Understanding Metamodelling

ER2003 15 October 2003

Presented by Brian Henderson-Sellers
With preparation input from
C. Atkinson, T. Kühne and C. Gonzalez-Perez
brian@it.uts.edu.au

Topics

1) What is metamodelling?
2) The meaning of "is-a"
3) UML extension mechanisms
4) Process modelling in the context of
   UML metamodelling hierarchy
5) Summary
1) What is metamodelling?

A metamodel is at a higher level of abstraction than a model. It is often called “a model of a model”. It provides the rules/grammar for the modelling language (ML) itself. The ML consists of instances of concepts in the metamodel.

“is-instance-of“ is key relationship i.e.
instance -> class
element -> set

Metamodels

- A metamodel describes the rules and constraints of metatypes and metarelationships
- Concrete metatypes are instantiated for use in regular modelling work.
  (Abstract metatypes do not appear as classes in a UML analysis/design model)
Example model which uses the rules of UML

Example model which does not adhere to UML
OMG’s 4-Layer Hierarchy

The 4-Layer Hierarchy

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta-metamodel</td>
<td>The infrastructure for a metamodeling architecture. Defines the language for specifying metamodels.</td>
<td>MetaClass, MetaAttribute, MetaOperation</td>
</tr>
<tr>
<td>metamodel</td>
<td>An instance of a meta-metamodel. Defines the language for specifying a model.</td>
<td>Class, Attribute, Operation, Component</td>
</tr>
<tr>
<td>model</td>
<td>An instance of a metamodel. Defines a language to describe an information domain.</td>
<td>StockShare, askPrice, sellLimitOrder, StockQuoteServer</td>
</tr>
<tr>
<td>user objects (user data)</td>
<td>An instance of a model. Defines a specific information domain.</td>
<td>&lt;Acme_SW_Share_98789&gt;, 654.36, sell_limit_order, &lt;Stock_Quote_Svr_32123&gt;</td>
</tr>
</tbody>
</table>
"is-instance-of"

- is not transitive
- has multiple dimensions e.g.
  - Domain or logical model (class Mortgage is instance of class LoanType)
  - Representational or physical model (class Mortgage is instance of class Class)

[Note. These two dimensions are confounded in the UML]

Is-instance-of is also ambiguous

```
\begin{array}{c}
\begin{array}{c}
L_2 \\
Breed \\
L_1 \\
Poodle \\
L_0 \\
fido \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
P_1 \\
\end{array}
\end{array}
\end{array}
```

- Physical
- Logical
Convention in UML/OO

is that the rectangles in a diagram are surrogates for their instances e.g.

Person \[\rightarrow\] Apple

means an instance of class Person eats an instance of class Apple NOT that class Person eats class Apple

BUT instance-of relationship violates this rule

Class

\[«\text{instanceOf}»\]

Poodle

refers to actual things on diagram, not to instances of it (as on previous slide) viz. class Poodle is an instance of metaclass Class
Less ambiguity if we used different notations

Classes as Objects

Poodle as class

Poodle as object
Poodle is example of a “clabject”

```
Class
  <instanceOf>>
  1
  
Fido
  <instanceOf>>

Breed

Poodle

```

Strict Metamodelling

```
MetaModel 1..* contains 1..*
  «instanceOf»

MetaModelElement
  «instanceOf»

Model 1..* contains 1..*
  «instanceOf»

Element

In an n-level modelling architecture, M₀, M₁, …, Mₙ₋₁, every element of an Mᵢ level model must be an instance-of exactly one element of an Mᵢ₊₁ level model, for all m ≤ n-1 and any relationship other than the instance-of relationship between two elements X and Y implies that level(X)=level(Y).
```

Ambiguous Classification

- Fido is classified by two types/classifiers
- Fido's instance-of relationship crosses two metalevel boundaries

Loose Metamodelling

"A Model is an instance of a Metamodel"

- Unstated criteria to distribute elements
  » elements are put where one finds a need to mention them
  » predefined elements go to $M_2$
- Metalevels become packages
  » should not be called metamodeling
- Exemplified by ambiguous classification problem
  » element is classified by two classifiers
  » instance-of relationship crosses two meta boundaries
2) The Meaning of "Is-A"

The English term is-a is highly ambiguous and often misused cf. Is-instance-of

Philosopher is a Person
Socrates is a Person
Socrates is a Philosopher
Dictionary is a Bag
Bag is a Dictionary
Set is a Collection
{1, 2} is a Bag
{1, 2} is a Collection
{1, 2} is a Set

Many Meanings of "Is-A" (1)

Classification leads to conclusions

Socrates is a man
Man is mortal
Socrates is mortal

Homo erectus stands upright
Homo sapiens is a homo erectus
Homo sapiens stands upright
Many Meanings of "Is-A" (2)

Imprecision leads to errors

Socrates is a Man  
Man is a Species  
Socrates is a Species

Corrected statements

Socrates is an instance-of Man  
Man is an instance-of Species  
Socrates is an instance-of a Species instance

Many Meanings of "Is-A" (3)

Generalization

kind-of

subclass-of

subtype-of

Instantiation

created-by

member-of

conforms-to
So, rewriting …

Philosopher is a kind-of Person
Socrates is a member-of Person
Socrates is best-described-by Philosopher
Dictionary is a kind-of Bag
Bag is a subclass-of Dictionary
Set is a subtype-of Collection
\{1, 2\} is created-by Bag
\{1, 2\} is a member-of Collection
\{1, 2\} conforms-to Set

Precise "Is-A" relationships

Counteracting imprecision

Socrates is a Man
Man is a Mammal
Man is a Species
Socrates is a Mammal
Socrates is not a Species
3) UML Extension Mechanisms

How to create your own version of the UML while remaining within the OMG standard

- Constraints [not discussed further here]
- Stereotypes
  » mainly for classification of classes,
- Tagged Values
  » (immutable) class level attributes

Stereotypes

- Stereotypes are applied to a BaseClass
- They
  » classify elements
  » may introduce tags
  » may define constraints
Application of stereotype

- After definition, this stereotype is then applied to model-level (M1) classes e.g.

```
«persistent»
Address
```

Stereotypes

- Stereotypes effectively introduce specialized user-defined meta-classes
  - Meta-Subclasses define subsets of modelling elements
  - Stereotypes can only influence modelling elements, not their instances
Uses of Stereotypes

Wrong

- Trying to express properties of class instances

Correct

- Expressing properties of classes

Tagged Values

- Describe class properties
- Are similar to (static) class variables in programming languages, but
  - are immutable
  - are not visible to instances
Consider first the Observer Pattern

Where to predefine Properties? (1)

An Example

Where to predefine Properties? (2)

- What is the best way to let Table act as a subject and PieChart as an observer?

- Two ways of obtaining predefined properties:
  - instantiation (metamodelling)
  - inheritance (generalization)
Where to predefine Properties? (3)

Metamodelling

M2
Subject
<< instanceof >>

M1
Table
- observers
- cells
- attach()
- detach()
- notify()
- getState()

Inheritance

M1
Subject
- observers
- attach()
- detach()
- notify()

Table
- cells
- getState()

Changing the Notation

Metamodelling

M2
- « Subject »
  Table
  observers
  cells
  attach()
  detach()
  notify()
  getState()

M1

Inheritance

M1
Subject
- observers
- attach()
- detach()
- notify()

Table
- cells
- getState()
Classification v Generalization

There are two (different!) forms of derivation. Both may be called "is-a".

<table>
<thead>
<tr>
<th>Class</th>
<th>Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : String</td>
<td>&gt; (Magnitude m) : Boolean</td>
</tr>
</tbody>
</table>

Renaming "is-a" to underline the difference

- classification
- generalization
In summary to this section

- Metamodelling uses “is-instance-of” (or “is-a”) which is not transitive
- The alternative of inheritance (UML Generalization relationship) is transitive

4) Process metamodelling cf. UML metamodelling hierarchy

- Background
- Metamodelling versus modelling
- OMG’s SPEM
Background: The 4-Layer Hierarchy

Problems

- Where to put process elements within this hierarchy?
- What relationships should be allowed between metalevels?

UML Artefact Description

$M_0$

$M_1$

$M_2$
Process Description

Activities have a duration

Moving to Software Domain

Shifting the Hierarchy Up

- Strictness violation removed
  - Bob’s artifacts are at the same level now
- Activity is now at M₃
  - conflict with MOF
- Activity attributes do not affect BobDesigns
  - duration not at Activity

Metamodelling vs Modelling (1)
Metamodelling vs Modelling (2)

Instantiation ($M_2/M_1$)  Inheritance ($M_1$)

- ObjectDiagram is an instance-of Diagram
- ObjectDiagram is a subclass of Diagram
Making a decision

- Do we want attributes in Diagram (e.g., count) to become slots in BobsObjectDiagram? ($\Rightarrow M_1$)
- Do we want instances of ObjectDiagram (e.g., BobsObjectDiagram) be members-of Diagram? ($\Rightarrow M_1$)
- Do we need ObjectDiagram to have individual state with regard to Diagram? ($\Rightarrow M_2$)

Strictness Restored (1)

```
<table>
<thead>
<tr>
<th>Activity</th>
<th>1</th>
<th>results in</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M2

<table>
<thead>
<tr>
<th>DesignActivity</th>
<th>1</th>
<th>results in</th>
<th>ClassDiagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>containsPatterns</td>
<td></td>
<td></td>
<td>classCount</td>
</tr>
</tbody>
</table>

M2

| «instanceOf» | | «instanceOf» | |

| «DesignActivity» | | «ClassDiagram» | BobsClassDiagram |
| BobDesigns | | | contains 1 |
| duration=50 | | classCount=2 | |
| UsePatterns=TRUE | | | |

M1
```
Strictness Restored (2)

After the Refactoring

- elements Activity and Diagram are now at level $M_2$.
- the instance-of relationships formerly existing between levels $M_3$ and $M_2$ have been replaced with inheritance links within level $M_2$.
- element Activity can now define an attribute duration.
- element DesignActivity no longer defines an attribute duration, but inherits it from Activity.

Modelling Spaces
Mixed Space Diagrams

Admitting non-strict Relationships

- Diagrams from different modelling spaces may be mixed
- The mixed diagram will contain non-strict relationships
- These only occur because modelling spaces are projected into the same diagram
- This is acceptable as long as the "bigger picture" is kept in mind

OMG’s SPEM

Recent standardization by the OMG proposed for use in process engineering
- heavily references RUP, OPEN, IBM process, Fujitsu SDEM21, DMR Macroscope
Conceptual model

Conceptually identical to OPEN’s metamodel

The SPEM solution

These three classes offer a partitioning feature for the class to which they are attached i.e. a substitute for subtyping
Newer solution: powertypes

- A powertype has instances which are subclasses of another type
- Seen examples already in Socrates (slide 20) and Date (slide 32)
- Powertypes do not fit into the strict metamodelling hierarchy

Another Example

TreeSpecies is a powertype

```
TreeSpecies
  name : String

SugarMaple
  location : String
  height()
  Name=SugarMaple

Tree
  location : String
  height()

myFriend : SugarMaple
  location = "myBackYard"
```
In terms of sets

Tree class

TreeSpecies class

TreeSpecies is a Powertype i.e. a set of all subsets of another set as defined by a given discriminator

An Example in Process Modelling

TaskKind

name : String

is classified into

Task

+assignedTeam : String

DefineOperation

name=DefineOperation
+assignedTeam : String

: DefineOperation
+assignedTeam=Liz,John

5) Summary (1)

- Metamodelling provides a higher abstraction level (to specify rules)
- Is-a can be ambiguous; as can is-instance-of
- Strict metamodelling requires is-instance-of only between levels – UML breaks this rule
- Property definition and transitivity of properties a problem

5) Summary (2)

- Metamodelling (using instantiation) versus modelling (using generalization)
- Mismatch between levels of UML and process metamodels like SPEM and OPF
- Possible way forward with powertypes