Software Design, Modelling and Analysis in UML

Lecture 22: Meta-Modelling

2017-02-07

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
• 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
A **signature with inheritance** is a tuple

\[ \mathcal{S} = (\mathcal{T}, \mathcal{C}, V, atr, \mathcal{E}, F, mth, \triangleleft) \]

where

- \((\mathcal{T},\mathcal{C},V,atr,\mathcal{E})\) is a signature with signals and behavioural features (\(F/mth\) are methods, analogous to \(V/atr\) attributes), and

- \(\triangleleft \subseteq (\mathcal{C} \times \mathcal{C}) \cup (\mathcal{E} \times \mathcal{E})\)

  is an **acyclic generalisation** relation, i.e. \(C \triangleleft^+ C\) for no \(C \in \mathcal{C}\).

In the following (for simplicity), we assume that all attribute (method) names are of the form \(C::v\) and \(C::f\) for some \(C \in \mathcal{C} \cup \mathcal{E}\) ("fully qualified names").

Read \(C \triangleleft D\) as...

- \(D\) **inherits** from \(C\),
- \(C\) is a **generalisation** of \(D\),
- \(D\) is a **specialisation** of \(C\),
- \(C\) is a **super-class** of \(D\),
- \(D\) is a **sub-class** of \(C\),
- ...
**Inheritance: Concrete Syntax**

**Common graphical representations** (of $\leq \{(C, D_1), (C, D_2)\}$):

[Diagram showing three graphical representations of inheritance relationships]

**Mapping** Concrete to Abstract Syntax by Example:

[Diagram showing mapping from concrete to abstract syntax examples]

**Note:** we can have **multiple inheritance**.
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:
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).  
“\( \text{If for each object } o_S \text{ of type } S \)  
\( \text{there is an object } o_T \text{ of type } T \)  
\( \text{such that for all programs } P \text{ defined in terms of } T \)  
\( \text{the behavior of } P \text{ is unchanged when } o_S \text{ is substituted for } o_T \)  
then \( S \) is a **subtype** of \( T \).”

In other words: Fischer and Wehrheim (2000)  
“\( \text{An instance of the sub-type shall be usable} \)  
\( \text{whenever an instance of the supertype was expected,} \)  
\( \text{without a client being able to tell the difference.} \)”
In FrontEnd's state machine:

\[
\text{\textit{itsSession} := new Session}
\]

\[
\text{\textit{itsSession}.dump()}
\]

\[
\text{\textit{itsSession} := new VIPSession}
\]

\[
T_1 = \text{Session}_{0,1} \quad T_2 = \text{VIPSession}_{0,1}
\]

OK, if \( T_1 <^* T_2 \)
Domain Inclusion vs. Uplink Semantics
**Wanted**: a formal representation of “if \( C \prec \ast D \) 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
- **Uplink** Semantics

\[
\text{for } u \in D(C^n) : \quad \text{dom } \sigma(u) = \bigcup \text{attr } C_0 \quad \text{(more theoretical)}
\]

\[
\begin{align*}
1 & : C \\
1 & : x = 0 \\
2 & : D \\
2 & : y = 1 \\
3 & : x = 2 
\end{align*}
\]
Inheritance and State-Machines: Example

\[
\begin{align*}
\text{\textbf{SM}_A:} & \quad s_1 \xrightarrow{\text{/} n \! \! F} s_2 \\
\text{\textbf{SM}_D:} & \quad s_1 \xrightarrow{E/} s_2 \\

\begin{array}{c}
\text{\textbf{A} } \\
\xrightarrow{n} \\
\begin{array}{c}
\text{\textbf{C}} \\
\uparrow \\
\text{\textbf{D}} \\
\end{array} \\
\end{array} \\
\begin{array}{c}
\text{\textbf{E}} \\
\uparrow \\
\text{\textbf{F}} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{l}
\text{\textbf{u}_1 : A} \\
st = s_1 \\
stab = 0
\end{array} & \quad \begin{array}{l}
\text{\textbf{u}_1 : A} \\
st = s_2 \\
stab = 1
\end{array} \\
\xrightarrow{n} & \quad \begin{array}{l}
\text{\textbf{u}_2 : D} \\
st = s_1 \\
stab = 1
\end{array} & \quad \begin{array}{l}
\text{\textbf{u}_2 : D} \\
st = s_1 \\
stab = 1
\end{array} \\
\xrightarrow{(\emptyset, \{(u_3 : F, u_2)\})} & \quad \begin{array}{l}
\text{\textbf{u}_3 : F}
\end{array} \\
\xrightarrow{\text{D}(+)} & \quad \begin{array}{l}
\text{\textbf{D}(E)}
\end{array} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\varepsilon & = \varepsilon \\
\varepsilon & = (u_2, u_3 : F)
\end{align*}
\]
Inheritance and State-Machines: Example

**SM\(_A\):**

- \( s_1 \xrightarrow{/n \cdot F} s_2 \)
- \( s_1 \xrightarrow{E/} s_3 \)

**SM\(_D\):**

- \( s_1 \xrightarrow{E/} s_2 \)

**u\(_1\):**

- \( A \)
- \( st = s_1 \)
- \( stable = 0 \)

\[ \xrightarrow{n} \]

**u\(_2\):**

- \( D \)
- \( st = s_1 \)
- \( stable = 1 \)

