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)
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
Atomic Models

Ordinary Differential Equation Models

Processing/Queuing/Coordinating

Petri Net Models

Spiking Neuron Models

Stochastic Models

Quantized Integrator Models

Fuzzy Logic Models

Discrete Time/StateChart Models

Reactive Agent Models

Coupled Models

Networks, Collaborations

N-Dim Cell Space

Processing Networks

Spiking Neuron Networks

Physical Space

Cellular Automata

Self Organized Criticality Models

Multi Agent Systems

Partical Differential Equations

DEVS Expressability

can be components in a coupled model
DEVS notation

\[ M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \tau_a \rangle \]

where

- \( X \): set of input values
- \( S \): set of states
- \( Y \): set of output values
- \( \delta_{\text{int}} \): Internal transition function
- \( \delta_{\text{ext}} \): External transition function
- \( \lambda \): Output Function
- \( \tau_a \): Time advance function
How DEVS works

- System is in State $s$
- If no external event ($\delta_{ext}$) 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 ($\delta_{ext}$) 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 $ta$ 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
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

<table>
<thead>
<tr>
<th>Entity3</th>
</tr>
</thead>
<tbody>
<tr>
<td>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=</td>
</tr>
</tbody>
</table>
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
Alphabet editor

- Connection Editor
  - Specify relationship between different entities
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
• Note: the cardinalities are consistent with UML class notion of cardinalities but these are enforced at run-time
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)
Connection ports that arrows will automatically lock on to
Entity Relationship

- We have now specified everything save any notion of layout
  - This is sufficient to generate a diagram editor and test our prototype
1. The relations are specified in just two clicks, and drawing entities requires at most one click in the toolbar and one on the canvas.

2. This was difficult to draw, since there are no layout constraints yet, and selecting something brings it to the foreground… (must select everything else to bring THEM to the foreground if that happens)
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 rules
  - Or
  - 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
• 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
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 coupledDevsV2 in the LHS.

Perhaps an error was made in the Alphabet editor…
• Indeed, this insert rule for portDevsV2 is complete nonsense
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
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.
Pimping out the ER

- Add Constraints/Actions code → Layout!
- 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
• 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
• 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 effect 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
• 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/drag
  are 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

- 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

- **Generation of DEVS modelling and simulation environments**
  - Authors: Ernesto Posse and Jean-Sébastien Bolduc

- **Domain-Specific Modelling for analysis and design of traffic networks**
  - Authors: Hans Vangheluwe and Juan de Lara
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

- A Generic Graphical Editor for Visual Languages based on Algebraic Graph Grammars
  - Author: Roswitha Bardohl

- GenGED - A visual definition tool for visual modeling environments
  - Authors: Bardohl,R., Ermel,C., and Weinhold,I.
  - 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

- **Scenario Animation for Visual Behavior Models: A Generic Approach Applied to Petri Nets**
  - Authors: Bardohl, R. and Ermel, C.

- **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