Vergil: Component-Based Design Environment

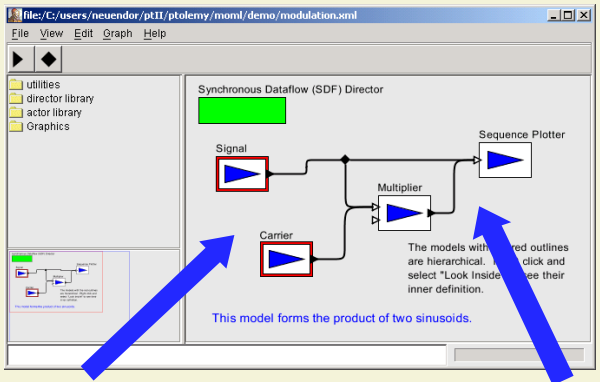
Steve Neuendorffer

Ptolemy Miniconference
Berkeley, CA, March 22-23, 2001

Graph Editor

editable model visualization

A library of actors and directors



hierarchical models

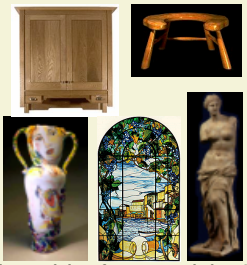
atomic models

Why Component Based?

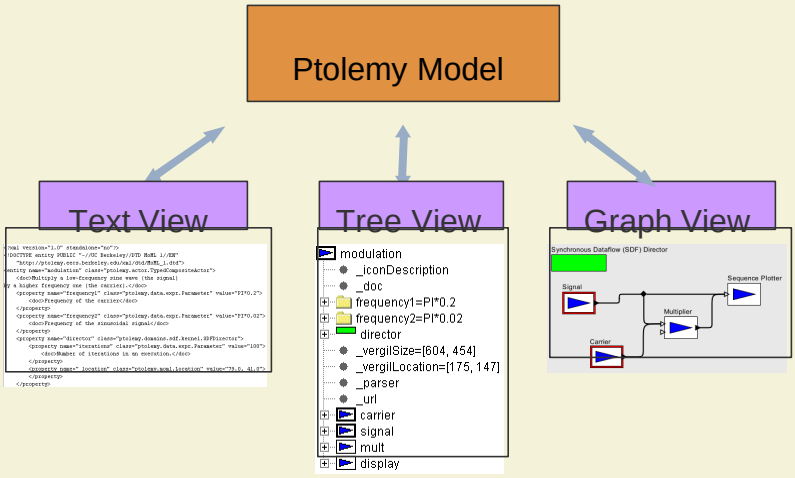
Design Environment

Design Tools

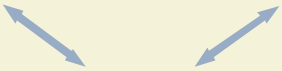
Design Artifacts



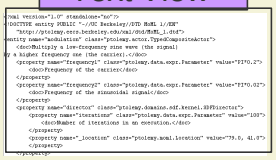
Digital Design Tools



Digital Design Artifacts



Text View



Design Environment Configuration

- Given a file, what type of artifact does the file contain?
- Given a digital artifact, what type of tool can manipulate it?
- Vergil configuration uses a Ptolemy Model!

