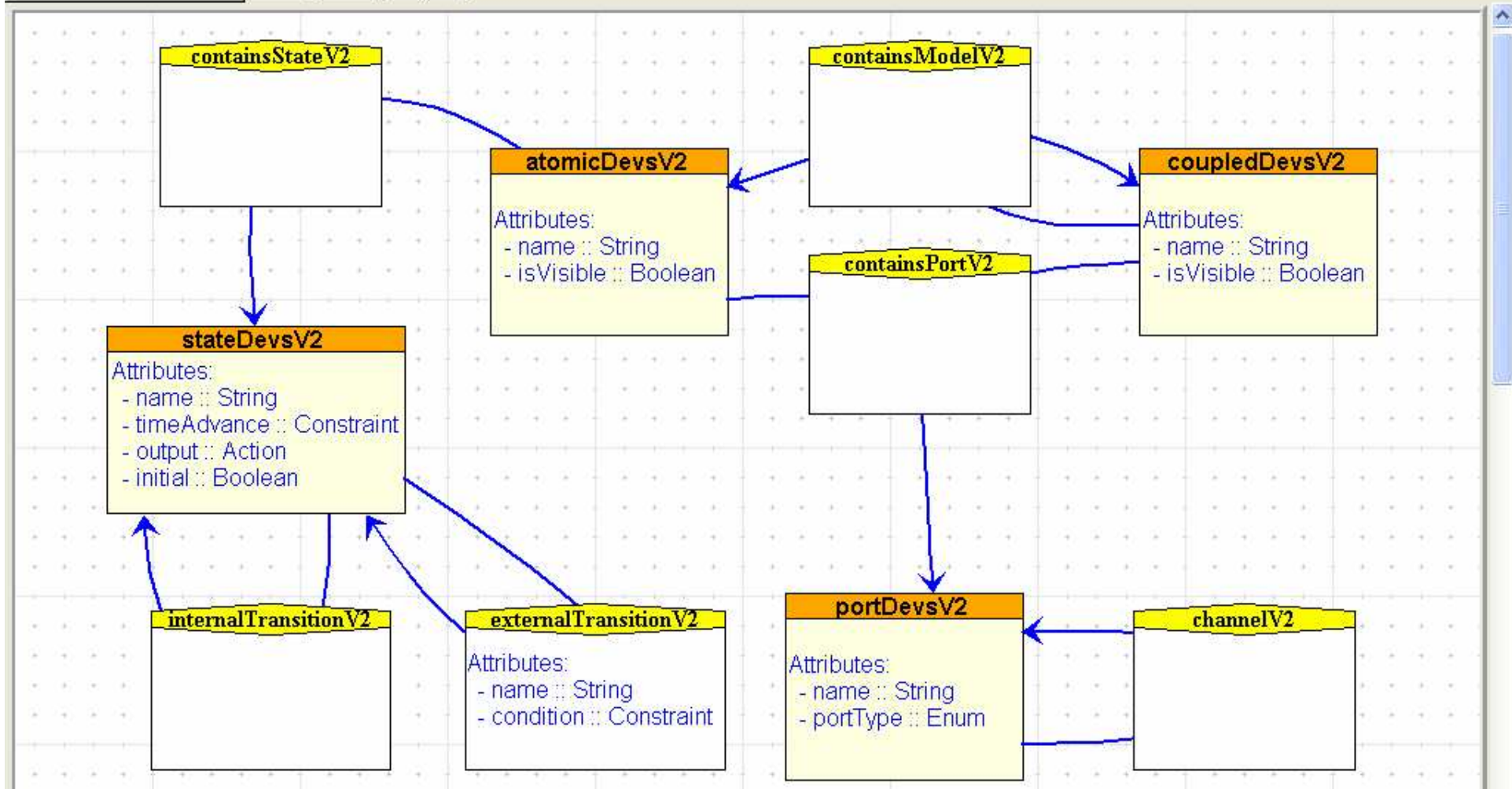


• Note: the cardinalities are consistent with UML class notion of cardinalities but these are enforced at run-time

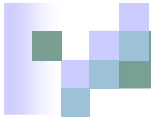


EntityRelationshipV3

Entity rel. EDIT GEN



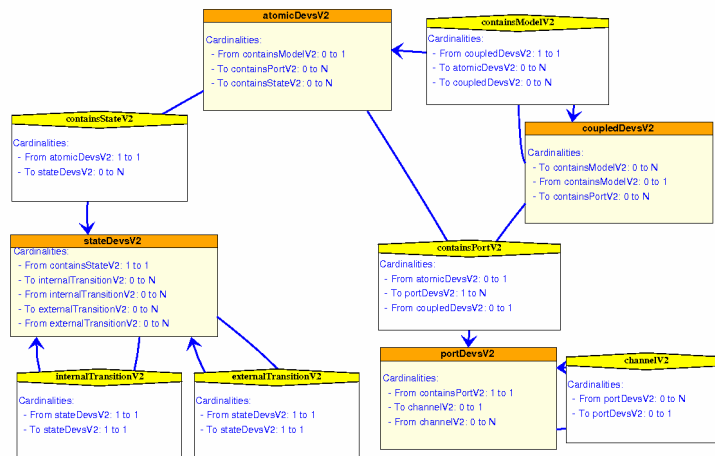
- Attributes must also be set, such as names, but also non-visual attributes like timeAdvance and output, that are used for code generation



Entity Relationship



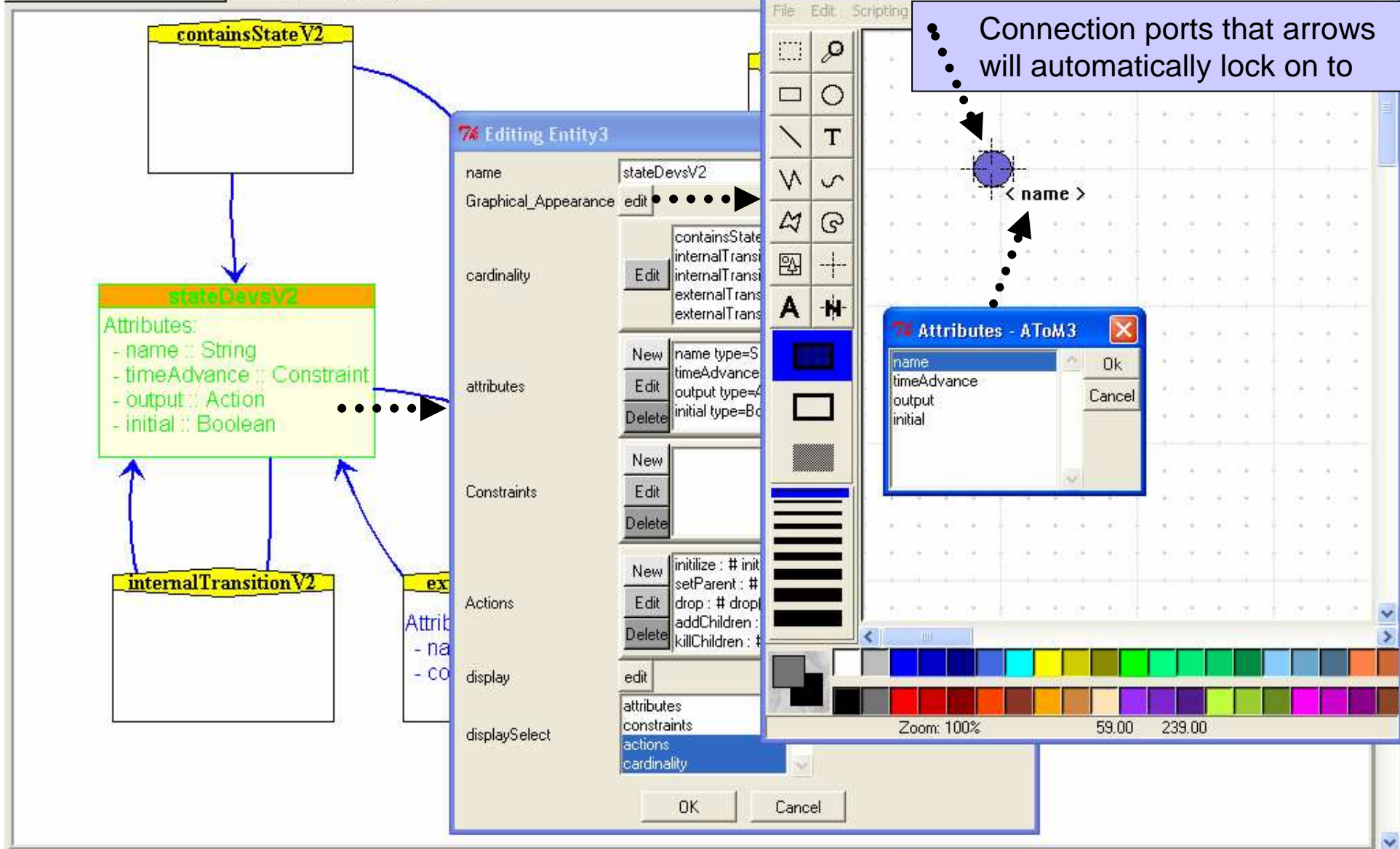
- Now we must provide a graphical representation
 - *Note:* This is the inverse order of how we do things in GenGED and that we are not even dealing with layout yet (except implicitly in the case of arrows)