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

\[ (\emptyset, \{(u_3:F, u_2)\}) \xrightarrow{} u_1 \]

**ε = ε**
(ii) Dispatch

\[(\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})}{_u} (\sigma', \varepsilon')\]

if

- \(u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \land \exists u_E \in \mathcal{D}(E) : u_E \in \text{ready}(\varepsilon, u)\)
- \(u\) is stable and in state machine state \(s\), i.e. \(\sigma(u)(\text{stable}) = 1\) and \(\sigma(u)(\text{st}) = s\),
- a transition is enabled, i.e.

\[\exists (s, F, \text{expr}, \text{act}, s') \in (\mathcal{SM}_C) : F = E \land I[\text{expr}](\bar{\sigma}, u) = 1\]

where \(\bar{\sigma} = \sigma[u.\text{params}_E \mapsto u_E]\).

and

- \((\sigma', \varepsilon')\) results from applying \(t_{\text{act}}\) to \((\sigma, \varepsilon)\) and removing \(u_E\) from the ether, i.e.

\[t_{\text{act}}[u](\bar{\sigma}, \varepsilon \ominus u_E), \quad \sigma' = (\sigma''[u.\text{st} \mapsto s', u.\text{stable} \mapsto b, u.\text{params}_E \mapsto \emptyset])|\mathcal{D}(C)\{u_E\}\]

where \(b\) depends (see (i))

- Consumption of \(u_E\) and the side effects of the action are observed, i.e.

\[\text{cons} = \{u_E\}, \quad \text{Snd} = \text{Obs}_{t_{\text{act}}}[u](\bar{\sigma}, \varepsilon \ominus u_E).\]
Recall: 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).

“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.”
Subtyping: Example

\[ s.\text{att} > 0 \] / GoodAns

\[ s.\text{att} < 0 \] / WrongAns

\[ s.\text{att} = 0 \] / Silence

\[ s.\text{att} = 1 \] / StupidJoke

\[ s.\text{att} = 2 \] / Task
Meta-Modelling: Idea
Meta-Modelling: Why and What

- **Meta-Modelling** is one major prerequisite for understanding
  - the standard documents OMG (2011a,b), and
  - the MDA ideas of the OMG.

- The idea is somewhat **simple**:
  - if a *modelling language* is about modelling *things*,
  - and if UML models are *things*,
  - then why not *describe* (or: *model*) the set of all UML models *using a modelling language*?
Meta-Modelling: Example

For example, let’s consider a class.

- A **class** has (**among others**)
  - a **name**,
  - any number of **attributes**,  
  - any number of **behavioural features**.

Each of the latter two has
- a **name** and
- a **visibility**.

Behavioural features in addition have
- a boolean attribute **isQuery**,  
- any number of parameters,
- a return type.

Can we model this (in UML, for a start)?
The UML 2.x Standard Revisited
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.
UML Superstructure Packages *(OMG, 2007a, 15)*

Figure 7.5 - The top-level package structure of the UML 2.1.1 Superstructure
Figure 7.12 - Classes diagram of the Kernel package
Figure 7.11 - Operations diagram of the Kernel package
Figure 7.10 - Features diagram of the Kernel package
Claim: Extract from UML 2.0 Standard

Diagram showing the relationships between UML elements:
- Comment
- Element
- NamedElement
  - name
  - visibility
- Type
- TypedElement
  - type
  - 0..1
  - *
- Class
- Classifier
- StructFeature
- BehavFeature
- Property
- Operation
  - 0..1
  - *
- Parameter
- Namespace
- RedefElement
  - redefElem
  - *
Figure 7.9 - Classifiers diagram of the Kernel package
Figure 7.4 - Namespaces diagram of the Kernel package
Figure 7.3 - Root diagram of the Kernel package
Table of Contents

1. Scope ................................................................. 1
2. Conformance ......................................................... 1
   2.1 Language Units .................................................. 2
   2.2 Compliance Levels ............................................. 2
   2.3 Meaning and Types of Compliance ...................... 6
   2.4 Compliance Level Contents ................................. 8
3. Normative References ............................................ 10
4. Terms and Definitions ........................................... 10
5. Symbols .............................................................. 10
6. Additional Information .......................................... 10
   6.1 Changes to Adopted OMG Specifications ............... 10
   6.2 Architectural Alignment and MDA Support ............. 10
   6.3 On the Run-Time Semantics of UML ...................... 11
      6.3.1 The Basic Premises ..................................... 11
      6.3.2 The Semantics Architecture ........................... 11
      6.3.3 The Basic Causality Model ............................. 12
      6.3.4 Semantics Descriptions in the Specification ....... 13
   6.4 The UML Metamodel ......................................... 13
      6.4.1 Models and What They Model ........................ 13
      6.4.2 Semantic Levels and Naming .......................... 14
   6.5 How to Read this Specification ............................ 15
      6.5.1 Specification format ...................................... 15
      6.5.2 Diagram format ........................................... 18
   6.6 Acknowledgements ............................................ 19

