Meta-Modelling
Model vs. System

Model of a model — The correspondence continuum

*Meaning is rarely a simple mapping from a symbol to an object; instead it often involves a continuum of (semantic) correspondences from symbol to (symbol to)* object.*  [Barry Smith. The correspondence continuum. 1987]

• Example
  • A photo of a landscape is a model of the landscape.
  • A photocopy of the photo is model of a model of the landscape.
  • A digitalization of the photocopy is a model of the model of the model of the landscape.
  • etc.
**System**: a group of interacting, interrelated, or interdependent elements forming a complex whole.

**Model**: an abstract representation of a system created for a specific purpose.
A very popular model: Geographical maps

The System

- France in 1453
- The French cheese map
- Railroad map in western France
- Percentage of termite infestation in France
- Presidential elections in France

Modelling with UML, with semantics
Limited substitutability principle

- The purpose of a model is always to be able to answer some specific sets of questions in place of the system, exactly in the same way the system itself would have answered similar questions.

- A model represents certain specific aspects of a system and only these aspects, for a specific purpose.
Lewis Carroll and the 1:1 map

“That’s another thing we’ve learned from your Nation” said Mein Herr, “map-making. But we’ve carried it much further than you. What do you consider the largest map that would be really useful?”

“About six inches to the mile.”

“Only six inches!” exclaimed Mein Herr. “We very soon got to six yards to the mile. Then we tried a hundred yards to the mile. And then came the grandest idea of all! We actually made a map of the country, on the scale of a mile to the mile!”

”Have you used it much?” I enquired.

“*It has never been spread out*, yet” said Mein Herr: “the farmers objected: they said it would cover the whole country, and shut out the sunlight! So we now use the country itself, as its own map, and I assure you it does nearly as well.”

Lewis Carroll. Sylvie and Bruno concluded.
Lewis Carroll and the blank map

He had bought a large map representing the sea,
Without the least vestige of land:
And the crew were much pleased when they found it to be
A map they could all understand.
“What's the good of Mercator's North Poles and Equators,
Tropics, Zones, and Meridian Lines?”
So the Bellman would cry: and the crew would reply
“They are merely conventional signs!
Other maps are such shapes, with their islands and capes!
But we've got our brave Captain to thank:”
(So the crew would protest) “that he's bought us the best—
A perfect and absolute blank!”

Lewis Carroll. The Hunting Of The Snark — An Agony in Eight Fits.
Every map has a legend (implicit or explicit)

The legend is the metamodel

Same visual notation, different context, different meaning
Maps without legends are meaningless

First round of political election in France in 2002

Percentage of places infested by termites in France
The legend is a meta-model

- System
  - + ask()

- Model
  - + ask()

- Meta-model
  - + terminology
  - + assertions

repOf
conformsTo
The model of a model is not a meta-model
Meta-models act as filters

Furniture
  Table
  Chair

Person
  Attendant
  Presenter

sitsOn

A system

The metamodel

A model

Mary
Table 237
Chair 34
Paul
Victor
Emily

Modelling with UML, with semantics
Meta-models as simple ontologies

- Meta-models are precise abstraction filters.
- Each meta-model defines a domain-specific language.
- Each meta-model is used to specify which particular “aspect” of a system should be considered to constitute the model.

A metamodel defines a consensual agreement on how elements of a system should be selected to produce a given model.

An ontology is an explicit specification of a shared conceptualization.

The correspondence between a system and a model is precisely and computationally defined by a meta-model.
Multiple views and coordinated DSLs

- 1:1 map vs. blank map
- Limited substitutability principle
- A model has no meaning when separated from its meta-model.