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE entity PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">
<entity name="configuration" class="ptolemy.actor.gui.Configuration">
  <doc>Configuration to edit and run Ptolemy II models</doc>
  <!-- The directory of open models, each represented by an effigy. -->
  <entity name="directory" class="ptolemy.actor.gui.ModelDirectory"/>
  <!-- Factories for effigies. -->
  <entity name="effigyFactory" class="ptolemy.actor.gui.EffigyFactory">
    <entity name="Graph Editor" class=" ptolemy.actor. gui.PtolemyEffigy$Factory">
      <entity name="blank" class="ptolemy.actor.TypedCompositeActor"/>
    </entity> (etc.)
```

Effigies



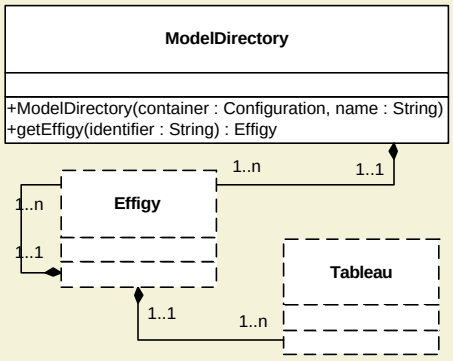
- An effigy represents an open design artifact
- Subclasses contains design artifacts, such as a Ptolemy model or an HTML string

Tableaux

- Every design tool displaying an Effigy is represented by a Tableau.
- Subclasses for the Graph Editor, Icon Editor, etc.

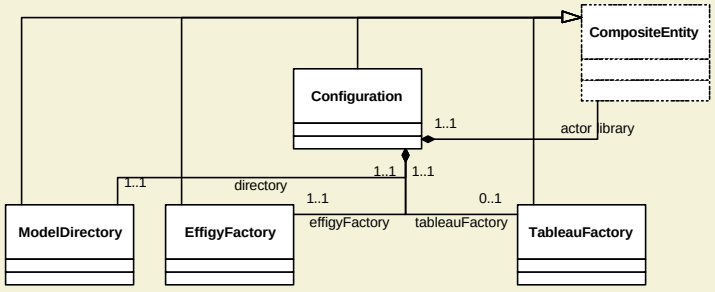
Model Directory

- The model directory contains all of the open effigies and tableaux.
- Effigies are indexed from the location they were loaded from.

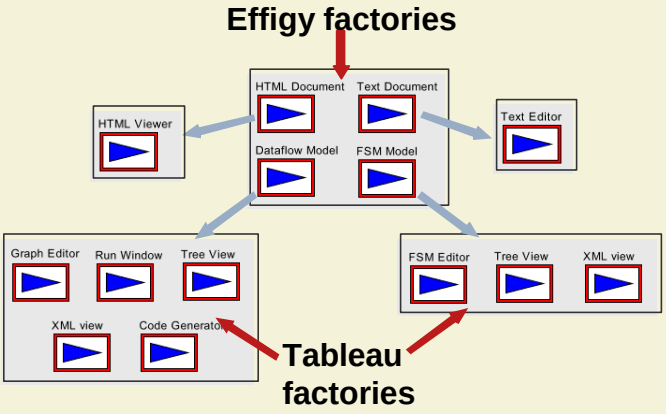


Configurations

- EffigyFactories create effigies for a file.
- TableauFactories create tableaux for an effigy.
- Alternate configurations customize interaction and integrate new tools.



Vergil Configuration



But how do we build the design tools themselves?

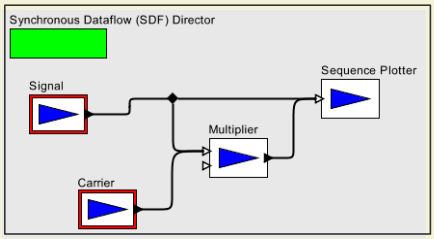
MoML

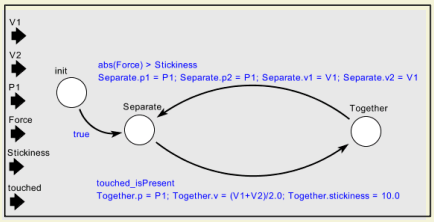
- Concrete Syntax in XML for describing component architectures
- Support for class derivation and modification
- Represents all Ptolemy models, including the Vergil configuration!

