Software Design, Modelling and Analysis in UML

Lecture 22: Meta-Modelling

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

Content

- Inheritance
- Abstract syntax
- Liskov Substitution Principle
- Well-typedness with inheritance
- Subset-semantics vs. uplink-semantics
- Meta-Modelling
- Idea
- Experiment: can we model classes?
- Revisit the UML 2.x standard (vs. experiment)
- Meta Object Facility (MOF)
- The principle illustrated (once again)
- And That's It!
- The map – in hindsight.
- Educational objectives – useful questions.
- Any open questions?

Inheritance: Concrete Syntax

Common graphical representations (of $\triangleright =$ \{ (C, D$_1$), (C, D$_2$) \}): 

$\begin{array}{c}
\text{C}\quad \text{D}_1 \\
\text{D}_1 \quad \text{D}_2 \\
\text{C}\quad \text{D}_2
\end{array}$

Mapping Concrete to Abstract Syntax by Example:

$\begin{array}{c}
\text{C}_0 \\
\text{C}_1 \\
\text{C}_2
\end{array}$

$\begin{array}{c}
\text{x} \\
\text{x} \\
\text{x}
\end{array}$

Note: we can have multiple inheritance.

Desired Semantics of Specialisation: Subtyping

There is a classical description of what one expects from sub-types, which is closely related to inheritance in object-oriented approaches: The principle of type substitutability: Liskov Substitution Principle (LSP) Liskov (1988); Liskov and Wing (1994).
There is a classical description of what one expects from sub-types, which is closely related to inheritance in object-oriented approaches: The principle of type substitutability:

Liskov Substitution Principle (LSP)

"If for each object \( o_S \) of type \( S \) there is an object \( o_T \) of type \( T \) such that for all programs \( P \) defined in terms of \( T \) the behavior of \( P \) is unchanged when \( o_S \) is substituted for \( o_T \) then \( S \) is a subtype of \( T \)."

In other words:

Fischer and Wehrheim (2000)

"An instance of the sub-type shall be usable whenever an instance of the supertype was expected, without a client being able to tell the difference."

Domain Inclusion vs. Uplink Semantics

Wanted: a formal representation of "if \( C \triangleright^* D \) then \( D \) is a \( C \)" , that is, (i) \( D \) has the same attributes and behavioural features as \( C \), and (ii) \( D \) objects (identities) can replace \( C \) objects.

Two approaches to semantics:

• Domain-inclusion Semantics (more theoretical)
• Uplink Semantics (more technical)

Inheritance and State-Machines: Example
a return type.

- any number of parameters,
- a boolean attribute

Behavioural features in addition have visibility:

- ?

using a modelling language (or: the set of all UML models) the set of all UML models is about modelling things. Among others has (MDA ideas of the OMG. The idea is somewhat simple, and if UML models are simple, then why not meta-modelling? Each of the latter two has meta-modelling:

Meta-Modelling: Idea:

- Meta-Modelling: Example
- Meta-Modelling: Why and What

In other words:

Liskov Substitution Principle (LSP)

is substituted for  

when  

is dispatched, i.e. 

in the domain of  cannot be distinguished from in the domain of

Therefore, a mechanism of type  enables a reaction to sub-typed.

Specify

This is the result of applying a dispatch, i.e. 

is expected in a context of type  because  is stable.

is expected in a context of type  because  is stable.

This is an instance of the StupidJoke!

This is an instance of the PoliteJoke!

This is an instance of the ClownJoke!

This is an instance of the TeacherJoke!

This is an instance of the StudentJoke!
• Meta-modelling has already been used for UML 1.x.
• For UML 2.0, the request for proposals (RFP) asked for a separation of concerns:
  Infrastructure and Superstructure.
• One reason: sharing with MOF (see later) and, e.g., CWM.

Core UML  MOF  CWM  Profiles

Figure 0-1  Overview of architecture

C: Class, Object  Action, Filmstrip  Package, Snapshot
C: Class, State, Transition, Flow, …

Superstructure (concrete syntax)
Superstructure (abstract syntax)

Infrastructure (with semantics)

Diagram Interchange
N: Node, Edge, …

Figure 7.5 - The top-level package structure of the UML 2.1.1 Superstructure

CommonBehaviors  UseCases  Classes  StateMachines  Interactions  CompositeStructures  Components  Deployments  AuxiliaryConstructs  Activities  Actions

