An Introduction to Metamodelling

Principles & Fundamentals

Department of Computer Science,
The University of York, Heslington, York YO10 5DD, England
http://www.cs.york.ac.uk/~paige

THE UNIVERSITY of York

Context of this work

- The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (http://www.modelware-ist.org/).
- Co-funded by the European Commission, the MODELWARE project involves 19 partners from 8 European countries. MODELWARE aims to improve software productivity by capitalizing on techniques known as Model-Driven Development (MDD).
- To achieve the goal of large-scale adoption of these MDD techniques, MODELWARE promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 license.

Metamodelling

- A controversial topic, and one that is currently critical within the UML/OMG/MDA community.
- A metamodel is just another model (e.g., written in UML).
- Metamodels are examples of domain-specific models.
  - Other example domains: real-time systems, safety critical systems, e-business.
- The domain of metamodelling is language definition.
- Thus, a metamodel is a model of some part of a language.
  - Which part depends on how the metamodel is to be used.
  - Parts: syntax, semantics, views/diagrams, ...

Uses for a Metamodel

- For defining the syntax and semantics of a language.
- To explain the language.
- To compare languages rigorously.
- To specify requirements for a tool for the language.
- To specify a language to be used in a meta-tool (e.g., XMF).
- To enable interchange between tools.

Language Design

- How would you go about designing a programming language?
  1. What sort of programs do you want to allow programmers to create? (ie., user requirements).
  2. Define a syntax (eg., EBNF).
  3. Define semantics using structural induction over the constructs of the language, e.g., what do while, if, ;, etc all mean?
  4. Implement a compiler and libraries.
  5. Implement supporting tools.
  6. Build your killer app.
- In doing so, you would follow well-known principles of programming language design.

Programming Language Design

- The primary purpose of a programming language (PL) is to help a programmer to write programs.
  - ie., language design is not an exercise in and of itself.
  - if the language gets in the way, then it's not a good one.
- Other requirements, e.g., portability, stability, existing popularity, sponsorship by powerful organizations, should not be dominant factors.

User Requirements for a PL

1. A PL should give assistance in expressing what a program should accomplish and how it should execute.
2. A PL will encourage and assist in producing self-documenting code.
   - To find out what a program does, you (ideally) will be able to look in one place.
3. A PL will give assistance in finding errors.

Principles of Programming Language Design

- Simplicity is absolutely necessary.
  - Otherwise how will the designer know the consequences of their design decisions?
  - Pursue this to the extreme!
- Security.
- Fast generation of efficient code.
- Readability.
- Clear syntax to enable identification of syntax errors.
- Suitable structures for solving relevant problems.
- Proof rules for features of the language.
- Use patterns from other languages.
- Uniqueness.

Modelling Language Design

- How would you design a modelling language like UML?
- In theory, you would like to apply roughly the same process as one does with PLs.
- Can we learn by analogy?
  1. What are the user requirements for the graphical language?
  2. Define the syntax for the graphical language.
  3. Define the semantics for the graphical language via some analogy to structural induction over the syntax.
  4. Implement a compiler, tools, etc.
  5. Build a killer app.
- Ideally, all done using well understood rules for visual language design.

Requirements for UML

- A standard notation.
- A general purpose modelling language, initially for software, but now encompassing all of system modelling in all domains, using dialects.
- Enables communication.
- (Presumably) Supported by tools.
- Usable, user-friendly.
- Extensible.

Principles for Modelling Language Design

- Simplicity.
- Security.
- Drawable by tools and by hand.
- Readability.
- Clear syntax to enable identification of syntax errors.
- Suitable structures for solving relevant problems.
- Proof rules for features of the language.
- Use patterns from other languages.
- Uniqueness.
- Underlying simple mapping for semantics of model.

Graphical Syntax

- Captured using a metamodel.
- In general, you can capture the abstract syntax of a language and its concrete syntax.
  - Abstract syntax is analogous to abstract syntax trees for programming languages
  - It conveys the essence of the syntax, and aggregate certain details that are less interesting (eg., syntax for a boolean expression language).
  - An abstract syntax tree is usually heterogeneous.
  - Concrete syntax captures all the gory details.

Graphical Syntax (2)

- In general, this can be expressed in a suitably expressive existing graphical language.
  - Then the semantics of the existing graphical language influences (or even defines) the semantics of the new language.
- However, UML is the standard modelling language.
- So UML's graphical syntax is defined in UML.
  - Bootstrapping problem.
  - This is why MOF/CWM have been introduced – you need to assume some axioms somewhere!

Examples

- Abstract Syntax for UML 2.0
- Abstract Syntax for OCL 2.0
- See formal specifications available at www.omg.org.
- Work through parts of these specifications and explain some of the modelling concepts.
- Notice the recurring use of certain patterns, e.g., Composite.

Model vs Metamodel

- A model conforms to or complies with the metamodel.
- You can also think of a model as an instance of a metamodel.
- Thus, wrt the UML metamodel, a class is an instance of a ModelElement and a Classifier.
- It is usually helpful when drawing these things to think carefully about what level you're working in.

Model Conformance

L = (n:Notation, m:Metamodel)

Metamodel = Syntax ∪ Semantics

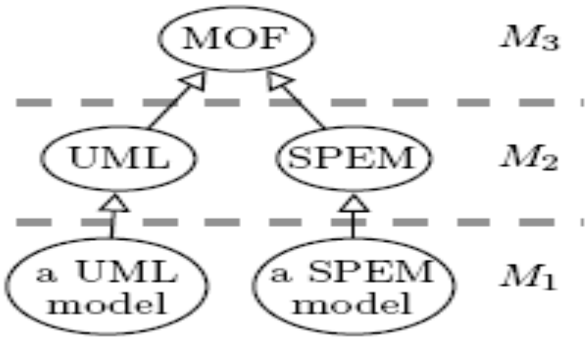
Semantics = Single_View ∪ Cross_Cutting

- For any model m in notation n,

  conforms(m,L) = ∀ c ∈ Metamodel • m **sat** c

- If `!conforms(m,L)` this means that there is an inconsistency in the model m.
- Can also handle uncertainty, i.e., omission of essential information in the model.
  - In general, uncertainty leads to instantiations of the **sat** relation that cannot be discharged.

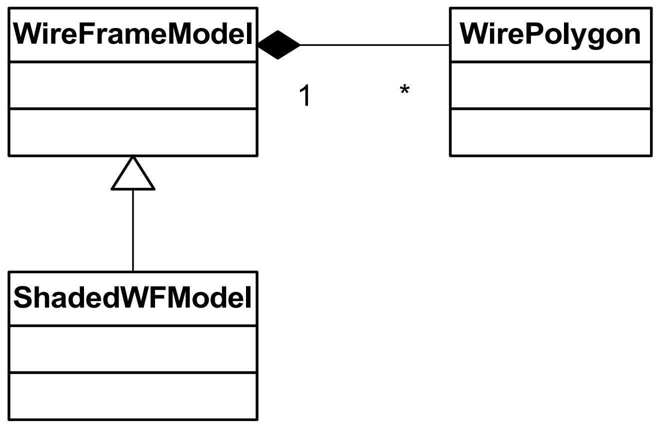
OMG's Pyramid Diagram



- The UML metamodel is at the M2 level (the meta-level).
- A specific UML model is at the M1 level (the model level).
- MOF is at the meta-meta-level: it is the language in which other languages are defined.
- There is also an M0 level which is an object configuration (ie., a snapshot).

Semantics

- Recall that UML's semantics is loosely defined (at best).
- Suppose you want to know what the following diagram means.



- Look at the metamodel.
- Each class is an instance of Classifier.

Tools for Building Metamodels

- UML.
  - For: Promotes understandability, deals well with reasonably large structures, CASE for drawing models.
  - Against: meta-circularity, semantic checking, pre-existing semantic fragmentation.
- A formal specification language.
  - For: avoids meta-circularity, existing proof system for semantic checking, possible to construct proofs of consistency/soundness.
  - Against: less understandable, expertise needed, transparency of the semantic mapping, backlash against FM.
  - Work has been done using Z, B, PVS, other FM.

Tools for Building Metamodels

- An executable language.
  - e.g., OCaml, Eiffel, JML.
  - For: understandable, works in the small and in the large, pre-existing tools for compilation, metamodels can be tested and simulated easily.
  - Against: risk of losing abstraction level, need transparency of mapping from modelling language, ongoing misconception that programs and models must be written in different languages.