```
<!ELEMENT class (class | configure | deleteEntity |
deletePort | deleteRelation| director | doc | entity | group | import
| input | link | port | property | relation | rename | rendition |
unlink)*>
<!ATTLIST class    name CDATA #REQUIRED extends CDATA
 #IMPLIED source CDATA #IMPLIED>
<!ELEMENT configure (#PCDATA)>
<!ATTLIST configure source CDATA #IMPLIED>
<!ELEMENT deleteEntity EMPTY>
<!ATTLIST deleteEntity name CDATA #REQUIRED>
<!ELEMENT deletePort EMPTY>
<!ATTLIST deletePort name CDATA #REQUIRED>
<!ELEMENT deleteProperty EMPTY>
<!ATTLIST deleteProperty name CDATA #REQUIRED>
<!ELEMENT deleteRelation EMPTY>
<!ATTLIST deleteRelation name CDATA #REQUIRED>
<!ELEMENT doc (#PCDATA)>
<!ATTLIST doc name CDATA #IMPLIED>
<!ELEMENT entity (class | configure | deleteEntity | deletePort
|deleteRelation | director | doc |entity | group | import | input | link
| port |property | relation | rename | rendition | unlink)*>
<!ATTLIST entity   name CDATA #REQUIRED
    class CDATA #IMPLIED source CDATA #IMPLIED>
<!ELEMENT group ANY>
<!ATTLIST group name CDATA #IMPLIED>
<!ELEMENT input EMPTY>
<!ATTLIST input source CDATA #REQUIRED>
<!ELEMENT link EMPTY>
<!ATTLIST link insertAt CDATA #IMPLIED
            port CDATA #REQUIRED
            relation CDATA #REQUIRED
            vertex CDATA #IMPLIED>
<!ELEMENT port (configure | doc | property | rename)*>
<!ATTLIST port class CDATA #IMPLIED
            name CDATA #REQUIRED>
<!ELEMENT property (configure | doc | property | rename)*>
<!ATTLIST property class CDATA #IMPLIED
             name CDATA #REQUIRED
             value CDATA #IMPLIED>
<!ELEMENT relation (configure | doc | property | rename | vertex)*>
<!ATTLIST relation name CDATA #REQUIRED
            class CDATA #IMPLIED>
<!ELEMENT rename EMPTY>
<!ATTLIST rename name CDATA #REQUIRED>
<!ELEMENT unlink EMPTY>
<!ATTLIST unlink index CDATA #IMPLIED insideIndex CDATA
 #IMPLIED port CDATA #REQUIRED relation CDATA
 #REQUIRED>
<!ELEMENT vertex (configure | doc | location | property | rename)*>
<!ATTLIST vertex name CDATA #REQUIRED
            pathTo CDATA #IMPLIED
            value CDATA #IMPLIED>
```

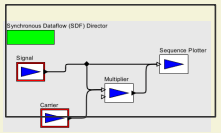
Graph Visualization in Diva

- Interactors specify how mouse events are handled.
- Renderers specify how nodes and edges appear.
- Other nice things:
  - Context Menus
  - Automatic Layout
  - Cut and Paste
  - Printing
  - Direct Manipulation



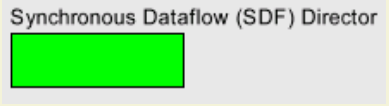


Direct Manipulation Editing



MoMLChangeRequest

```
<port name="input"/>
```



- Specified in MoML
- Requests are queued and executed when possible
- Graph Editor listens for mutations

Allows dynamically edit executing models!

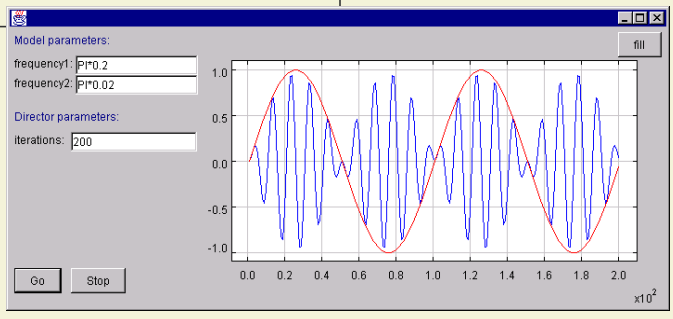
Graph Editor Configuration

- Java Interfaces
  - Placeable actors
- Attributes of the Vergil Configuration
  - Actor libraries
  - Parameter Styles
- Attributes of the Model
  - Parameter Styles

User Interfaces for Actors

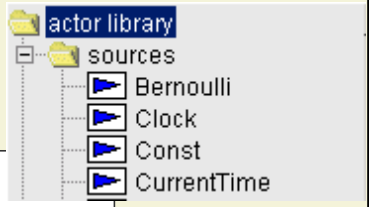
Actors with specialized interaction implement the Placeable interface.

