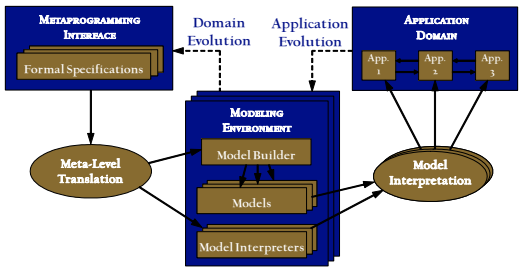
A Paradigm for Teaching Modeling Environment Design

Berkeley University of California
Jonathan Sprinkle, James Davis, Greg Nordstrom
Vanderbilt University

Abstract

Model-based design is easy to praise, but non-trivial to follow, especially when designing from scratch! We describe a paradigm for instructing students in the art and science of Model-Integrated Computing (MIC), as well as a motivating example which demonstrates the phases of the design of a modeling environment.



0. Basic Concepts

- ***Module Interconnection***: associate components through interfaces
- ***Aspects***: multiple views or slices of the same model
- ***Hierarchy***: encapsulation and hiding through containment
- ***Object association***: association between objects
- ***Specialization***: object refinement through inheritance

1. Domain Knowledge

Before creating a modeling environment, the domain concepts must be understood;

Example: Kinds of Publications

Domain knowledge: *There are many kinds of publications, but they all have certain common characteristics. Use domain knowledge obtained from years of thinking and best-practices in*

Example: Authors of publications

Domain knowledge: *Publications have authors, and authors are typically associated with more than one publication. Consider how to reuse authors across publications, and avoid re-entering information*

2. Concepts Encoding

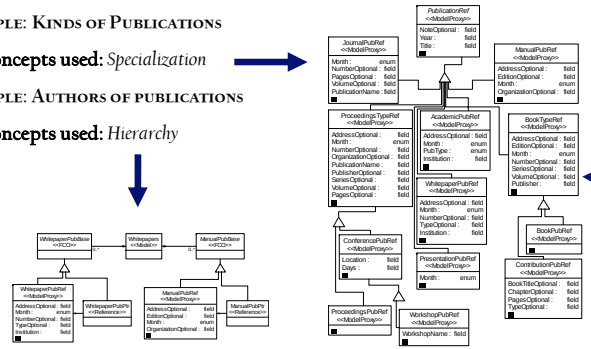
Once the concepts are understood the domain concepts are mapped using the basic concepts of modeling

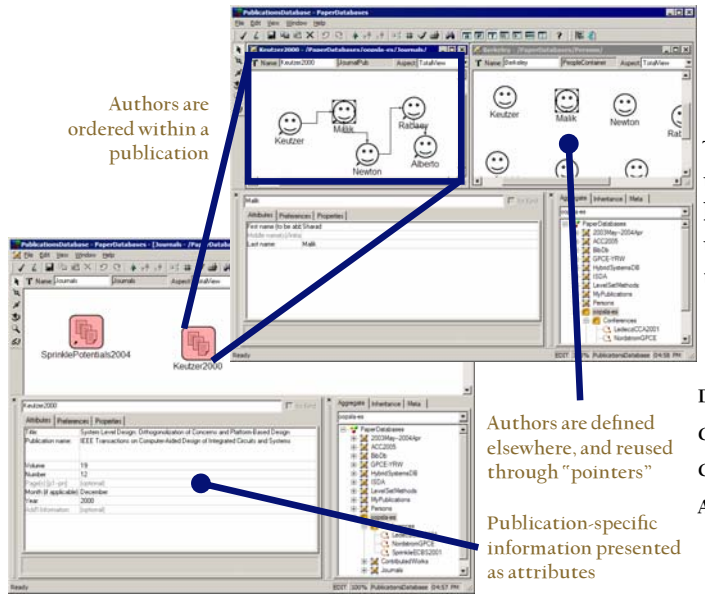
Example: Kinds of Publications

Concepts used: *Specialization*

Example: Authors of publications

Concepts used: *Hierarchy*



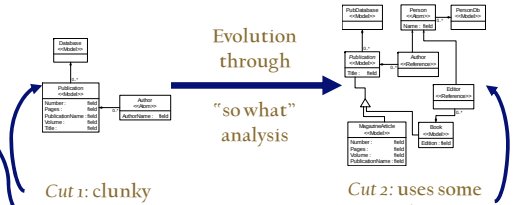


3. Translators & 4. Artifacts

Models are only useful if they can be translated or interpreted into some other useful form.

Example: Generate database

Example: Generate HTML version of bibliography



"So What?"

The designer learns that there is a difference between a useful modeling environment, and one for the sake of having one. Designs evolve, and a "fixed-point" of usability vs. overhead is reached. Metamodeling reduces the overhead of creating the GUI and syntax parser.

For More Information

Download the tool: http://www.eecs.berkeley.edu/~sprinkle/oopsla/

GME Information: http://www.isis.vanderbilt.edu/

Chess website: http://chess.eecs.berkeley.edu/

Authors: Jonathan Sprinkle (sprinkle@EECS.Berkeley.Edu)
James Davis (james.davis@vanderbilt.edu)
Greg Nordstrom (greg.nordstrom@rfmcentral.com)