Part I - Structure .................................................. 21

7. Classes ............................................................. 23
Table of Contents

1. Scope .............................................. 1

2. Conformance ........................................ 2

2.1 Language Units ..................................... 2

2.2 Compliance Levels .................................. 3

2.3 Meaning and Types of Compliance .......... 6

3. Normative References ............................ 10

4. Terms and Definitions ............................ 11

5. Symbols ............................................. 11

6. Additional Information ........................... 15

6.1 Changes to Adopted Norms .................. 15

6.2 Architectural Alignment ....................... 17

6.3 On the Run-Time Semantic Levels .......... 18

6.3.1 The Basic Premise .......................... 18

6.3.2 The Semantics Ascriptions ............... 19

6.3.3 The Basic Causes ........................... 19

6.3.4 Semantics Descriptions ................. 20

6.4 The UML Metamodel ......................... 21

6.4.1 Models and What They Model .......... 22

6.4.2 Semantic Levels and Naming .......... 23

6.5 How to Read this Specification ............. 25

6.5.1 Specification format ....................... 25

6.5.2 Diagram format ............................. 26

6.6 Acknowledgements ............................. 28

Part I - Structure ................................. 29

7. Classes ............................................. 30

7.1 Overview ........................................ 23

7.2 Abstract Syntax ................................. 24

7.3 Class Descriptions .............................. 38

7.3.1 Abstraction (from Dependencies) ...... 38

7.3.2 AggregationKind (from Kernel) ...... 38

7.3.3 Association (from Kernel) ............... 39

7.3.4 AssociationClass (from AssociationClasses) .... 47

7.3.5 BehavioralFeature (from Kernel) ..... 48

7.3.6 BehavioredClassifier (from Interfaces) ... 49

7.3.7 Class (from Kernel) ....................... 49

7.3.8 Classifier (from Kernel, Dependencies, PowerTypes) ... 52

7.3.9 Comment (from Kernel) .................. 57

7.3.10 Constraint (from Kernel) ............... 58

7.3.11 DataType (from Kernel) ............... 60

7.3.12 Dependency (from Dependencies) .... 62

7.3.13 DirectedRelationship (from Kernel) .. 63

7.3.14 Element (from Kernel) ................. 64

7.3.15 ElementImport (from Kernel) ........ 65

7.3.16 Enumeration (from Kernel) ............ 67

7.3.17 EnumerationLiteral (from Kernel) ... 68

7.3.18 Expression (from Kernel) ............. 69

7.3.19 Feature (from Kernel) ................... 70

7.3.20 Generalization (from Kernel, PowerTypes) ...... 71

7.3.21 GeneralizationSet (from PowerTypes) .... 75

7.3.22 InstanceSpecification (from Kernel) ... 82

7.3.23 InstanceValue (from Kernel) .......... 85

7.3.24 Interface (from Interfaces) ............ 86

7.3.25 InterfaceRealization (from Interfaces) ... 89

7.3.26 LiteralBoolean (from Kernel) ........ 89

7.3.27 LiteralInteger (from Kernel) ........ 90

7.3.28 LiteralNull (from Kernel) ............. 91

7.3.29 LiteralSpecification (from Kernel) .... 92

7.3.30 LiteralString (from Kernel) .......... 92

7.3.31 LiteralUnlimitedNatural (from Kernel) ... 93

7.3.32 MultiplicityElement (from Kernel) ... 94

7.3.33 NamedElement (from Kernel, Dependencies) .... 97

7.3.34 Namespace (from Kernel) ........... 99

7.3.35 OpaqueExpression (from Kernel) ....... 101

7.3.36 Operation (from Kernel, Interfaces) .. 103

7.3.37 Package (from Kernel) ................. 107

7.3.38 PackageableElement (from Kernel) .... 109

7.3.39 PackageImport (from Kernel) ........ 110

7.3.40 PackageMerge (from Kernel) .......... 111

7.3.41 Parameter (from Kernel, AssociationClasses) ... 120

7.3.42 ParameterDirectionKind (from Kernel) ... 122

7.3.43 PrimitiveType (from Kernel) .......... 122

7.3.44 Property (from Kernel, AssociationClasses) ... 123

7.3.45 Realization (from Dependencies) .... 129

7.3.46 RedefinableElement (from Kernel) .... 130

UML Superstructure Specification, v2.1.2
Reading the Standard

Table of Contents

1. Scope ................................................................. 2
2. Conformance ......................................................... 5
   2.1 Language Units ............................................. 5
   2.2 Compliance Levels ........................................ 12
   2.3 Meaning and Types ........................................ 12
   2.4 Compliance Level Contents ............................ 13
3. Normative References ............................................. 13
4. Terms and Definitions ............................................. 15
5. Symbols ............................................................... 16
6. Additional Information ............................................ 18
   6.1 Changes to Adopted Subclauses .................... 18
   6.2 Architectural Alignments ............................... 20
   6.3 On the Run-Time Semantics ......................... 21
      6.3.1 The Basic Premises ............................... 21
      6.3.2 The Semantics Architecture .................. 22
      6.3.3 The Basic Causality .............................. 23
      6.3.4 Semantics Descriptions ....................... 24
   6.4 The UML Metamodel ........................................ 25
      6.4.1 Models and What They Mean .................. 25
      6.4.2 Semantic Levels .................................... 27
      6.4.3 The Unified Modeling Language ............. 28
   6.5 How to Read this Specification ....................... 28
      6.5.1 Specification format .............................. 28
      6.5.2 Diagram format .................................... 30
   6.6 Acknowledgements .......................................... 31
7. Classes ............................................................... 32
8. Components ........................................................ 33
9. Composite Structures ............................................ 37
10. Deployments ......................................................... 38