Model

repOf

System

Plumber’s view

Architect’s view

Landlord’s view

Renter’s view

Mason’s view

Carpenter’s view

Electrician’s view

Interior Designer’s view

Tax Collector’s view
Modelling with UML, with semantics

Multiple views and aspects of a software system

System functions from the user view

Objects and basic relations between these objects

Physical components of an application

Class static structure and relations between these classes

Schemas of component installation on hardware devices

Representation of behavior in term of states

Representation of objects and their mutual links and potential interactions

Representation of objects and their temporal interactions

Representation of operation behavior in terms of actions
Meta-models

- A meta-model is just another model.
  - **Model of a set of models**
- Meta-models are specifications.
  - Models are valid if no false statements according to meta-model (e.g. well-formed)
  - Meta-models typically represent domain-specific models (real-time systems, safety critical systems, e-business)
- The domain of meta-modelling is language definition.
  - A meta-model is a model of some part of a language
  - Which part depends on how the meta-model is to be used
  - Parts: syntax, semantics, views/diagrams, ...

- **Meta-meta-model**
  - Model of meta-models
  - Reflexive meta-models expressed using itself
A "lattice" of meta-models

A collection of several hundreds of small meta-models (DSLs) with high abstraction power.

The system

A model

Modelling with UML, with semantics
The basic assumptions of MDE and MDSD

- Models as first class entities
- *Conformance* and *Representation* as kernel relations central to MDE
  - MDSD as a special case of MDE

![Diagram]

- MetaModel
- Model
- System
- Model conformsTo MetaModel
- Model isRepresentedBy System
Meta-modelling hierarchy or the meta-modelling stack

- **M0**: System
- **M1**: Model
- **M2**: Metamodel
- **M3**: Metametamodel

- **MOF**: The MOF (some kind of "representation ontology")
- **UML** and **IDL**: The UML metamodel and other MMs
- **Various usages of these models**
- **Some UML Models and other MMs**
Abstract Syntax Systems Compared

Technology #1 (formal grammars, attribute grammars, etc.)
- EBNF
- Pascal Language Grammar
- A specific Pascal Program

Technology #2 (MOF + OCL)
- MOF
- The UML meta-Model
- A Specific UML Model

Technology #3 (XML Meta-Language)
- A XML DTD or Schema
- A XML document

Technology #4 (Ontology engineering)
- Representation Ontologies
- KIF Theories
- +Description Logics
- +Conceptual Graphs
- +Xpath, XSLT
- +RDF, OIL, DAML
- +etc.

Modelling with UML, with semantics
Three-level hierarchy: Example — Petri-nets

Modelling with UML, with semantics
Modelling with UML, with semantics

Metametamodel: XML Schema for XML Schema

Metamodel: a Petri Net XML Schema

Model: an XML document

Classical representation

conformsTo

repOf

ψ alive

ψ dead
Modelling with UML, with semantics

**Metametamodel: EBNF grammar of EBNF**

```
productionRule := IDENT "::=" seq ";";
seq := alternative seq?;
alternative := rep ("|"alternative)?;
rep = atom ("?" | "*")?;
atom := terminal | "(" seq ")";
terminal := STRING IDENT;
```

**Metamodel: a Petri Net Grammar**

```
petrinet := "petrinet" "{"
  place* transition*
  arcPT* arcTP* "}";
place := "place" IDENT ";";
transition := "transition" IDENT IDENT ";";
arcPT := "arcPT" IDENT "-" IDENT;
arcTP := "arcTP" IDENT "-" IDENT;
```

**Model: a string**

```
petrinet {
  place P1;
  place P2;
  transition T1;
  arcPT P1 -> T1;
  arcTP T1 -> P2;
}
```
**Basic entities of MDE and MDSD**

*System*: a group of interacting, interrelated, or interdependent elements forming a complex whole.

*Technological Space*: a model management framework usually based on some algebraic structures (trees, graphs, hypergraphs, etc.).

*Model*: an abstract representation of a system created for a specific purpose.

---

**Modelling with UML, with semantics**
The notion of Technological Space (TS)

- A Technological Space corresponds to:
  - A **uniform representation system**
    - Syntactic trees
    - XML trees
    - Sowa graphs
    - UML graphs
    - MOF graphs
  - A **working context**
  - A **set of concepts**
  - A **set of methods**
  - A **shared knowledge and know-how**
  - etc.
- It is usually related to a given community with an established expertise, know-how and research problems.
- It has a set of associated tools and practices, etc.
  - Protégé, Rational Rose, …

Modelling with UML, with semantics
Modelling with UML, with semantics

Main Technological Spaces

- Grammar
- Program
  - Syntax
- Schema
- Document
  - XML
- DBMS
- Data
  - MDL
- Ontology
- Engineering
- Top Level O.

TS’s may be connected via bridges
Unified Modeling Language 2
History and Predecessors

• The UML is the “lingua franca” of software engineering.
• It subsumes, integrates and consolidates most predecessors.
• Through the network effect, UML has a much broader spread and much better support (tools, books, trainings etc.) than other notations.
• The transition from UML 1.x to UML 2.0 has
  • resolved a great number of issues;
  • introduced many new concepts and notations (often feebly defined);
  • overhauled and improved the internal structure completely.
• While UML 2 still has many problems, it is much better than what we ever had before.

current version (“the standard”) UML 2.4.1
formal/2011-08-06 of August ’11
Usage Scenarios

- UML has not been designed for specific, limited usages.

- There is currently no consensus on the rôle of the UML:
  - Some see UML only as tool for sketching class diagrams representing Java programs.
  - Some believe that UML is “the prototype of the next generation of programming languages”.

- UML is a really a system of languages (“notations”, “diagram types”) each of which may be used in a number of different situations.

- UML is applicable for a multitude of purposes and during all phases of the software lifecycle – to varying degrees.
Usage Scenarios
Diagram types in UML 2

UML is a coherent system of languages rather than a single language. Each language has its particular focus.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Class Diagram</th>
<th>static structure (generic/snapshot)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Composite Structure Diagram</td>
<td>logical system structure</td>
</tr>
<tr>
<td></td>
<td>Component Diagram</td>
<td>physical system structure</td>
</tr>
<tr>
<td></td>
<td>Deployment Diagram</td>
<td>computing infrastructure / deployment</td>
</tr>
<tr>
<td></td>
<td>Package Diagram</td>
<td>containment hierarchy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Use Case Diagram</th>
<th>abstract functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Activity Diagram</td>
<td>control flow and dataflow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Sequence Diagram</th>
<th>message exchange over time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Communication Diagram</td>
<td>structure of interacting elements</td>
</tr>
<tr>
<td></td>
<td>Timing Diagram</td>
<td>coordinated state change over time</td>
</tr>
<tr>
<td></td>
<td>Interaction Overview Diagram</td>
<td>flows of interactions</td>
</tr>
<tr>
<td></td>
<td>State Machine Diagram</td>
<td>event-triggered state change</td>
</tr>
</tbody>
</table>
Internal Structure: Overview

- The UML is structured using a metamodeling approach with four *layers*.
- The $M_2$-layer is called metamodel.
- The metamodel is again structured into *rings*, one of which is called superstructure, this is the place where concepts are defined ("the metamodel" proper).
- The Superstructure is structured into a tree of *packages* in turn.
### Internal Structure: Layers

<table>
<thead>
<tr>
<th>( M_3 )</th>
<th>Meta-Metamodel</th>
<th>EBNF</th>
<th>Meta Object Facility (MOF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_2 )</td>
<td>Metamodel</td>
<td>Java grammar</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Common Warehouse Metamodel (CWM)</td>
</tr>
<tr>
<td>( M_1 )</td>
<td>Model</td>
<td>a Java program</td>
<td>Albatros Air Autopilot</td>
</tr>
<tr>
<td>( M_0 )</td>
<td>System</td>
<td>an execution of a Java program</td>
<td>a runtime state in a deployment of Albatros Air Autopilot</td>
</tr>
</tbody>
</table>
Internal Structure: Layers