```
public class Plotter extends TypedAtomicActor
    implements Configurable, Placeable {
      public void place(Container container) {
      }
    }
```



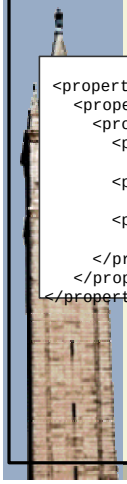
Actor Libraries

Objects specified in the Configuration are available in the Palette.



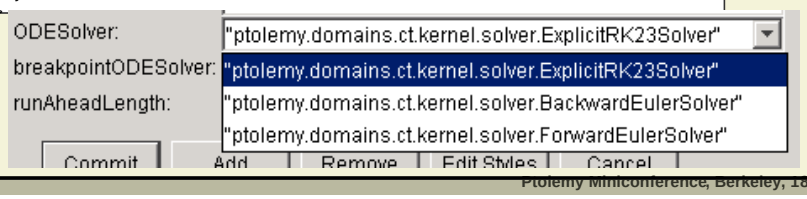

```
<entity name="sources"
          class="ptolemy.moml.EntityLibrary">
     <doc>Domain-polymorphic sources</doc>
     <entity name="Bernoulli"
          class="ptolemy.actor.lib.Bernoulli"/>
     <entity name="Clock"
          class="ptolemy.actor.lib.Clock"/>
     <entity name="Const"
          class="ptolemy.actor.lib.Const"/>
     <entity name="CurrentTime"
     class="ptolemy.actor.lib.CurrentTime"/>
     ..
</entity>
```


Parameter Styles



Attributes in a model, or the configuration specify the style of parameter editing.