EntityRelationshipV3

Entity rel. EDIT GEN



74 Editing Entity3

name stateDevsV2

Graphical_Appearance edit

cardinality

attributes

Constraints

Actions

display

displaySelect

edit

attributes

constraints

actions

cardinality

OK Cancel

74 Icon Editor - AToM3

File Edit Scripting

Connection ports that arrows will automatically lock on to

< name >

Attributes - AToM3

name

timeAdvance

output

initial

Ok

Cancel


Zoom: 100%

59.00 239.00

Editing widthXfillXdecoration

width	3
fill	blue
stipple	gray50
arrow	<input type="checkbox"/>
arrowShape (base to tip)	8
arrowShape (wing to tip)	10
arrowShape (wing to base)	3
decoration	edit
decoration_Position	<input type="radio"/> Up <input type="radio"/> Down <input type="radio"/> Middle <input checked="" type="radio"/> No decoration

Refresh Arrow Preview



OK Cancel

Editing linkEditor

FirstLink	edit
FirstSegment	edit
Center	edit
SecondSegment	edit
SecondLink	edit
Is visual?	<input checked="" type="checkbox"/>

OK Cancel

Editing Relationship3

name: externalTransitionV2

Graphical_Appearance: edit

cardinality: stateDevsV2 dir= Destination, min= 1, max=1
stateDevsV2 dir= Source, min= 1, max=1

attributes: name type=String init.value=
condition type=Constraint init.value=condition : #

Constraints:

Actions: connect : # connect
disconnect : # disconne
recursiveAction : # recursiv

display: edit

displaySelect: attributes
constraints
actions
cardinality

OK Cancel

internalTransitionV2

externalTransitionV2

Attributes:
- name :: String
- condition :: Constraint

port

Attributes:
- name
- portType

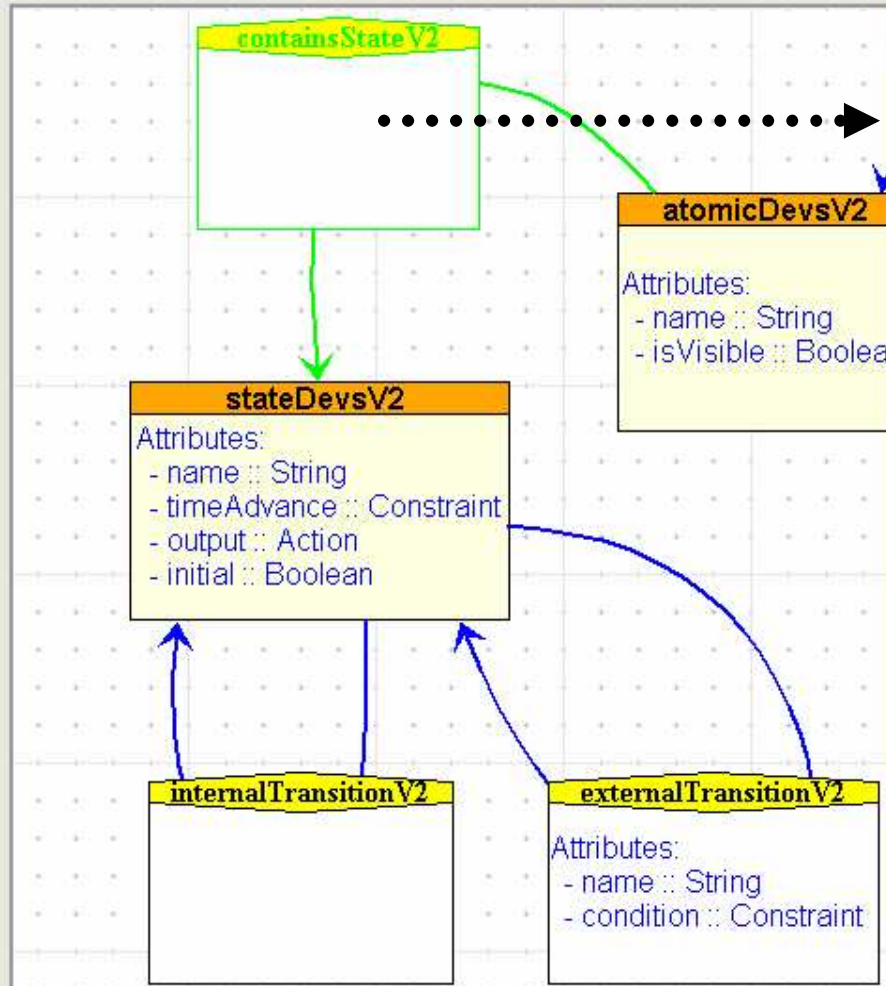
containsModelV2

coupledDevsV2



EntityRelationshipV3

Entity rel. EDIT GEN



74 Editing Relationship3

name: containsStateV2

Graphical_Appearance: edit

cardinality:

- atomicDevsV2 dir= Destination, min= 1, max=1
- stateDevsV2 dir= Source, min= 0, max=N

attributes:

- New
- Edit
- Delete

Constraints:

- New
- Edit
- Delete

Actions:

- New
- Edit
- Delete

display: edit

displaySelect:

- attributes
- constraints
- actions
- cardinality

OK Cancel

74 Editing linkEditor

FirstLink: edit

FirstSegment: edit

Center: edit

SecondSegment: edit

SecondLink: edit

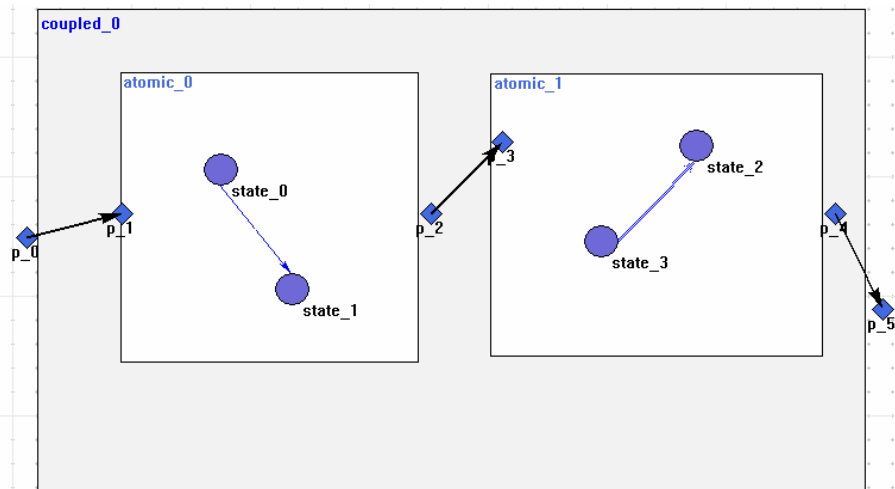
Is visual?