Figure 7.12 - Classes diagram of the Kernel package

<<enumeration>>  AggregationKind

ValueSpecification

{redefines general}

+ /superClass

+subsettedProperty

{subsets classifier, subsets namespace, subsets featuringClassifier}

+ class

+ownedAttribute

{subsets attribute, subsets ownedMember, ordered}

{subsets redefinedElement}

+ redefinedProperty

+ /opposite

0..1

0..1

0..1

0..1 *

+ /endType

0..1

0..1

0..1

0..1

0..1

0..1

{subsets association, subsets namespace, subsets featuringClassifier}

+ owningAssociation

0..1 *

{subsets owner}

+ navigableOwnedEnd

*0..1

{subsets owner}

+owningProperty(subsets ownedElement)

+defaultValue

0..1 0..1

{readOnly, ordered}

+ /endType

0..1

0..1

0..1
So: things on meta level (comprising the notions of Benefits for Open Questions...).

A Classifier defines a type. Type conformance between generalizable Classifiers is defined so that a Classifier conforms to itself and to all of its ancestors in the generalization hierarchy. The specific semantics of how generalization affects each concrete subtype of Classifier varies. All instances of a general classifier also apply to instances of the specific classifier.

References the Element(s) being commented.

• /attribute: Property [*]

Classifier::feature

The specific type of classifier can be shown in guillemets above the name. Some specializations of Classifier have their own distinct notations.

• substitution : Substitution

Package PowerTypes

Figure 7.30 - Examples of attributes

...
Modelling vs. Meta-Modelling

- So, if we have a meta model $M_U$ of UML, then the set of UML models is the set of instances of $M_U$.
- A UML model $M$ can be represented as an object diagram (or system state) wrt. the meta-model $M_U$.
- Other view: An object diagram wrt. meta-model $M_U$ can (alternatively) be rendered as the UML model $M$.

Well-formedness as Constraints in the Meta-Model

- The set of well-formed UML models can be defined as the set of object diagrams satisfying all constraints of the meta-model $M_U$.
- Constraint example, 
  \[ \text{Generalization hierarchies must be directed and acyclical. A classifier cannot be both a transitively general and transitively specific classifier of the same classifier.} \]
  \[ 
  \text{not self} \] \( \to \) \( \text{includes} \) \( \text{self} \)
  \[ \text{allParents} \] \( \to \) \( \text{includes} \) \( \text{self} \)

- The other way round: Given a UML model $M$, unfold it into an object diagram $O_1$ wrt. $M_U$. If $O_1$ is a valid object diagram of $M_U$ (i.e. satisfies all invariants from $\text{Inv}(M_U)$), then $M$ is a well-formed UML model.
  That is, if we have an object diagram validity checker for of the meta-modelling language, then we have a well-formedness checker for UML models.

And That’s It!

The Map

- $UML\_\text{ModelInstances} = N^SWE\_\text{CD}, SM = (T, C, V, atr)$
- $SM = (\Sigma^D^S, A^S, \to SM)$
- $\varphi \in \text{OCL expr}$
- $CD, SD^S, SD^B = (Q^SD, q_0, A^S, \to SD, F^SD)$
- $\pi = (\sigma_0, \varepsilon_0)^\to u_0^\cdot \cdot \cdot^w \pi = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_i \in N$
Example: Object Diagrams for Documentation

• OCL Consistency and Satisfiability
  - Map class diagram to (extended) signature.

• OCL Semantics
  - Study UML syntax.

• OCL Syntax
  - Give a system state satisfying this constraint?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - Please explain this OCL constraint.

For what purposes are class diagrams useful?

Please formalise this constraint in OCL.

In what sense is OCL a three-valued logic? For what purpose?

Dr. Bernd Westphal

Prof. Dr. Andreas Podelski, Albert-Ludwigs-Universität Freiburg, Germany

Capabilities for these tasks/questions:

Educational Objectives:

What is a signature, an object, a system state, etc.?

What is the purpose of signature, object, etc. in the course?

What does it mean that an OCL expression is satisfiable?

How do Basic Object System Signatures relate to UML class diagrams?

Please un-abbreviate all abbreviations in this OCL expression.

How are \( \Sigma \)

and Structure \( S \)

related?

What is an object diagram? What are object diagrams good for?

Give a system state satisfying this constraint?

Please explain this OCL constraint.

Please formalise this constraint in OCL.

What is the purpose of signature, object, etc. in the course?

Please un-abbreviate all abbreviations in this OCL expression.

Please explain this OCL constraint.

Please formalise this constraint in OCL.

What does it mean that an OCL expression is satisfiable?

How do Basic Object System Signatures relate to UML class diagrams?

Please un-abbreviate all abbreviations in this OCL expression.

Please explain this OCL constraint.
Step, Run-to-Completion Step

Ether Associations: the rest.