8. Components ........................................................ 143
8.1 Overview ........................................................ 143
8.2 Abstract syntax ................................................. 144
8.3 Class Descriptions ............................................. 151
   8.3.2 Connector (from BasicComponents) ............. 152
   8.3.3 ConnectorKind (from BasicComponents) ........ 157
   8.3.4 ComponentRealization (from BasicComponents) 157
8.4 Diagrams ......................................................... 159

9. Composite Structures ............................................ 161
9.1 Overview ........................................................ 161
9.2 Abstract syntax ................................................. 161
9.3 Class Descriptions ............................................. 166
   9.3.3 Collaboration (from Collaborations) ............. 167
   9.3.4 CollaborationUse (from Collaborations) ........ 171
   9.3.5 ConnectableElement (from InternalStructures) 174
   9.3.6 Connector (from InternalStructures) ............ 174
   9.3.7 ConnectorEnd (from InternalStructures, Ports) 176
   9.3.8 EncapsulatedClassifier (from Ports) ............. 178
   9.3.9 InvocationAction (from InvocationActions) .... 178
   9.3.10 Parameter (from Collaborations) ............... 179
   9.3.11 Port (from Ports) ................................... 179
   9.3.12 Property (from InternalStructures) .......... 179
   9.3.13 StructuredClassifier (from InternalStructures) 186
   9.3.14 Trigger (from InvocationActions) ............... 190
   9.3.15 Variable (from StructuredActivities) .......... 191
9.4 Diagrams ......................................................... 191

10. Deployments ......................................................... 194
7.3.8 Classifier (from Kernel, Dependencies, PowerTypes)

A classifier is a classification of instances, it describes a set of instances that have features in common.

Generalizations

- “Namespace (from Kernel)” on page 99
- “RedefinableElement (from Kernel)” on page 130
- “Type (from Kernel)” on page 135

Description

A classifier is a namespace whose members can include features. Classifier is an abstract metaclass.

A classifier is a type and can own generalizations, thereby making it possible to define generalization relationships to other classifiers. A classifier can specify a generalization hierarchy by referencing its general classifiers.

A classifier is a redefinable element, meaning that it is possible to redefine nested classifiers.

Attributes

- isAbstract: Boolean
  
  If true, the Classifier does not provide a complete declaration and can typically not be instantiated. An abstract classifier is intended to be used by other classifiers (e.g., as the target of general metarelationships or generalization relationships). Default value is false.

Associations

- /attribute: Property [*]
  
  Refers to all of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets Classifier::feature and is a derived union.

- /feature : Feature [*]
  
  Specifies each feature defined in the classifier. Subsets Namespace::member. This is a derived union.

- /general : Classifier[*]
  
  Specifies the general Classifiers for this Classifier. This is derived.
7.3.8 Classifier (from Kernel, Dependencies, PowerTypes)

A classifier is a classification of instances, it describes a set of instances that have features in common.

Generalizations

- “Namespace (from Kernel)” on page 99
- “RedefinableElement (from Kernel)” on page 130
- “Type (from Kernel)” on page 135

Description

A classifier is a namespace whose members can include features. Classifier is an abstract metaclass. A classifier is a type and can own generalizations, thereby making it possible to define generalization relationships to other classifiers. A classifier can specify a generalization hierarchy by referencing its general classifiers.

Attributes

- isAbstract: Boolean
  - If true, the Classifier does not provide a complete declaration and can typically not be instantiated. An abstract classifier is intended to be used by other classifiers (e.g., as the target of general metarelationships or generalization relationships). Default value is false.

Associations

- /attribute: Property [*]
  - Refers to all of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets Namespace::member and is a derived union.
- /feature: Feature [*]
  - Specifies each feature defined in the classifier. Subsets Namespace::member. This is a derived union.
- /general: Classifier [*]
  - Specifies the general Classifiers for this Classifier. This is derived.
- /inheritedMember: NamedElement [*]
  - Specifies all elements inherited by this classifier from the general classifiers. Subsets Namespace::member. This is derived.
- /redefinedClassifier: Classifier [*]
  - References the Classifiers that are redefined by this Classifier. Subsets RedefinableElement::redefinedElement

Package Dependencies

- substitution: Substitution
  - References the substitutions that are owned by this Classifier. Subsets Element::ownedElement and NamedElement::clientDependency.

Package PowerTypes

- powertypeExtent: GeneralizationSet
  - Designates the GeneralizationSet of which the associated Classifier is a power type.

Constraints

1. The general classifiers are the classifiers referenced by the generalization relationships.

\[
\text{general} = \text{self.parents()}
\]

2. Generalization hierarchies must be directed and acyclical. A classifier cannot be both a transitively general and transitively specific classifier of the same classifier.

\[
\text{not self.allParents() -> includes(self)}
\]

3. A classifier may only specialize classifiers of a valid type.

\[
\text{self.parents() -> forAll(c | self.maySpecializeType(c))}
\]

4. The inheritedMember association is derived by inheriting the inheritable members of the parents.

\[
\text{self.inheritedMember -> includesAll(self.inheritablesMembers(self)])}
\]

Package PowerTypes

5. The Classifier that maps to a GeneralizationSet may neither be a specific nor a general Classifier in any of the Generalization relationships defined for that GeneralizationSet. In other words, a power type may not be an instance of itself nor may its instances also be its subclasses.

Additional Operations

1. The query allFeatures() gives all of the features in the namespace of the classifier. In general, through mechanisms such as inheritance, this will be a larger set than feature.

\[
\text{Classifier::allFeatures(): Set(Feature)}\]

\[
\text{allFeatures = member -> select(oclIsKindOf(Feature))}
\]

2. The query parents() gives all of the immediate ancestors of a generalized Classifier.

\[
\text{Classifier::parents(): Set(Classifier)}\]

\[
\text{parents = generalization.generation}
\]
The query `allParents()` gives all of the direct and indirect ancestors of a generalized Classifier.

Classifier::allParents(): Set(Classifier);

\[allParents = self.parents() \cup (self.parents() \mapsto allParents())\]

This specifies the Generalization relationships for this Classifier. These Generalizations navigate to more general classifiers.

The query `inheritableMembers()` gives all of the members of a classifier that may be inherited in one of its descendants, subject to whatever visibility restrictions apply.

Classifier::inheritableMembers(c: Classifier): Set(NamedElement);

\[\text{pre: } c\in allParents() \cup \text{includes}(self)\]

\[\text{inheritableMembers = member \mapsto select(m | c\text{ hasVisibilityOf}(m))}\]

The query `inherits()` defines how to inherit a set of elements. Here the operation is defined to inherit them all. It is intended to be redefined in circumstances where inheritance is affected by redefinition.

Classifier::inherits(inhs: Set(NamedElement)): Set(NamedElement);

\[\text{pre: } self\text{ allParents()} \cup \text{includes}(inhs)\]

\[\text{self.inheritedMember} \supseteq \text{all(self.inherit(self.parents() \mapsto allMembers()))}\]

The query `conformsTo()` gives true for a classifier that defines a type that conforms to another. This is used, for example, in the specification of signature conformance for operations.

Classifier::conformsTo(other: Classifier): Boolean;

\[\text{pre: } self\text{ allParents()} \cup \text{includes}(other)\]

The query `isAbstract()` determines whether a classifier is abstract. By default an abstract classifier cannot be instantiated.

Classifier::isAbstract(): Boolean;

\[\text{true}\]

A classifier can specify a generalization hierarchy by referencing its general classifiers.

Generalizations defined for that GeneralizationSet. In other words, a power type may not be an instance of any of the general classifiers that map to that GeneralizationSet. A classifier cannot be both a transitively general and not a specific classifier.

Additional Operations

The query `specializeType()` determines whether a classifier may have a generalization relationship to classifiers of the specified type. It is intended to be redefined by classifiers that have different specialization constraints.

Classifier::specializeType(c: Classifier): Boolean;

\[\text{maySpecializeType = c oclIsKindOf(c.ocdType)}\]

A classifier is a classification of instances according to their features. A Classifier may participate in generalization relationships with other Classifiers. An instance of a specific Classifier is also an (indirect) instance of each of the general Classifiers. Therefore, features specified for instances of the general classifiers are implicitly specified for instances of the specific classifier. Any constraint applying to instances of the general classifier also applies to instances of the specific classifier.

The specific semantics of how generalization affects each concrete subtype of Classifier varies. All instances of a classifier have values corresponding to the classifier’s attributes.

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 notion of power type was inspired by the notion of power set. A power set is defined as a set whose instances are subsets. In essence, then, a power type is a class whose instances are subclasses. The powertypeExtent association relates a Classifier with a set of generalizations that a) have a common specific Classifier, and b) represent a collection of subsets for that class.

Semantic Variation Points

The precise lifecycle semantics of aggregation is a semantic variation point.

Notation

Classifier is an abstract model element, and so properly speaking has no notation. It is nevertheless convenient to define in one place a default notation available for any concrete subclass of Classifier for which this notation is suitable. The default notation for a classifier is a solid-outline rectangle containing the classifier’s name, and optionally with compartments separated by horizontal lines containing features or other members of the classifier. The specific type of classifier can be shown in guillemets above the name. Some specializations of Classifier have their own distinct notations.

Package PowerTypes

A classifier is a namespace whose members can include features. Classifier is an abstract metaclass.

Attributes

- isAbstract: Boolean
- Generalization
- “Namespace (from Kernel)” on page 99
- “Type (from Kernel)” on page 135

Associations

- /attribute: Property [*]
- /feature: Feature [*]
- /general: Classifier [*]

Generalization

A classifier may only specialize classifiers of a valid type.

Semantics

A classifier may participate in generalization relationships with other Classifiers. An instance of a specific Classifier is implicitly specified for instances of the specific classifier. Any constraint applying to instances of the general classifier also applies to instances of the specific classifier.

Style Guidelines

- Attribute names typically begin with a lowercase letter. Multi-word names are often formed by concatenating the words and using lowercase for all letters except for upcasing the first letter but the first.
- Center the name of the classifier in boldface.
- Center keyword (including stereotype names) in plain face within guillemets above the classifier name.
- For those languages that distinguish between uppercase and lowercase characters, capitalize names (i.e., begin them with an uppercase character).
- Left justify attributes and operations in plain face.
- Begin attribute and operation names with a lowercase letter.
- Show full attributes and operations when needed and suppress them in other contexts or references.
The attributes in Figure 7.30 are explained below.

**ClassA**
- name: String
- shape: Rectangle
  - + size: Integer [0..1]
  - / area: Integer (readOnly)
- height: Integer
- width: Integer

**ClassB**
- id (redefines name)
- shape: Square
- height = 7
- / width

An attribute can be shown as a text string. The format of this string is specified in the Notation sub-clause of “Property (from Kernel)” on page 135.

An attribute may also be shown using association notation, with no adornments at the tail of the arrow as shown in Figure 7.31.

**Style Guidelines**
- Attribute names typically begin with a lowercase letter. Multi-word names are often formed by concatenating the words Center keyword (including stereotype names) in plain face within guillemets above the classifier name.
- Attributes are single-valued except for those with multiplicity 0..*.
- Attributes may be shown using association notation, with no adornments at the tail of the arrow as shown in Figure 7.31.
The notion of power type was inspired by the notion of power set. A power set is defined as a set whose instances are subsets of the specified type. A power type can be used to define the set of all instances of the specified type, including all subtypes. For example, the Bank Account Type classifier could have a powertype association with a GeneralizationSet. This GeneralizationSet could then associate with two Generalizations where the class (i.e., general Classifier) Bank Account has two specific subclasses (i.e., Classifiers): Checking Account and Savings Account. Checking Account and Savings Account, then, are instances of the power type: Bank Account Type. In other words, Checking Account and Savings Account are both: instances of Bank Account Type, as well as subclasses of Bank Account. (For more explanation and examples, see Examples in the GeneralizationSet sub clause, below.)

### 7.3.9 Comment (from Kernel)

A comment is a textual annotation that can be attached to a set of elements.

#### Generalizations

- "Element (from Kernel)" on page 64.

#### Description

A comment gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler.

A comment can be owned by any element.

#### Attributes

- multiplicity: body: String {0..1}
  - Specifies a string that is the comment.

#### Associations

- annotatedElement: Element[*]
  - References the Element(s) being commented.

#### Constraints

- No additional constraints

#### Semantics

- A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

#### Notation

A Comment is shown as a rectangle with the upper right corner bent (this is also known as a “note symbol”). The rectangle contains the body of the Comment. The connection to each annotated element is shown by a separate dashed line.

#### Presentation Options

The dashed line connecting the note to the annotated element(s) may be suppressed if it is clear from the context, or not important in this diagram.

---

**Figure 7.30 - Example**

The attributes in:
- ClassA:4
  - Specifies a string that is the comment.

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.30 - Example**

The attributes in:
- ClassA:4
  - Specifies a string that is the comment.

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.30 - Example**

The attributes in:
- ClassA:4
  - Specifies a string that is the comment.

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.30 - Example**

The attributes in:
- ClassA:4
  - Specifies a string that is the comment.

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.30 - Example**

The attributes in:
- ClassA:4
  - Specifies a string that is the comment.

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.

---

**Figure 7.30 - Example**

The attributes in:
- ClassA:4
  - Specifies a string that is the comment.

**Figure 7.31 - Annotation window**

An attribute may be added to the model.

A Comment adds no semantics to the annotated elements, but may represent information useful to the reader of the model.
Meta Object Facility (MOF)
Open Questions...

- Now you’ve been “tricked”...
  - We didn’t tell what the **modelling language** for meta-modelling is.

- **Idea:** have a **minimal object-oriented core** comprising the notions of **class**, **association**, **inheritance**, etc. with “self-explaining” semantics.

- This is **Meta Object Facility (MOF)**, which (more or less) coincides with UML Infrastructure OMG (2007a).

- So: things on meta level
  - M0 are object diagrams/system states
  - M1 are **words of the language UML**
  - M2 are **words of the language MOF**
  - M3 are **words of the language MOF**
Benefits

- In particular:
  - Benefits for **Modelling Tools**.
  - Benefits for **Language Design**.
  - Benefits for **Code Generation and MDA**.
Meta-Modelling: Principle
Class name: Str
Property name: Str
Type name: Str

\[ \begin{align*}
C &: \text{Int} \\
&= \text{\{Int\}, \{C\}, \{v\}, \{C \mapsto v\}}, \\
&D \cong \Sigma D \\
&\in \sigma \\
&= \{u \mapsto \{v \mapsto 0\}\}
\end{align*} \]
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 $M$. 

\[ I = (\{\text{Int}\}, \{C\}, \{v\}, \{C \mapsto v\}), \\mathcal{D} \sim \sum \mathcal{D} \]
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.

Constraint example,

“[2] Generalization hierarchies must be directed and acyclic. A classifier cannot be both a transitively general and transitively specific classifier of the same classifier.

\[ \text{not self . allParents()} \to \text{includes(self)} \] (OMG, 2007b, 53)

- 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!
\[ \mathcal{C}, CD, SM \]

\[ \varphi \in \text{OCL} \]

\[ \mathcal{I} = (\mathcal{I}, \mathcal{C}, V, \text{at}), SM \]

\[ \mathcal{M} = (\Sigma, A, \mathcal{I}, \rightarrow_{SM}) \]

\[ \pi = (\sigma, \epsilon, 0) \overset{(\text{cons}, \text{Snd})}{\longrightarrow} (\sigma_1, \epsilon_1) \]

\[ w_\pi = ((\sigma_i, \text{cons}, \text{Snd})_{i \in N}) \]

\[ OD \]

\[ \text{OD} \]

\[ CD, SD \]

\[ B = (Q_{SD}, q_0, A, \mathcal{I}, \rightarrow_{SD}, F_{SD}) \]

\[ G = (N, E, f) \]
Software Design, Modelling and Analysis in UML

Lecture 1: Introduction

2016-10-18

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Content

- **Lecture 1**: Introduction
- **Lecture 2**: Semantical Model

## Contents & Goals

**Last Lecture:**
- Introduction: Motivation, Content, Formalia

**This Lecture:**
- **Educational Objectives**: Capabilities for following tasks/questions.
  - What is a signature, an object, a system state, etc.?
  - What is the purpose of signature, object, etc. in the course?
  - How do Basic Object System Signatures relate to UML class diagrams?

- **Content**:
  - Basic Object System Signatures
  - Structures
  - System States
**Content**

- **Lecture 1**: Introduction
- **Lecture 2**: Semantical Model
- **Lecture 3**: Object Constraint Language (OCL)

**Contents & Goals**

**Last Lecture:**
- Basic Object System Signature \( \mathcal{S} \) and Structure \( \mathcal{D} \), System State \( \sigma \in \Sigma_{\mathcal{S}} \)

**This Lecture:**
- **Educational Objectives**: Capabilities for these tasks/questions:
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this system state?
  - Give a system state satisfying this constraint?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are \( \mathcal{D}(C) \) and \( T(C) \) related?

- **Content**:
  - OCL Syntax
  - OCL Semantics (over system states)
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics

Contents & Goals

Last Lecture:
- OCL Syntax

This Lecture:
- Educational Objectives: Capabilities for these tasks/questions:
  - Please un-abbreviate all abbreviations in this OCL expression.
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this system state?
  - Give a system state satisfying this constraint?
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are $D(C)$ and $T_C$ related?

Content:
- OCL Semantics
- OCL Consistency and Satisfiability
Content

- **Lecture 1**: Introduction
- **Lecture 2**: Semantical Model
- **Lecture 3**: Object Constraint Language (OCL)
- **Lecture 4**: OCL Semantics
- **Lecture 5**: Object Diagrams
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
Content

• Lecture 1: Introduction
• Lecture 2: Semantical Model
• Lecture 3: Object Constraint Language (OCL)
• Lecture 4: OCL Semantics
• Lecture 5: Object Diagrams
• Lecture 6: Class Diagrams I
• Lecture 7: Class Diagrams II
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Class Diagrams II
- Lecture 8: Class Diagrams III

Last Lectures:
- completed class diagrams... except for associations.

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - Please explain this class diagram with associations.
  - Which annotations of an association arrow are semantically relevant?
  - What's a role name? What's it good for?
  - What is "multiplicity"? How did we treat them semantically?
  - What is "reading direction", "navigability", "ownership", ...?
  - What's the difference between "aggregation" and "composition"?

Content:
- Study concrete syntax for "associations".
- (Temporarily) extend signature, define mapping from diagram to signature.
- Study effect on OCL.
- Btw.: where do we put OCL constraints?
What is a class diagram?
When is an object diagram called partial? What are partial ones good for?
Could you please map this signature to a class diagram?
Please explain this OCL constraint.
Map class diagram to (extended) signature.
In what sense is OCL a three-valued logic? For what purpose?
OCL Consistency and Satisfiability
What is “reading direction”, “navigability”, “ownership”, . . . ?
What does it mean that an OCL expression is satisfiable?
Educational Objectives:
Associations in OCL syntax.
Capabilities for these tasks/questions:
completed class diagrams. . . except for associations.
– 6 – 2015-11-12 – Sprelim –

OCL Semantics
OCL Syntax
OCL: consistency, satisfiability
Content:
Please formalise this constraint in OCL.
Educational Objectives:
(What if things are missing?
Prepare (extend) definition of signature.
– 2 – 2015-10-22 – Sprelim –

What is the semantics of ‘abstract’?
When is an object diagram an object diagram (wrt. what)?
Lecture 2
System States
Content:
OCL Syntax
Lecture 9
Map class diagram to (extended) signature cont’d.
– 4 – 2015-11-03 – Sprelim –

Please explain this class diagram with associations.
Capabilities for following tasks/questions.
What is “multiplicity”? How did we treat them semantically?
Capabilities for these tasks/questions:
Please un-abbreviate all abbreviations in this OCL expression.
– 9 – 2015-12-01 – Sprelim –

Does this OCL constraint hold in this system state?
Basic Object System Signature
Content:
For what purposes are class diagrams useful?
partial vs. complete; for analysis; for documentation. . .
Introduction: Motivation, Content, Formalia
Lecture 7
2
Associations and OCL: semantics.
Capabilities for following tasks/questions.
Educational Objectives:
3
Which annotations of an association arrow are semantically relevant?
Could you please map this class diagram to a signature?
– 03 – 2014-10-29 – Sprelim –

Lecture 3
Please formalise this constraint in OCL.
Example: Object Diagrams for Documentation
Visibility as an extension of well-typedness.
Structures
Give a system state satisfying this constraint?
How are
Please explain this OCL constraint.
Basic Object System Signatures
Object Diagrams
Compute the value of a given OCL constraint in a system state with links.
Stereotypes – for documentation.
Lecture 5
How are
Please explain this OCL constraint.