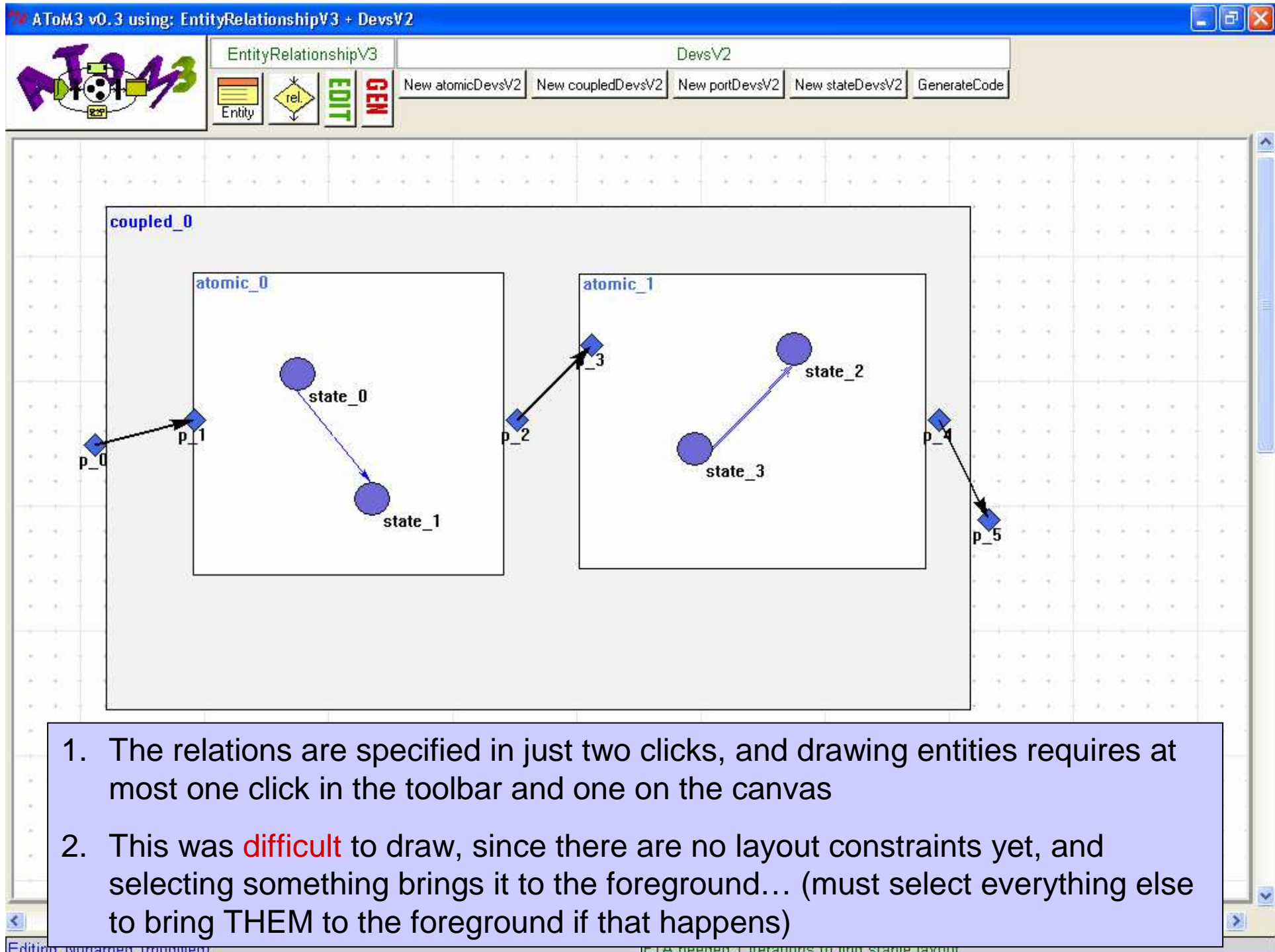
OK Cancel

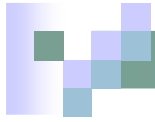
Entity Relationship



- We have now specified everything save any notion of layout
 - This is sufficient to generate a diagram editor and test our prototype



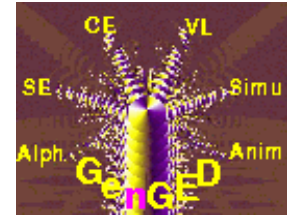




Overview

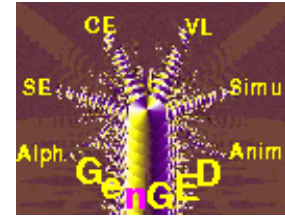
- Introduction to DEVS
- Round 1: Basic Diagram Editor
- Round 2: The Visual Modeling Environment
 - Grammars and Constraint/Actions
 - Implementation
- Round 3: Generating PyDEVS code
- Conclusion and Future Work

Grammars & Constraints



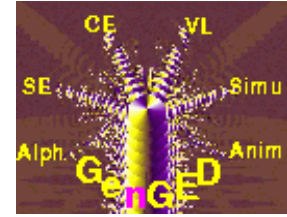
- Build a **syntax grammar**
 - The GenGED grammar editor can automatically generate “**Alphabet Rules**” from the visual alphabet
 - In the GenGED examples, this is sufficient for a **syntax grammar**

Grammars & Constraints



- **Syntax grammar** rules allow us to explicitly define how entities are:
 - Inserted**
 - Deleted**
 - Re-named**
 - Connected**

Grammars & Constraints



- We may also want to define a **parse grammar**
 - Ensures that the diagram represents a **correct** DEVS model!
- The **parse grammar** works by:
 - Reducing it to an empty diagram with the parse rulesOr
 - Augmenting an empty diagram to the arbitrary diagram with the parse rules

Grammars & Constraints

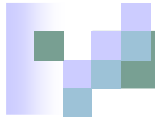


- No **syntax grammar**, but we can use:
 1. **Pre/Post conditions** (Constraints)
 - Can use this to make sure a state in one atomic DEVS does not have a transition to a state in another atomic DEVS
 2. **Pre/Post actions**
 - Directly create hierarchical structure, add layout constraints
 - Or simply forward events to another model designed to handle reactive behavior...

Grammars & Constraints



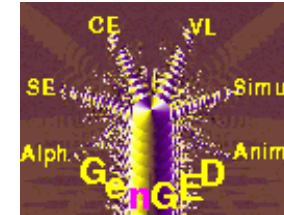
- Parse grammar
- In theory, it would be possible to write a graph grammar in AToM³ that reduces a model to an empty diagram with rules that can only be applied to a correct diagram
- In practice, I don't think this has ever been tried in AToM³...



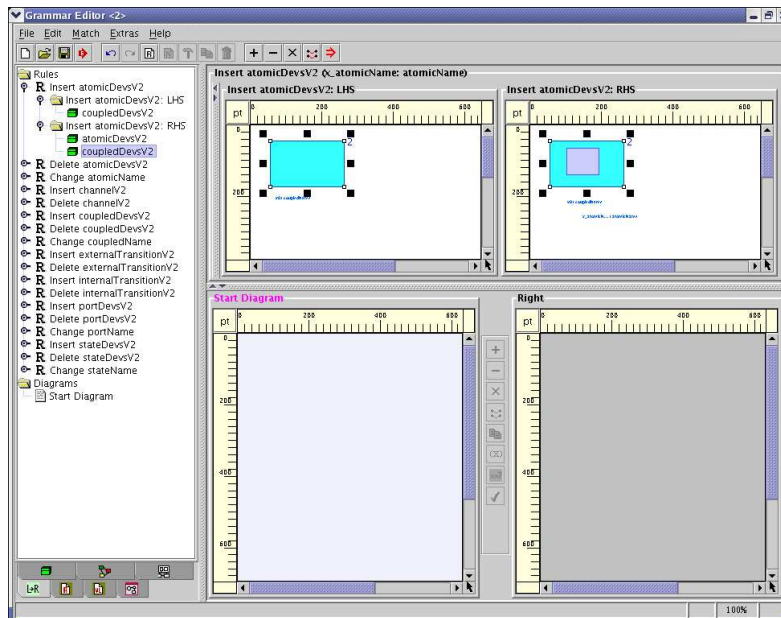
Overview

- Introduction to DEVS
- Round 1: Basic Diagram Editor
- Round 2: The Visual Modeling Environment
 - Grammars and Constraint/Actions
 - Implementation
- Round 3: Generating PyDEVS code
- Conclusion and Future Work

Grammar editor



- Automatic generation of “Visual Language Rules”
- Should be sufficient for a **syntax grammar** which we can generate from the rules



Grammar Editor

File Edit Match Extras Help

Rules

- Insert atomicDevsV2
- Delete atomicDevsV2
- Change atomicName
- Insert channelV2
- Delete channelV2
- Insert coupledDevsV2
 - Insert coupledDevsV2: L
 - Insert coupledDevsV2: R
 - coupledDevsV2
- Delete coupledDevsV2
- Change coupledName
- Insert externalTransitionV2
- Delete externalTransitionV2
- Insert internalTransitionV2
- Delete internalTransitionV2
- Insert portDevsV2
- Delete portDevsV2
- Change portName
- Insert stateDevsV2
- Delete stateDevsV2
- Change stateName
- test
 - test: LHS
 - coupledDevsV2
 - test: RHS
 - atomicDevsV2
 - coupledDevsV2

Diagrams

- Start Diagram

Insert coupledDevsV2 (x_coupledName: coupledName)

Insert coupledDevsV2: LHS

Insert coupledDevsV2: RHS

- Above we see the automatically generated rule for inserting a coupledDevsV2 entity
- The LHS is empty, so we can always add a new coupledDevsV2 entity
- The RHS includes not just the entity, but the associated Datatype (attribute), thus saving us time!
- FYI: “V2” means nothing special

100%

Grammar Editor <2>

File Edit Match Extras Help

Rules

- R Insert atomicDevsV2
 - Insert atomicDevsV2: LHS
 - coupledDevsV2
 - Insert atomicDevsV2: RHS
 - atomicDevsV2
 - coupledDevsV2
- R Delete atomicDevsV2
- R Change atomicName
- R Insert channelV2
- R Delete channelV2
- R Insert coupledDevsV2
- R Delete coupledDevsV2
- R Change coupledName
- R Insert externalTransitionV2
- R Delete externalTransitionV2
- R Insert internalTransitionV2
- R Delete internalTransitionV2
- R Insert portDevsV2
- R Delete portDevsV2
- R Change portName
- R Insert stateDevsV2
- R Delete stateDevsV2
- R Change stateName

Diagrams

- Start Diagram

Insert atomicDevsV2 (atomicName: atomicName)

Insert atomicDevsV2: LHS

Insert atomicDevsV2: RHS

• Above we see the automatically generated rule for inserting an atomicDevsV2 entity

• The LHS is NOT empty, so we can only insert this entity when a coupledDevsV2 is present in our working diagram

• It is not possible to edit automatically generated rules and since this is the only rule to allow insertion of an atomicDevsV2, we have a problem as we can NOT create a new rule that does not require a coupleDevsV2 in the LHS

• Perhaps an error was made in the Alphabet editor...