Associations and OCL: semantics.

Lecture 10: State Machines Overview

• For completeness: Modelling Guidelines for Class Diagrams
• Example: Object Diagrams for Documentation

Stereotypes – for documentation.

Map class diagram to (extended) signature.

OCL Semantics (over system states)

Could you please model the following behaviour.

Could you please map this signature to a class diagram?

What is: Signal, Event, Ether, Transformer, Step, RTC.

OCL: consistency, satisfiability

Capabilities for following tasks/questions.

Could you please map this class diagram to a signature?

When is an object diagram an object diagram (wrt. what)?

When is a set of OCL constraints said to be consistent?

When is an object diagram called partial? What are partial ones good for?

What's a role name? What's it good for?

What is a signature, an object, a system state, etc.?

What is visibility good for?

What is the purpose of signature, object, etc. in the course?

What is the semantics of 'abstract'?

What is visibility as an extension of well-typedness.

What is multiplicity? How did we treat them semantically?

What is visibility good for?

What is a role name? What's it good for?

What is 'reading direction', 'navigability', 'ownership', . . . ?

What is a signature, an object, a system state, etc.?

What's a role name? What's it good for?
Entry / exit / do actions, internal transitions: Hierarchical State Machines II

Lecture 14

What is the idea of deferred events?

What about junction, choice, terminate, etc.?

Transformers

UML Core State Machines

What are roles of OCL contraints in behavioural models?

What is divergence in the context of UML models?

What is: Signal, Event, Ether, Transformer, Step, RTC.

Stereotypes.

Can you please model the following behaviour.

Stereotypes – for documentation.

What's the difference between "aggregation" and "composition"?

Capabilities for following tasks/questions.

What is: Signal, Event, Ether, Transformer, Step, RTC.

OCL Semantics (over system states)

Capabilities for following tasks/questions.

What is "reading direction", "navigability", "ownership", . . . ?

What is visibility good for?

Which annotations of an association arrow are semantically related?

How are system states and object diagrams related?

How are Basic Object System Signatures relate to UML class diagrams?

Give a system state satisfying this constraint?

Please un-abbreviate all abbreviations in this OCL expression.

Last Lecture:

Please explain this OCL constraint.

This Lecture:

Example: Object Diagrams for Documentation

For completeness: Modelling Guidelines for Class Diagrams

Completed class diagrams. . . except for associations.

Last Lectures:

This Lecture:

Study UML syntax.

Study effect on OCL.

This Lecture:

Study UML syntax.

This Lecture:

Study UML syntax.

Last Lecture:

Please explain this class diagram with associations.

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:

Last Lecture:

This Lecture:
What is the subset/uplink semantics of inheritance?

Transformers

What is: Signal, Event, Ether, Transformer, Step, RTC.

Educational Objectives:

Capabilities for following tasks/questions.

In what sense is OCL a three-valued logic? For what purpose?

Could you please map this signature to a class diagram?

What is visibility good for?

Does this OCL constraint hold in this system state?

What is the abstract syntax of this LSC?

Construct the TBA for this LSC.

What is the abstract syntax of this LSC?

What does this State Machine mean? What happens if I inject this event?

What is the purpose of signature, object, etc. in the course?

Is this UML model consistent with that OCL constraint?

How is the semantics of LSCs constructed?

How do Basic Object System Signatures relate to UML class diagrams?

Please formalise this constraint in OCL.

Give a system state satisfying this constraint?

Purposes of Behavioural Models

What is "reading direction", "navigability", "ownership", . . . ?

Could you please map this signature to a class diagram?

Compute the value of a given OCL constraint in a system state with links.

What is “reading direction”, “navigability”, “ownership”, . . . ?

Related?

Partial vs. complete; for analysis; for documentation. . .

OCL Propagation & Invariants

OCL Consistency and Satisfiability

Rhapsody code generation

Rules (i) to (v) for hierarchical state machines

Entry / exit / do actions, internal transitions

Passive reactive objects

Forbidden scenarios

A Magic Wand

System configuration cont'd

System configuration

What are constructive and reflective descriptions of behaviour?

System configuration

What is software design, what is software architecture?

OCL in BES

What is an OCL constraint? How do I use it?

OCL Semantics (over system states)

OCL Semantics

OCL Semantics

Lecture 12: OCL Semantics

Lecture 13: OCL Semantics

Lecture 14: OCL Semantics

Last Lecture:

Lecture 15: OCL Semantics

Lecture 16: OCL Semantics

This Lecture:

This Lecture:

This Lecture:

Last Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture:

This Lecture: