Comparison of Several Meta-modeling Tools

Yi Lu
Computer Science Department
McGill University
1.12.2003
Outline

- Why not modeling
- Meta-modeling
- MetaEdit+
- DOME
- Comparison AToM3 with MetaEdit+ and DOME
Why not modeling?

- Modeling is the process to build a framework that can reflect systems in the real world, based on certain modeling formalism which defines all the models that can be created using the modeling environment.

- The modeling formalism contains two parts:
  - Syntax: specify the concepts used to construct models, the relationships existing among these concepts and the rules that govern the construction of models.
  - Semantics: be further divided into several kinds: denotational semantics, operational semantics. Correctness of semantics is a key issue, usually a form of a separate constraint language, OCL (Object Constraints Language)
Why not modeling

Modeling tools usually have a two-level architecture as shown below:

Disadvantages of modeling:

This unchangeable feature causes some problems:

1. Fixed model can’t specify domains under development adequately
2. Can’t satisfy the requirements of users with different culture, education and characteristics
Why not modeling?

- Solution: to customize the models to the requirements of the domains and the users
  - To come up a general formalism that can satisfy “everything”: tricky
  - To allow the users to specify their own models for specific domains, when the domain develop or user’s requirements change, the users can update the models. That’s what meta-modeling technique provides…
Meta-modeling

- Meta-modeling provides a modeling environment that is configurable for a wide range of domains:
  - use a higher-level, meta-level modeling language to define a domain-specific modeling language instead of models.
  - specification of the modeling language is a model called meta-model.
  - meta-modeling language should be abstract enough such that it can specify most domains. As a consequence, it can describe itself in the form of meta-meta-model.
Implementation of meta-modeling

The key is the lower layer is always defined by the language of the next upper layer.
Meta-modeling tools

- AToM3
- MetaEdit+
- DOME
- GME
- KOGGE

.......

first three, based on Petri Net example
MetaEdit+

- A commercial metaCASE tool developed by the company MetaCase Consulting, Finland.
- Many successful stories:
  - development of mobile phones for NOKIA,
  - development of e-commerce platform for PECUNET
- Platforms: Windows, Linux, HP, Solaris
- It consists of two parts:
  - Method Workbench: design your own method (including the modeling formalism, code generators, and other concepts)
  - MetaEdit+: design your own model in the modeling environment following a given method definition.
MetaEdit+

Meta-modeling language GOPRR:

- **Object**: entity in ER formalism: place and transition
- **Property**: attribute
- **Relationship**: tran2pl, pl2tran
- **Role**: specify the components that lie at the end of a relationship connected to an object. Two roles: From and To
MetaEdit+

- Graph: connects individual components together to form the modeling technique using some sub-tools:
  - Type tool: what types are visible in the Type menus and Toolbar of editors
  - Binding tool: how the relationships, roles and objects are connected to each other, cardinality attribute.
  - Constraint tool: the number of roles or relationships an object can participate in, no specific constraint language
  - Sub-graph tool:
    explosion: explode an instance to a new graph: entity in ER -> state transition diagram
    decomposition: decompose an object type to a new graph in the same diagram.
MetaEdit+

- Code generation: has its own report definition language.
  - Document generation: HTML, word document.
  - Source code generation: test cases, simulation.

Petri Net example
DOME

- Under development of Honeywell Technology Center
- Written in VisualWorks Smalltalk
- Platforms: Windows, Linux, Sun Solaris
- Tools:
  - ProtoDOME: execute the meta-model and generate the modeling environment.
  - Alter and Projector: two extension languages used for constraint specification, code generation, simulation and test cases, etc.
The DOME Tool Specification Language:

• Node Specification: entity in ER formalism

• Connector Specification: relationship in ER formalism

• Connection Constraint: determine how the connectors can be used to link nodes together, cardinalities.

• Part/Whole Relationship: containment. token and place

• Graph Specification: represent the class of object that contains the nodes and connectors in a diagram, to declare graph-level properties, relationships and methods
DOME

- **Tool Palette**: the toolbar with the buttons for creating the objects defined in DOME Tool Specification, automatically generated by DOME.

- **Custom Button**: can only be put in Tool Palette, used to specify test cases and simulation, through defining methods in language Alter.

- **Code generation**: Alter

  **Petri Net example**
## Comparison AToM3 with MetaEdit+ and DOME

<table>
<thead>
<tr>
<th>Aspects</th>
<th>AToM3</th>
<th>MetaEdit+</th>
<th>DOME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platforms</td>
<td>Windows, Unix</td>
<td>Windows, Unix, Sun Solaris, HP</td>
<td>Windows, Linux, Sun Solaris</td>
</tr>
<tr>
<td>Meta-modeling language</td>
<td>ER</td>
<td>GOPRR</td>
<td>The DOME Tool Specification language</td>
</tr>
<tr>
<td>Graphical specification?</td>
<td>Yes</td>
<td>No</td>
<td>Partly, the graphical appearance can't be edited in a graphical way</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Partly, not implement complete yet</td>
<td>Yes, decomposition</td>
<td>Yes, sub-diagram</td>
</tr>
<tr>
<td>Inheritance</td>
<td>No</td>
<td>Yes, ( make dependant )</td>
<td>Yes</td>
</tr>
<tr>
<td>Constraint</td>
<td>Python function or OCL</td>
<td>No specific constraint language</td>
<td>Alter language</td>
</tr>
<tr>
<td>Simulation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Simulation method and implementation workload</td>
<td>Graph Grammar, an intuitive way, less code by hand</td>
<td>Report definition language, all code by hand</td>
<td>Alter function, all code by hand</td>
</tr>
<tr>
<td>Code generation and workload</td>
<td>Python source code</td>
<td>Can be any language</td>
<td>Can be any language</td>
</tr>
<tr>
<td></td>
<td>Little code by hand</td>
<td>Most code by hand</td>
<td>Most code by hand</td>
</tr>
<tr>
<td>Report generation</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
The following work

- Other meta-modeling tools: GME, KOGGE...