M₀  System
   |   : Movie
      +title="Fight Club"

M₁  Model
   |   Movie
      +title: String
       <snapshot>  m:Movie

M₂  Metamodel
   |   Attribute
       <InstanceOf>  <InstanceOf>  <InstanceOf>
       Class  InstanceSpecification

M₃  Metamodel
   |   Meta-Metamodel
       <InstanceOf>  <InstanceOf>  <InstanceOf>
       Class

Modelling with UML, with semantics
Internal Structure: Rings
Internal Structure: Packages
UML is not (only) object oriented

- A popular misconception about UML is that it is “object oriented” by heart – whatever that means.

- It is true that
  - UML defines concepts like class and generalization;
  - UML is defined using (mainly) a set of class models;
  - UML 2 rediscovers the idea of behaviour embodied in objects.

- However, UML 2
  - also encompasses many other concepts of non- or pre-OO origin (Activities, StateMachines, Interactions, CompositeStructure, …);
  - may be used in development projects completely independent of their implementation languages (Java, Cobol, Assembler, …);
  - is not tied to any language or language paradigm, neither by accident nor purpose.
Unified Modeling Language 2

Classes and packages
History and predecessors

• **Structured analysis and design**
  • Entity-Relationship (ER) diagrams (Chen 1976)

• **Semantic nets**
  • Conceptual structures in AI (Sowa 1984)

• **Object-oriented analysis and design**
  • Shlaer/Mellor (1988)
  • Coad/Yourdon (1990)
  • Wirfs-Brock/Wilkerson/Wiener (1990)
  • OMT (Rumbaugh 1991)
  • Booch (1991)
  • OOSE (Jacobson 1992)
Usage scenarios

- Classes and their relationships describe the vocabulary of a system.
  - **Analysis**: Ontology, taxonomy, data dictionary, …
  - **Design**: Static structure, patterns, …
  - **Implementation**: Code containers, database tables, …

- Classes may be used with different meaning in different software development phases.
  - meaning of generalizations varies with meaning of classes

<table>
<thead>
<tr>
<th></th>
<th>Analysis</th>
<th>Design</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>√</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Set of objects</td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Code</td>
<td>×</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>
Classes

- Classes describe a set of instances with common features (and semantics).
  - Classes induce types (representing a set of values).
  - Classes are namespaces (containing named elements).

- **Structural features** (are typed elements)
  - properties
    - commonly known as attributes
    - describe the structure or state of class instances
    - may have multiplicities (e.g. 1, 0..1, 0..*, *, 2..5)

- **Behavioral features** (have formal parameters)
  - operations
    - services which may be called
    - need not be backed by a method, but may be implemented otherwise

<table>
<thead>
<tr>
<th>Passenger</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : Name</td>
</tr>
<tr>
<td>creditCard [0..1]</td>
</tr>
<tr>
<td>milesCard [0..1]</td>
</tr>
<tr>
<td>status / currentMiles</td>
</tr>
<tr>
<td>creditMiles(b : Booking)</td>
</tr>
<tr>
<td>consumeMiles(b : Booking)</td>
</tr>
<tr>
<td>cancelMiles()</td>
</tr>
</tbody>
</table>
Associations

- **Associations** describe sets of tuples whose values refer to typed instances.
  - In particular, structural relationship between classes
  - Instances of associations are called links.

- **Association ends** are properties.
  - correspond to properties of the opposite class (default multiplicity is 0..1)
  - Association ends may be navigable.
    - in contrast to general properties

```plaintext
<table>
<thead>
<tr>
<th>Passenger</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : Name</td>
</tr>
<tr>
<td>creditMiles(b : Booking)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MilesAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
</tr>
<tr>
<td>flightMiles</td>
</tr>
<tr>
<td>statusMiles</td>
</tr>
</tbody>
</table>
```

- ternary association

```
<table>
<thead>
<tr>
<th>association name</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectingFlights</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>qualified end (fh per date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight</td>
</tr>
<tr>
<td>time : Time</td>
</tr>
<tr>
<td>miles : int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FlightHandling</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>navigation direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>reading</td>
</tr>
<tr>
<td>association end</td>
</tr>
<tr>
<td>navigable</td>
</tr>
<tr>
<td>not navigable</td>
</tr>
<tr>
<td>association end</td>
</tr>
</tbody>
</table>
```
Association classes

- **Association classes** combine classes with associations.
  - not only connect a set of classifiers but also define a set of features that belong to the relationship itself and not to any of the classifiers

  ![Diagram of association classes]

  - each instance of Booking has one passenger and one flight
  - each link of Booking is one instance of Booking
Data types and enumerations

• **Data types** are types whose instances are identified by their value.
  • Instances of classes have an identity.
  • may show structural and behavioural features

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName</td>
</tr>
<tr>
<td>middleInitial</td>
</tr>
<tr>
<td>surname</td>
</tr>
<tr>
<td>salutation</td>
</tr>
<tr>
<td>title</td>
</tr>
</tbody>
</table>

• **Enumerations** are special data types.
  • instances defined by enumeration literals
    • denoted by `Enumeration::EnumerationLiteral` or `#EnumerationLiteral`
  • may show structural and behavioural features

- compartments for attributes and operations suppressed
- enumeration literals
Inheritance (1)

- **Generalizations** relate specific classes to more general classes.
  - instances of specific class also instances of the general class
  - features of general class also implicitly specified for specific class

![Diagram of inheritance relationships]

- if decorated with `{ root }`: no superclass
- if decorated with `{ leaf }`: no subclass

- implies substitutability (in the sense of Liskov & Wing)
  - must be specified on specific class separately by `{ substitutable }`

- Generalizations also apply to associations.
  - as both are Classifiers
Inheritance (2)

- **Generalization sets** detail the relation between a general and more specific classifiers.
  - { complete } (opposite: { incomplete })
    - all instances of general classifier are instances of one of the specific classifiers in the generalization set
  - { disjoint } (opposite: { overlapping })
    - no instance of general classifier belongs to more than one specific classifier in the generalization set
  - default: { disjoint, incomplete }

  ![Diagram](diagram.png)

- several generalization sets may be applied to a classifier
  - useful for taxonomies
• **Constraints** restrict the semantics of model elements.
  - constraints may apply to one or more elements
  - no prescribed language
    - OCL is used in the UML 2 specification
    - also natural language may be used

- UML predefined constraint
  (owner is either a person or a company)

- User defined constraint
Packages (1)

- **Packages** group elements.
  - Packages provide a **namespace** for its grouped elements.
  - Elements in a package may be
    - public (+, visible from outside; default)
    - private (-, not visible from outside)
  - Access to public elements by qualified names
    - e.g., Flights::MilesAccount
• **Package imports** simplify qualified names.

<table>
<thead>
<tr>
<th>Package</th>
<th>Element</th>
<th>Visibility</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X</td>
<td>private</td>
<td>separate private element import (otherwise public overrides private)</td>
</tr>
<tr>
<td>A</td>
<td>Q</td>
<td>public</td>
<td>all remaining visible elements of B</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>public</td>
<td>public import</td>
</tr>
<tr>
<td>B</td>
<td>Q</td>
<td>public</td>
<td>default visibility</td>
</tr>
<tr>
<td>B</td>
<td>R</td>
<td>private</td>
<td>private import, renaming</td>
</tr>
</tbody>
</table>
• **Package mergings** combine concepts incrementally.
  • ... but use with care

• The receiving package defines the increment.

• The receiving package is simultaneously the resulting package.

• Merging is achieved by (lengthy) transformation rules (not defined for behaviour).

• Package merging is used extensively in the UML 2 specification.