100%

Grammar Editor <2>

File Edit Match Extras Help

Rules

- R Insert atomicDevsV2
- R Delete atomicDevsV2
- R Change atomicName
- R Insert channelV2
- R Delete channelV2
- R Insert coupledDevsV2
- R Delete coupledDevsV2
- R Change coupledName
- R Insert externalTransitionV2
- R Delete externalTransitionV2
- R Insert internalTransitionV2
- R Delete internalTransitionV2
- R Insert portDevsV2
 - Insert portDevsV2: LHS
 - atomicDevsV2
 - coupledDevsV2
 - coupledDevsV2
 - Insert portDevsV2: RHS
 - atomicDevsV2
 - coupledDevsV2
 - coupledDevsV2
 - portDevsV2
- R Delete portDevsV2
- R Change portName
- R Insert stateDevsV2
- R Delete stateDevsV2
- R Change stateName

Diagrams

- Start Diagram

Insert portDevsV2 (<_portName: portName)

Insert portDevsV2: LHS

Insert portDevsV2: RHS

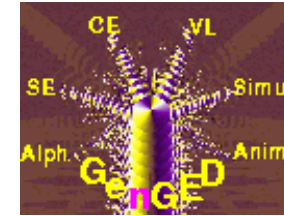
Start Diagram

Right

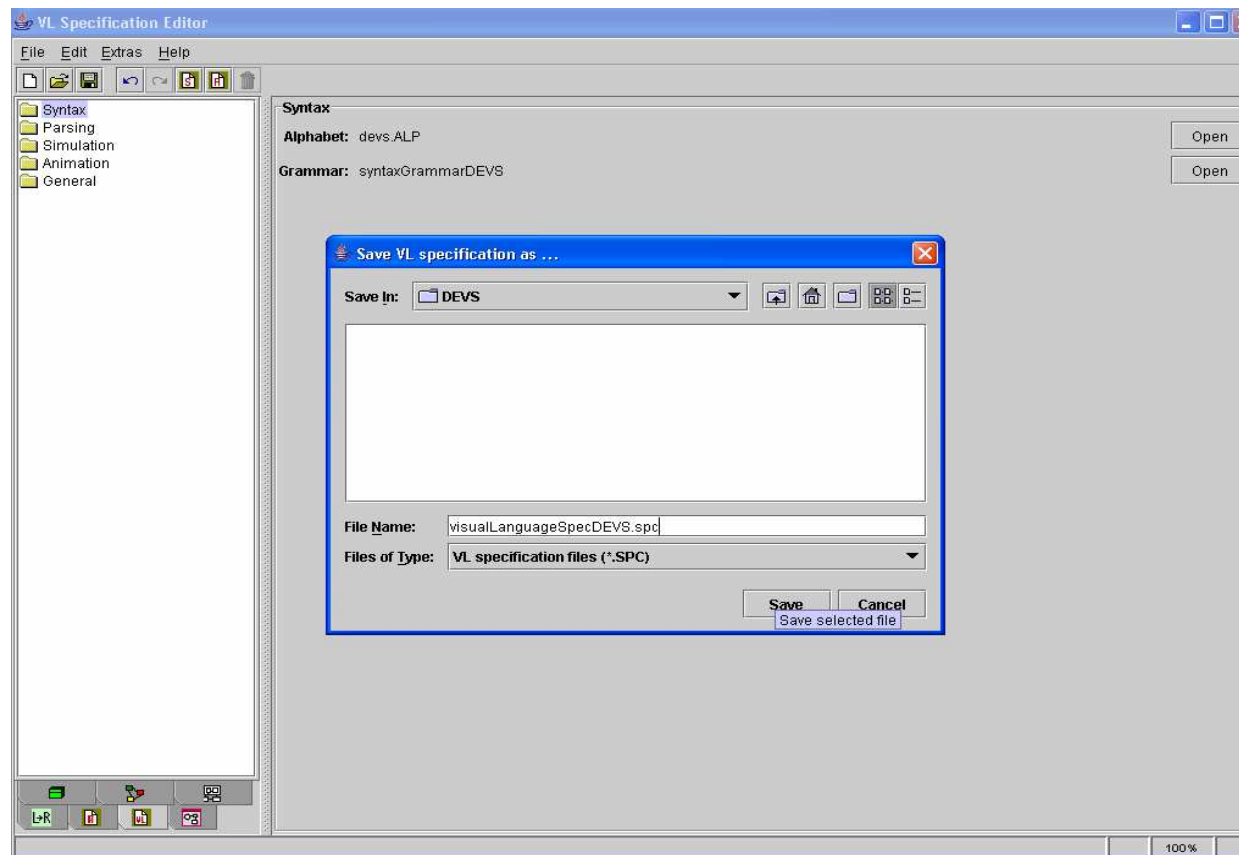
- Indeed, this insert rule for portDevsV2 is complete nonsense

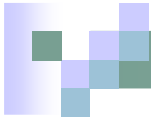
100%

Visual Language Spec

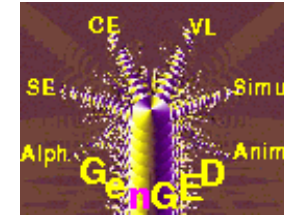


- Automatic generation of a **visual language environment**
 - NOTE: Flaws in the automatically generated syntax grammar are ignored

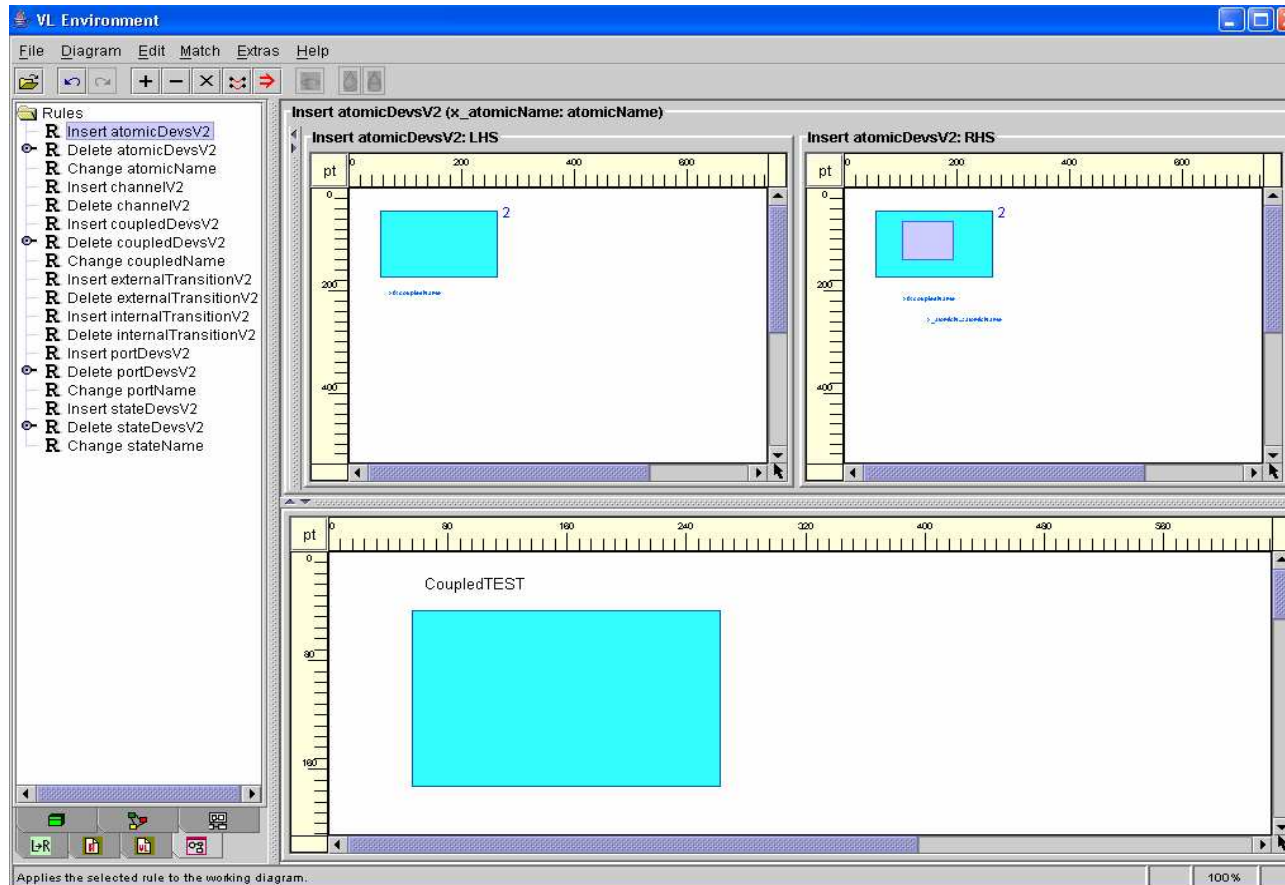




VL Environment



- Diagram creation/editing done using **syntax grammar** rules



VL Environment

File Diagram Edit Match Extras Help

Rules

- Insert atomicDevsV2
- Delete atomicDevsV2
- Change atomicName
- Insert channelV2
- Delete channelV2
- Insert coupledDevsV2
- Delete coupledDevsV2
- Change coupledName
- Insert externalTransitionV2
- Delete externalTransitionV2
- Insert internalTransitionV2
- Delete internalTransitionV2
- Insert portDevsV2
- Delete portDevsV2
- Change portName
- Insert stateDevsV2
- Delete stateDevsV2
- Change stateName

Insert atomicDevsV2 (x_atomicName: atomicName)

Insert atomicDevsV2: LHS

Insert atomicDevsV2: RHS

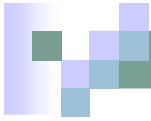
CoupledTEST

- The syntax grammar makes creation of entities with attributes much easier
- However creating a relationship is even harder now!
- In general: requires 8 clicks... none of which are in close proximity

Applies the selected rule to the working diagram.

100%

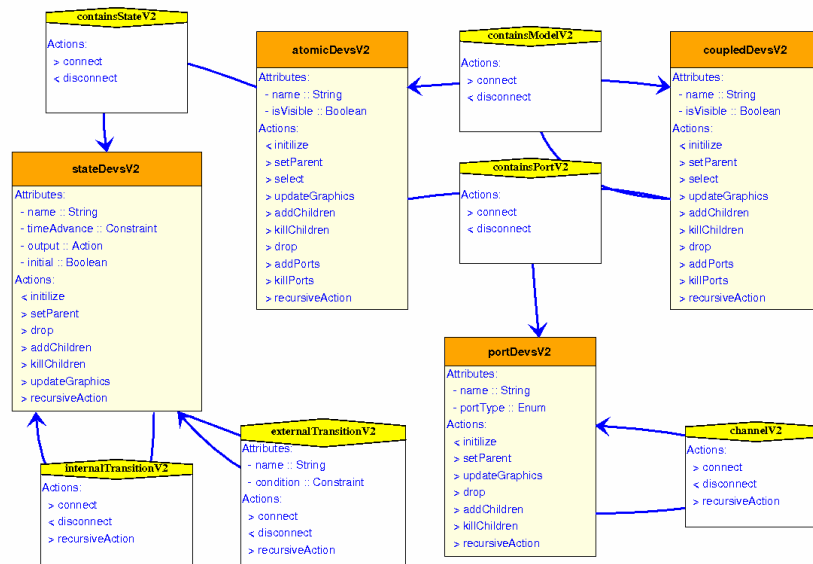
The screenshot shows the VL Environment software interface. At the top is a menu bar with 'File', 'Diagram', 'Edit', 'Match', 'Extras', and 'Help'. Below the menu is a toolbar with icons for file operations and editing. On the left is a 'Rules' panel with a list of rules, including 'Insert atomicDevsV2' which is selected. The main workspace is divided into three panels: 'Insert atomicDevsV2: LHS', 'Insert atomicDevsV2: RHS', and 'CoupledTEST'. The LHS and RHS panels show a diagram with a cyan rectangle labeled '2' and a blue label '>_dcoupleName'. The CoupledTEST panel shows a diagram with a cyan rectangle and a blue label '>_zondich...zondichName'. A text box in the center contains three bullet points. At the bottom, there is a status bar with the text 'Applies the selected rule to the working diagram.' and a zoom level of '100%'.



Pimping out the ER



- Add Constraints/Actions code → Layout!



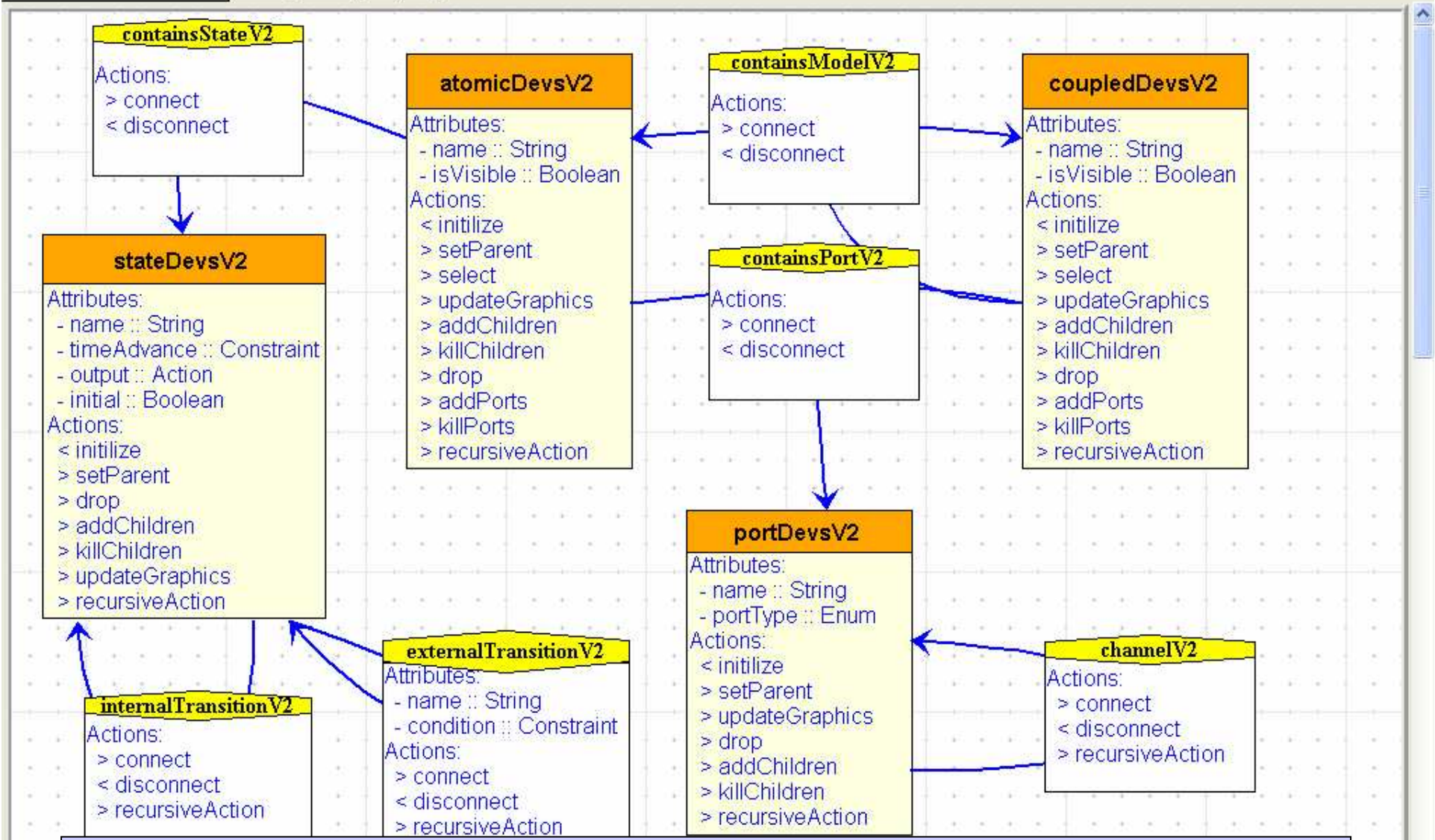


EntityRelationshipV3

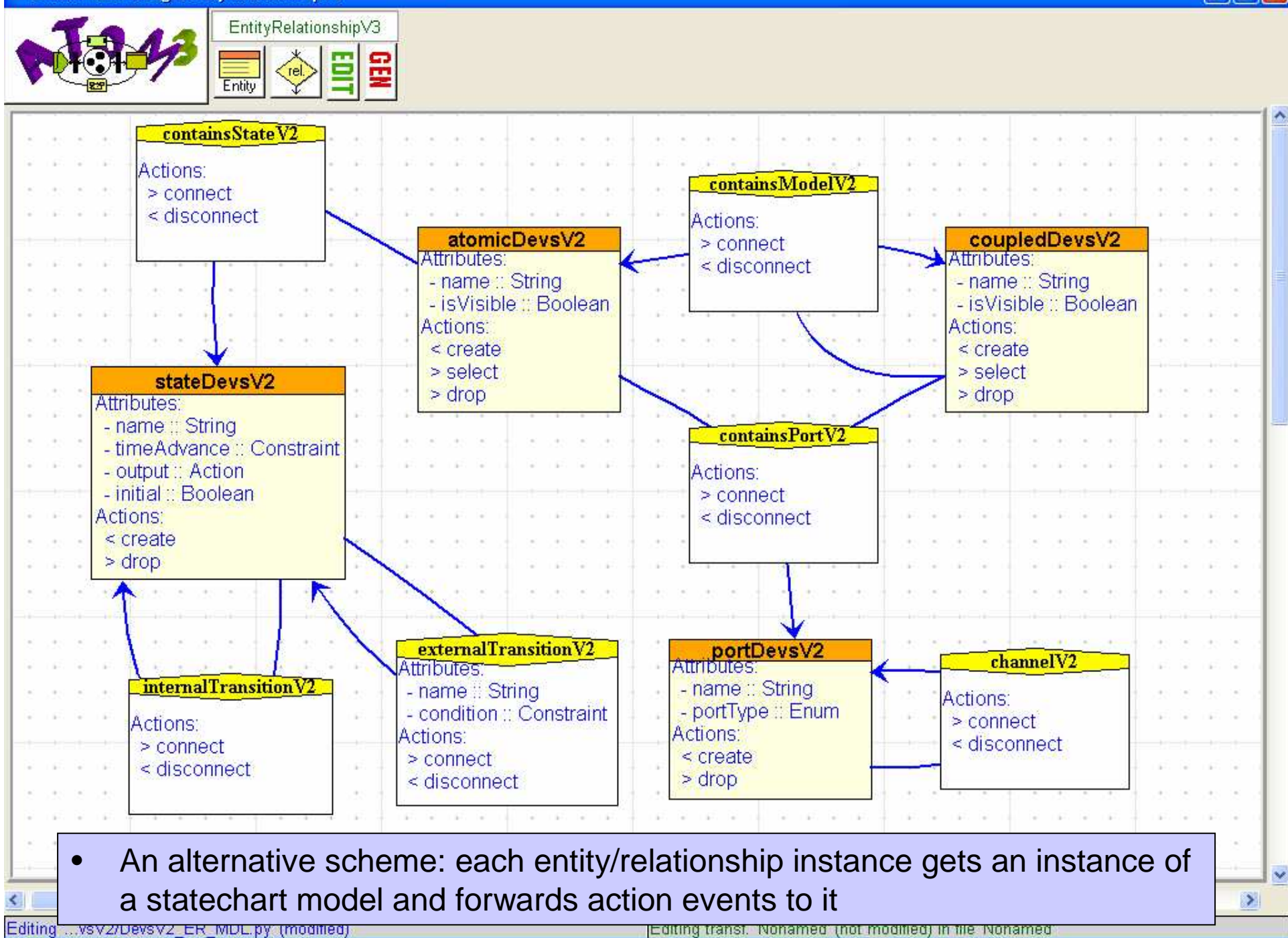
Entity rel. EDIT GEN

DevsV2

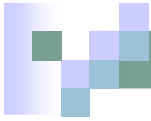
New atomicDevsV2 New coupledDevsV2 New portDevsV2 New stateDevsV2 GenerateCode



• A quick and dirty method, much copy&pasted code hidden away in here...



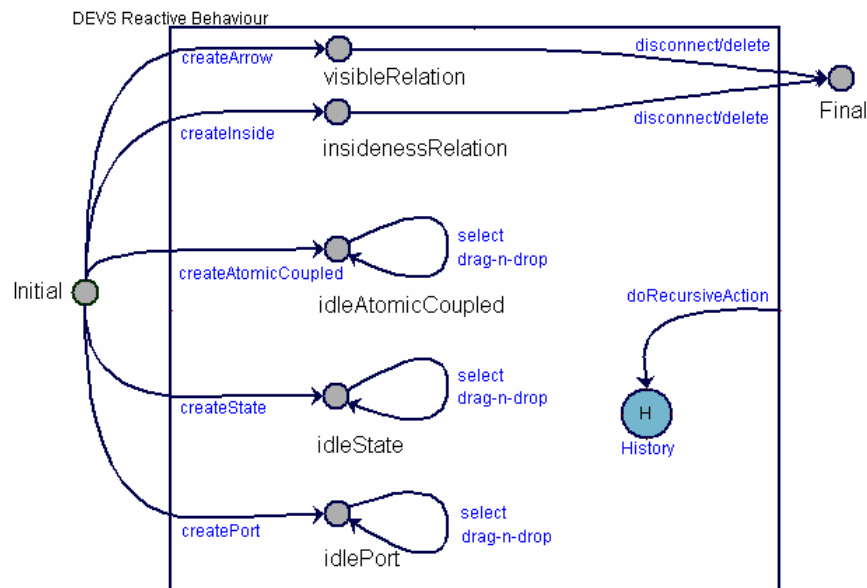
- An alternative scheme: each entity/relationship instance gets an instance of a statechart model and forwards action events to it



Reactive behavior

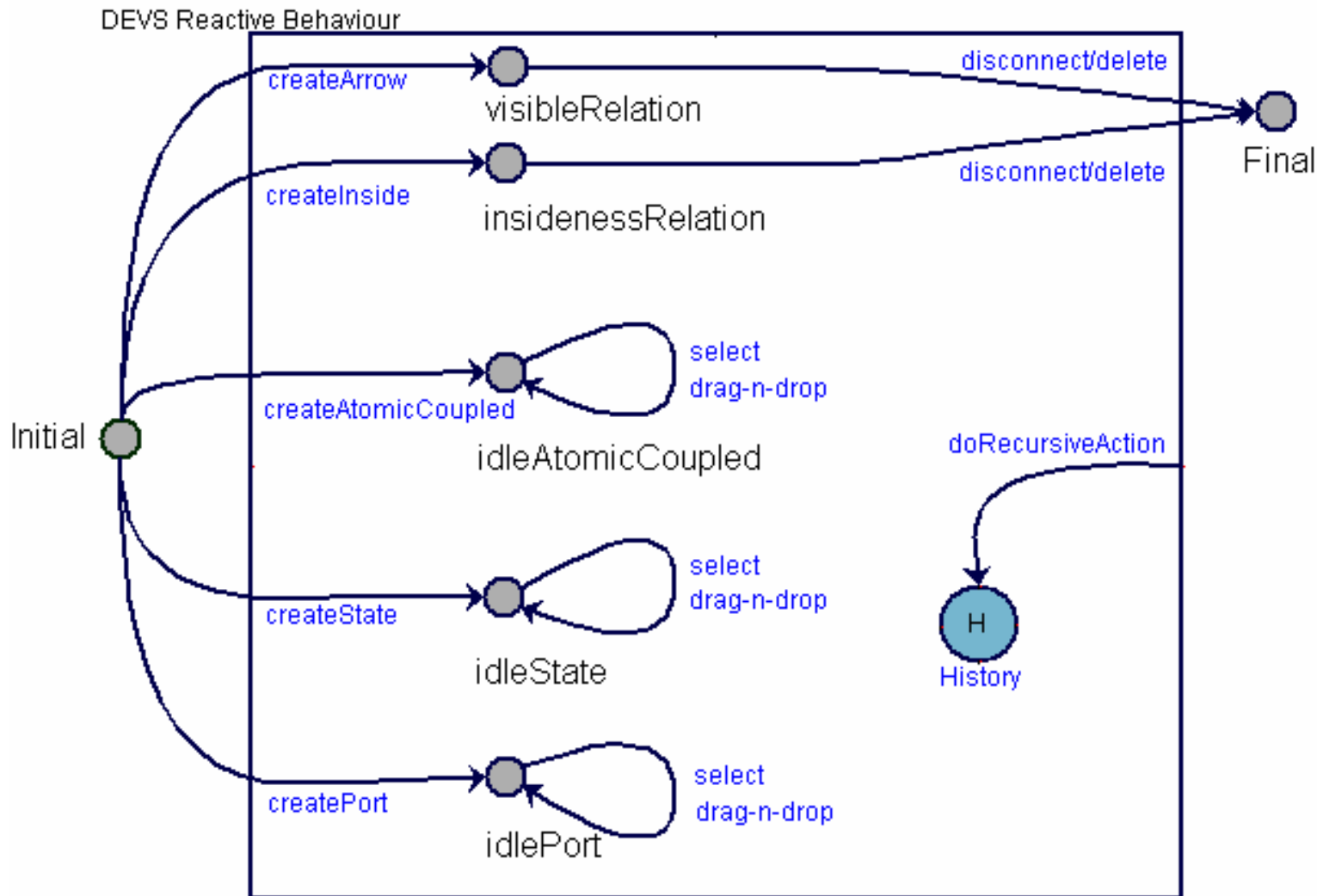


- Explicitly modeled behavior with DCharts (a form of statecharts by Thomas Feng)
- Each entity gets an instance of the DChart model
- Layout actions are triggered by creation, selection, and dragging





Reactive behavior



EntityRelationshipV3
DevsV2

Entity

rel.

EDIT

GEN

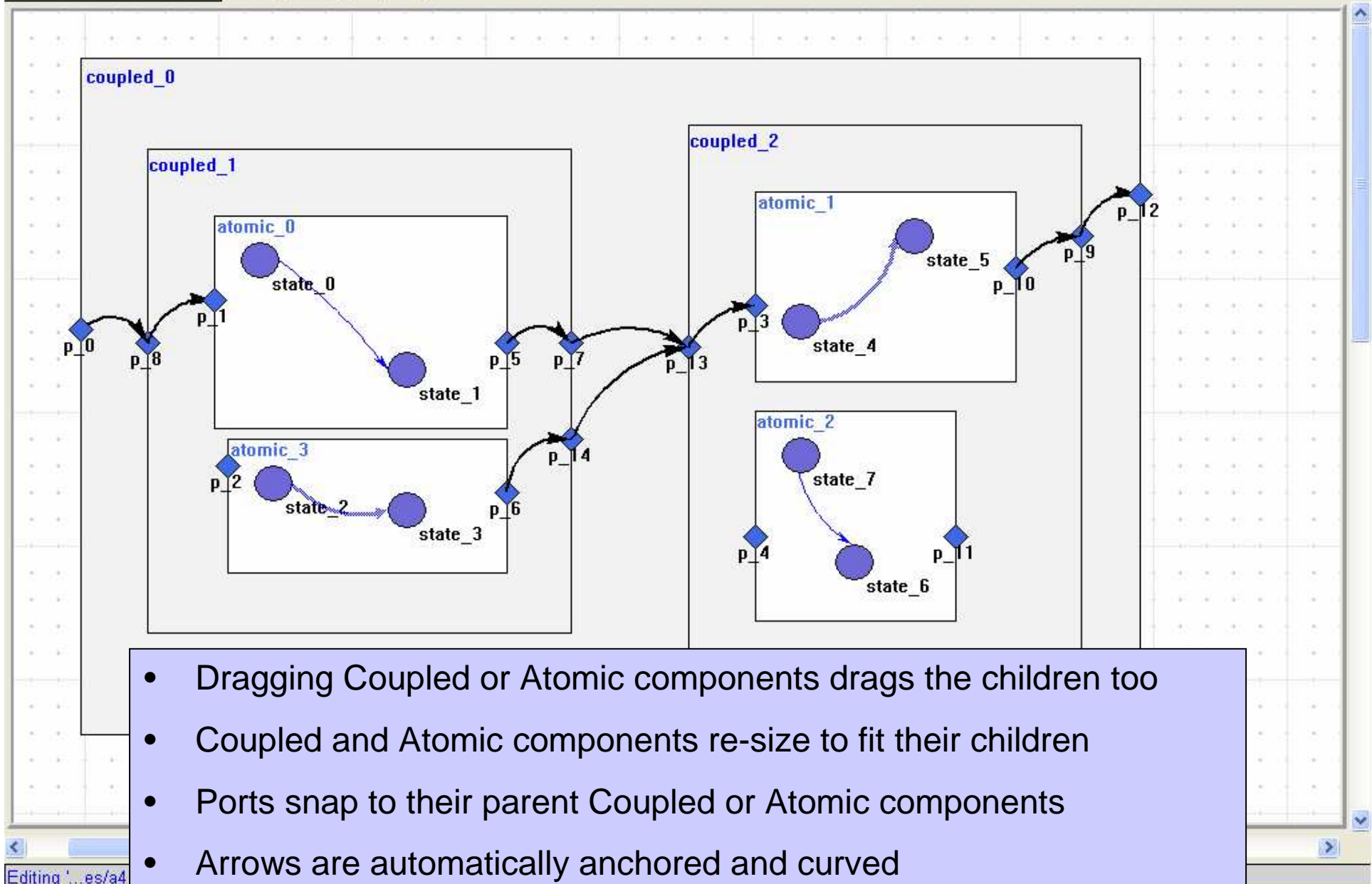
New atomicDevsV2

New coupledDevsV2

New portDevsV2

New stateDevsV2

GenerateCode



- Dragging Coupled or Atomic components drags the children too
- Coupled and Atomic components re-size to fit their children
- Ports snap to their parent Coupled or Atomic components
- Arrows are automatically anchored and curved

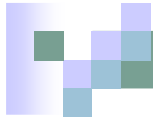
The screenshot displays the AToM3 software interface. At the top, the title bar reads "74 AToM3 v0.3 using: EntityRelationshipV3 + DevsV2". Below the title bar is a toolbar with icons for "Entity", "rel.", "EDIT", and "GEN". To the right of the toolbar are several buttons: "New atomicDevsV2", "New coupledDevsV2", "New portDevsV2", "New stateDevsV2", and "GenerateCode".

The main workspace shows a diagram with several components: "coupled_0", "coupled_1", and "ato". The diagram includes ports labeled "p_0", "p_8", "p_1", and "p_2". A dialog box titled "Editing ASG_DevsV2" is open in the foreground, containing the following fields and options:

- nameDevsV2: Demo
- authorDevsV2: Denis Dube
- Set Spaces Per Tab: 4
- Text Editor Menu
- descriptionDevsV2: # I'm only demonstrating the Layout Options in this screenshot...
- arrowOptimization:
 - None
 - Ports only
 - Straight Arrows
 - Bezier Arrows

At the bottom of the dialog box are "OK" and "Cancel" buttons. A status bar at the bottom of the window shows "Editing '...es/a4_MDL.py' (modified)" and "FTA needed 1 iterations to find stable layout".

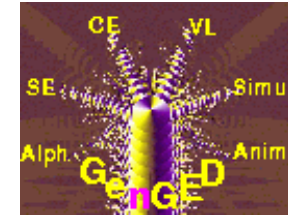
- Model level attributes can also be set, including optimization level, to trade beauty for performance



Overview

- Introduction to DEVS
- Round 1: Basic Diagram Editor
- Round 2: The Visual Modeling Environment
- **Round 3: Generating PyDEVS code**
- Conclusion and Future Work

Code Generation

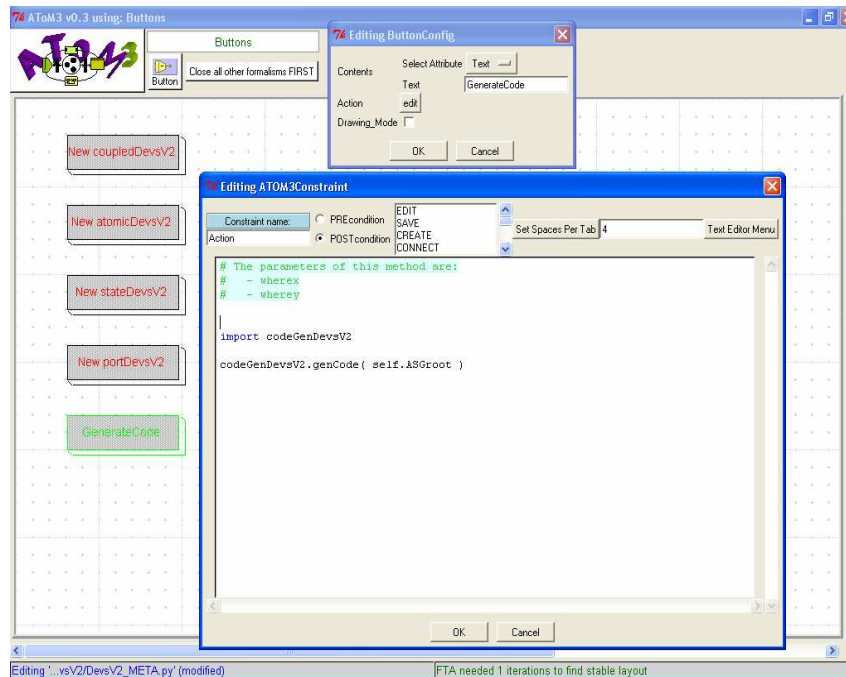


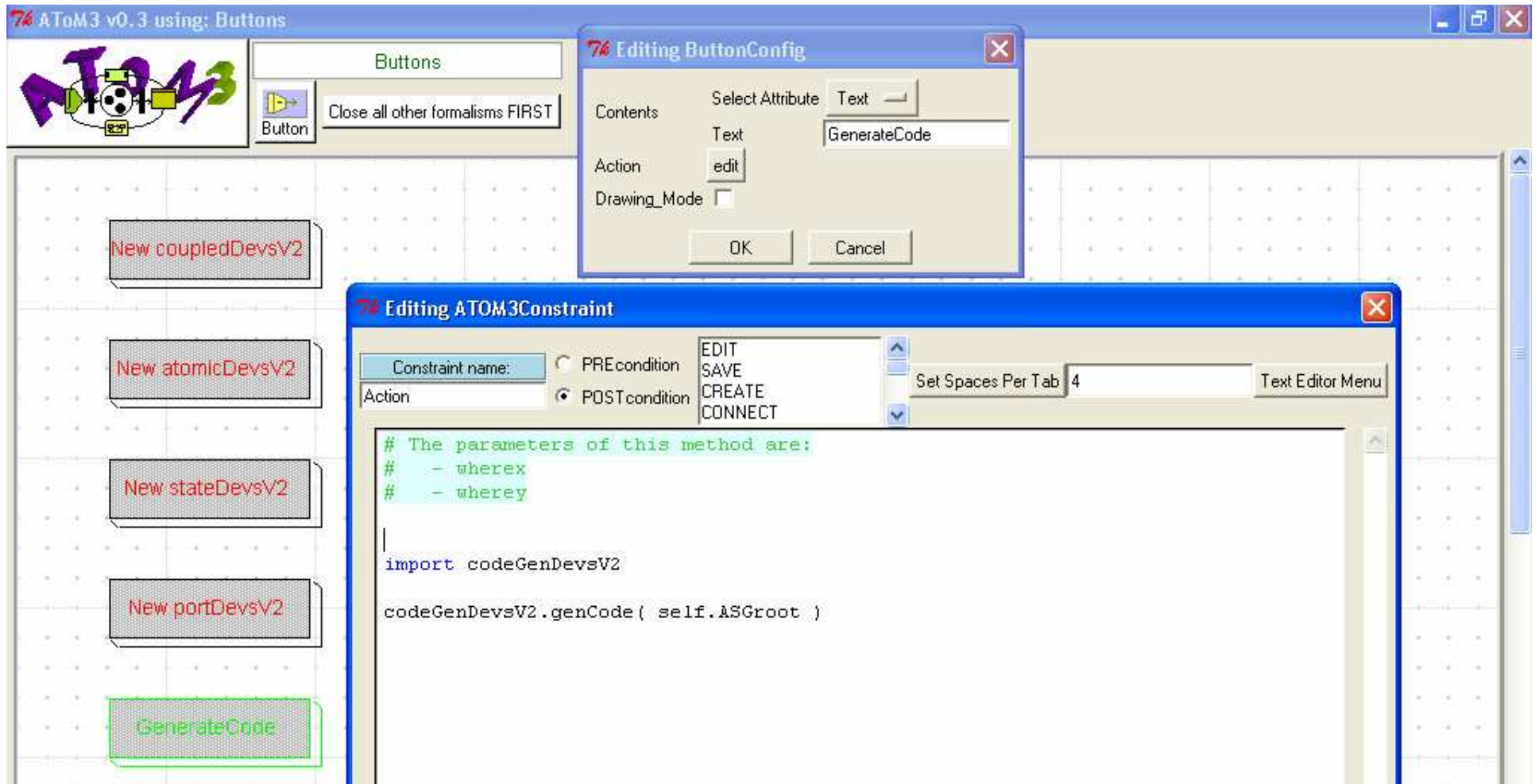
- No native support for generation of code from a model
- The ability to specify the generation of arbitrary text to a file as a side affect of running a grammar rule can probably be added to the system without much difficulty
- In light of the poor practical performance of the generated editor, I chose not to attempt this

Code Generation



1. Write out Python code
2. Add it to the buttons model for DEVS





1. Import arbitrary code
2. Run it with the root of our graph as a parameter
3. Code generator can now simply traverse all the nodes
 - ALTERNATIVE: Can use graph grammars too, indeed, the buttons model we are editing here was automatically generated from the ER Model specification using a very simple graph grammar that matches ER Entities once each

Code Generation



■ Or...

- Just adapt code from:
Ernesto Posse



- Modified to be **platform independent**
- Copies the **PyDEVs** simulator to the generation target
 - **PyDEVs** by: Jean-Sébastien Bolduc
and Hans Vangheluwe

DEVSTEST

File Edit View Favorites Tools >> Address D:\ResearchSummer2004\atom3 user area\User Formalisms\DevsV2\DEVSTEST

Back Search Folders

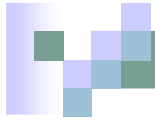
Folders

Name	Size	Type	Date Modified
atomic_0_devs_model		File Folder	3/21/2005 1:01 AM
atomic_1_devs_model		File Folder	3/21/2005 1:01 AM
coupled_1_devs_model		File Folder	3/21/2005 1:01 AM
coupled_0.py	1 KB	Python File	3/21/2005 1:01 AM
DEVS.py	16 KB	Python File	11/12/2002 10:56 AM
DEVS.pyc	13 KB	Compiled Python File	3/21/2005 1:01 AM
Simulator.py	13 KB	Python File	11/12/2002 10:56 AM
Simulator.pyc	10 KB	Compiled Python File	3/21/2005 1:01 AM

atom3 user area

- TrafficGrammar
- User External
- User Formalisms
 - classDiagramsV2
 - devs
 - DevsV2
 - DEVSTEST**
 - atomic_0_devs_model
 - atomic_1_devs_model
 - coupled_1_devs_model
 - atomic_2_devs_model
 - pyDEVS
 - DFDModels
 - EntityRelationship2
 - EntityRelationshipV4
 - HybridSystems
 - lambda
 - lambda2
 - lib
 - madBoxTest
 - NFA
 - Traffic
 - DevsV2.zip
 - EntityRelationship2.zip
 - EntityRelationshipV3.zip
- User Models
- Atom3TrafficEdition
- Documents
- GraphViz
- GraphViz
- IconEditor
- IEEE
- JGraphPad
- Web
- yed-2.2
- atom3.zip
- atom3_2005_03_18.zip

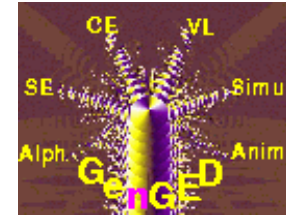
- Is it not nifty? Even the generated code is hierarchical!



Overview

- Introduction to DEVS
- Round 1: Basic Diagram Editor
- Round 2: The Visual Modeling Environment
- Round 3: Generating PyDEVS code
- Conclusion and Future Work

Conclusion



- GenGED is **high on concept** and has many interesting ideas, particularly concerning layout
 - I believe AToM³ could definitely benefit from the integration of a similar layouting tool
 - In particular, it makes the generation of a prototype diagram editor very easy, even for non-experts
- Unfortunately, this tool is low on implementation
 1. Very **limited** platform support
 2. Interface **not suited** for most practical applications
 3. **No native support** for generating code

Conclusion

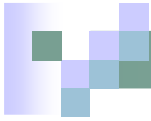


- **Very flexible**, at the cost of some **manual coding**
- API keeps growing, so more and more high level methods such as
 - scaling to fit x
 - hierarchical selection/dragare being made available to formalism creators
- Buggy (but getting better :D)
- No tutorials!?! (soon!)

Future Work



- The generated DEVS tool can be improved by:
 - Adding hierarchical hiding
 - Adding N atomic/coupled components
 - Adding special state/atomic component that is not as visual but doesn't limit the expressivity of DEVS
 - Adding hierarchical force transfer (overlap)
 - Testing with a meaningful DEVS example



References (1/4)

- **DEVS Today: Recent Advances in Discrete Event Based Information Technology**

- Author: Bernard P. Zeigler
- MASCOTS' 03, Orlando, FL, October 2003
- http://www.lsis.org/vie_du_labo/uploads/Recent_advances_in_discrete_ev_36.ppt&e=7620

- **DEVS TUTORIAL**

- Authors: John Kitzinger and Prasanna Sridhar
- <http://vlab.unm.edu/documents/Tutorial1.ppt&e=7620>



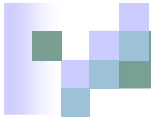
References (2/4)

- **pythonDEVS**
 - **Authors:** Hans Vangheluwe, Jean-Sébastien Bolduc, Ernesto Posse, Spencer Borland
 - <http://moncs.cs.mcgill.ca/MSDL/research/projects/DEVS/>
- **Generation of DEVS modelling and simulation environments**
 - **Authors:** Ernesto Posse and Jean-Sébastien Bolduc
 - In A. Bruzzone and Mhamed Itmi, editors, *Summer Computer Simulation Conference. Student Workshop*, pages S139 - S146. Society for Computer Simulation International (SCS), July 2003. Montréal, Canada.
 - <http://www.cs.mcgill.ca/~hv/publications/03.SCSC.DEVScodegen.pdf>
- **Domain-Specific Modelling for analysis and design of traffic networks**
 - **Authors:** Hans Vangheluwe and Juan de Lara
 - *Winter Simulation Conference*, pages 249 - 258. IEEE Computer Society Press, December 2004. Washington, DC.
 - <http://www.cs.mcgill.ca/~hv/publications/04.Wintersim.Traffic.pdf> 71



References (3/4)

- Sencario Views for Visual Behavior Models in GenGED
 - Authors: C. Ermel and R. Bardohl
 - Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'02), Satellite Event of First Int. Conference on Graph Transformation (ICGT'02), Barcelona, Spain, Oct. 2002, pages 71-83
 - http://www.tfs.cs.tu-berlin.de/~rosi/publications/EB02_gtVMT.ps.gz
- A Generic Graphical Editor for Visual Languages based on Algebraic Graph Grammars
 - Author: Roswitha Bardohl
 - Proc. IEEE Symposium on Visual Languages (VL'98), Sept.1998, Halifax, Canada, pages 48-55
 - http://www.tfs.cs.tu-berlin.de/~rosi/publications/Bar98_VL98.ps.gz
- GenGED - A visual definition tool for visual modeling environments
 - Authors: Bardohl,R., Ermel,C., and Weinhold,I.
 - Proc. Application of Graph Transformations with Industrial Relevance (AGTIVE'03), pages 407-414, Sept./Oct., 2003, Charlottesville/Virgina, USA. Also in Lecture Notes in Computer Science (LNCS) **3062**, Springer, 2004, pages 413-419
 - http://www.tfs.cs.tu-berlin.de/~rosi/publications/BEW03_AGTIVE03.ps.gz



References (4/4)

- Conceptual Model of the Generic Graphical Editor GenGEd for the Visual Definition of Visual Languages
 - Authors: Bardohl,R. and Ehrig,H.
 - Lecture Notes in Computer Science (LNCS) **1764**: Theory and Application of Graph Transformation (TAGT'98), Springer 1999, pages 252-266
 - http://www.tfs.cs.tu-berlin.de/~rosi/publications/BE99_TAGT98_Lncs.ps.gz
- Scenario Animation for Visual Behavior Models: A Generic Approach Applied to Petri Nets
 - Authors: Bardohl,R. and Ermel,C.
 - Proc. 10th Workshop on Algorithms and Tools for Petri Nets (AWPN'03) Sept. 2003, Eichstätt-Ingolstadt, Germany.
 - http://www.tfs.cs.tu-berlin.de/~rosi/publications/BE03_AWPN.ps.gz
- Specifying Visual Languages with GenGED
 - Authors: Bardohl,R., Ehrig,K., Ermel,C., Qemali,A. and Weinhold,I.
 - Proc. APPLIGRAPH Workshop on Applied Graph Transformation (AGT'02), Satellite Event of ETAPS 2002, Grenoble, France, April 12-13, 2002, pages 71-82
 - http://www.tfs.cs.tu-berlin.de/~rosi/publications/BEEQW02_AGT.ps.gz