```
<property name="CT" class="ptolemy.domains.ct.kernel.CTMixedSignalDirector">
  <property name="ODESolver">
    <property name="style" class="ptolemy.actor.gui.style.EditableChoiceStyle">
      <property name="choice0" class="ptolemy.data.expr.Parameter"
         value="&quot;ptolemy.domains.ct.kernel.solver.ExplicitRK23Solver&quot;"/>
      <property name="choice1" class="ptolemy.data.expr.Parameter"
         value="&quot;ptolemy.domains.ct.kernel.solver.BackwardEulerSolver&quot;"/>
      <property name="choice2" class="ptolemy.data.expr.Parameter"
         value="&quot;ptolemy.domains.ct.kernel.solver.ForwardEulerSolver&quot;"/>
    </property>
  </property>
</property>
```

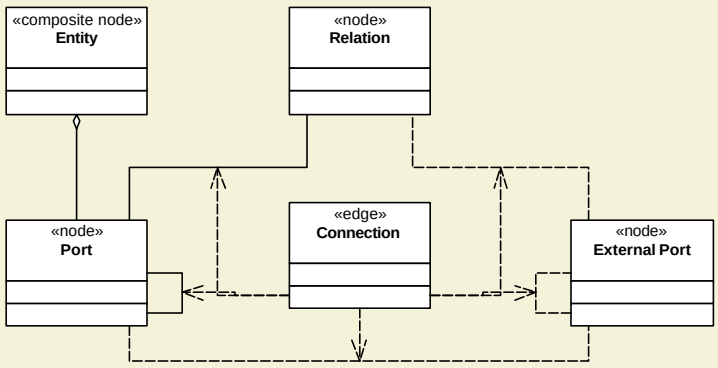


Current Research

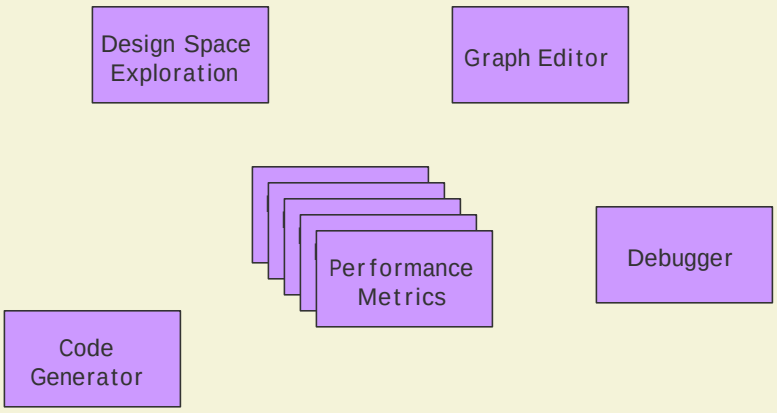
- Meta-Modeling of diagrams.
  - Such as in Dome, GME, Moses.
- Applying Models of Computation to interacting design tools
  - More than just Design Flow Management.

Meta-Modeling

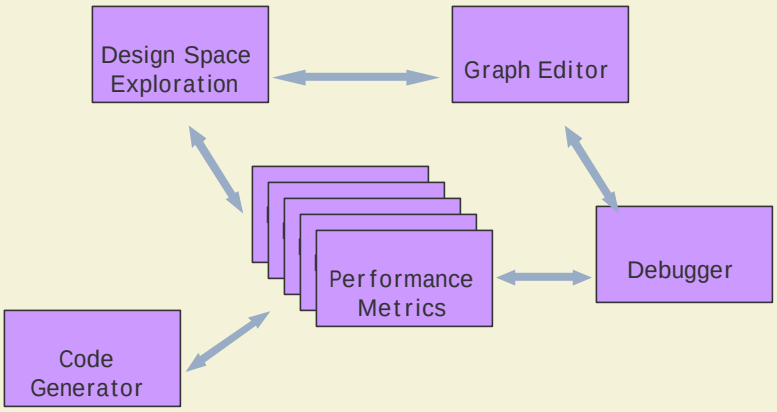
- Diagram structure specified using a description language.



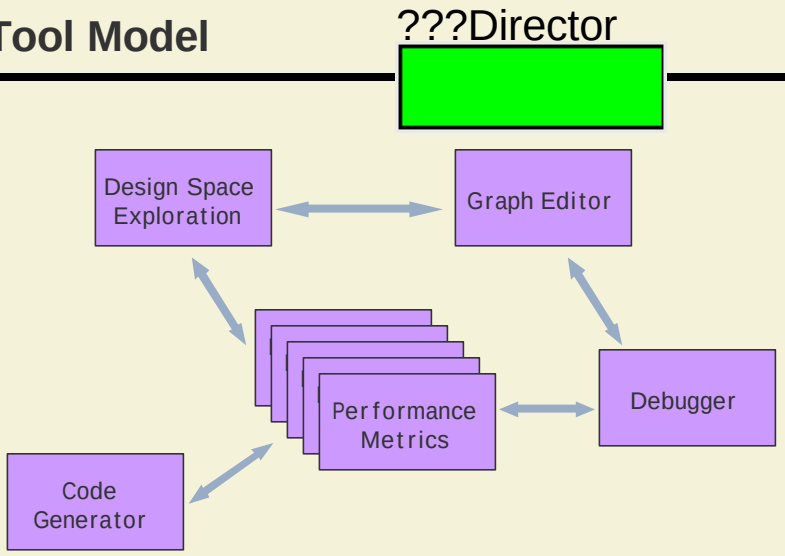
Tool Configuration



Tool Interaction



Tool Model



Conclusion

- An extensible design environment for integrating arbitrary digital design tools
  - Configuration is a Ptolemy model
- Creating digital tools should be simple
  - Especially graphical, direct manipulation tools
- User Interfaces could be a rich new area for the application of models of computation
  - JavaBeans done right