Lecture 8
2
Educational Objectives:
– 5 – 2015-11-05 – Sprelim –

Educational Objectives:
What is a signature, an object, a system state, etc.?
Capabilities for following tasks/questions.
How do Basic Object System Signatures relate to UML class diagrams?
Educational Objectives:
– 8 – 2015-11-26 – Sprelim –

Lecture 1
Representing class diagrams as (extended) signatures — for the moment without
OCL Semantics
What’s a role name? What’s it good for?
. . .
Study effect on OCL.
Lecture 4
Lecture 6
Content:
Content:
What is visibility good for?
Lecture 1: Introduction
Lecture 2: Semantical Model
Lecture 3: Object Constraint Language (OCL)
Lecture 4: OCL Semantics
Lecture 5: Object Diagrams
Lecture 6: Class Diagrams I
Lecture 7: Class Diagrams II
Lecture 8: Class Diagrams III
Lecture 9: Class Diagrams IV
Lecture 10: State Machines Overview
Example: Object Diagrams for Documentation

What is "multiplicity"? How did we treat them semantically?

Ether

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

Associations syntax and semantics.

2

Capabilities for following tasks/questions.

OCL: consistency, satisfiability

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

– 2 – 2015-10-22 – Sprelim –

Can you please model the following behaviour.

Visibility as an extension of well-typedness.

Educational Objectives:

2

What is visibility good for?

Step, Run-to-Completion Step

– 3 – 2014-10-29 – Sprelim –

Content:

How are system states and object diagrams related?

Basic Object System Signature

What's the purpose of a behavioural model?

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

– 03 – 2014-10-29 – Sprelim –

Content:

Introduction: Motivation, Content, Formalia

Study concrete syntax for "associations".

Capabilities for these tasks/questions:

What if things are missing?

Content:

– 7 – 2015-11-17 – Sprelim –

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

Representing class diagrams as (extended) signatures — for the moment without

Can you think of an object diagram which violates this OCL constraint?

What does "navigability", "ownership", . . . mean?

Please un-abbreviate all abbreviations in this OCL expression.

Object Diagrams

Give a system state satisfying this constraint?

– 6 – 2015-11-12 – Sprelim –

Please explain this class diagram with associations.

Lecture 11

– 8 – 2015-11-26 – Sprelim –

Content:

Please formalise this constraint in OCL.

– 5 – 2015-11-05 – Sprelim –

Educational Objectives:

Which annotations of an association arrow are semantically relevant?

– 11 – 2015-12-10 – Sprelim –

Please explain this OCL constraint.

(Mostly) completed discussion of modelling

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

Capabilities for following tasks/questions.

partial vs. complete; for analysis; for documentation. . .

Could you please map this signature to a class diagram?

– 10 – 2015-11-05 – Sprelim –

Lecture 10

Associations and OCL: semantics.

Does this OCL constraint hold in this system state?

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

Study effect on OCL.

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

Lecture 11

– 11 – 2015-12-10 – Sprelim –

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

Could you please map this signature to a class diagram?

UML standard: basic causality model

Lecture 6

Associations in OCL syntax.

Give a system state satisfying this constraint?

What is a class diagram?

Capabilities for these tasks/questions:

2

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

Content:

– 6 – 2015-11-12 – Sprelim –

Please explain this OCL constraint.

– 11 – 2015-12-10 – Sprelim –

Please explain this OCL constraint.

– 8 – 2015-11-26 – Sprelim –

Content:

Please formalise this constraint in OCL.

– 7 – 2015-11-17 – Sprelim –

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

Representing class diagrams as (extended) signatures — for the moment without

Can you think of an object diagram which violates this OCL constraint?

What does "navigability", "ownership", . . . mean?

Please un-abbreviate all abbreviations in this OCL expression.

Object Diagrams

Give a system state satisfying this constraint?

– 6 – 2015-11-12 – Sprelim –

Please explain this OCL constraint.

(Mostly) completed discussion of modelling

What is "multiplicity"? How did we treat them semantically?

Ether

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

Associations syntax and semantics.

2

Capabilities for following tasks/questions.

OCL: consistency, satisfiability

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

– 2 – 2015-10-22 – Sprelim –

Can you please model the following behaviour.

Visibility as an extension of well-typedness.

Educational Objectives:

2

What is visibility good for?

Step, Run-to-Completion Step

– 3 – 2014-10-29 – Sprelim –

Content:

How are system states and object diagrams related?

Basic Object System Signature

What's the purpose of a behavioural model?

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

– 03 – 2014-10-29 – Sprelim –

Content:

Introduction: Motivation, Content, Formalia

Study concrete syntax for "associations".

Capabilities for these tasks/questions:

What if things are missing?

Content:

– 7 – 2015-11-17 – Sprelim –

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

Representing class diagrams as (extended) signatures — for the moment without

Can you think of an object diagram which violates this OCL constraint?

What does "navigability", "ownership", . . . mean?

Please un-abbreviate all abbreviations in this OCL expression.

Object Diagrams

Give a system state satisfying this constraint?

– 6 – 2015-11-12 – Sprelim –

Please explain this OCL constraint.

(Mostly) completed discussion of modelling

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

Capabilities for following tasks/questions.

partial vs. complete; for analysis; for documentation. . .

Could you please map this signature to a class diagram?

– 10 – 2015-11-05 – Sprelim –

Lecture 10

Associations and OCL: semantics.

Does this OCL constraint hold in this system state?

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

Study effect on OCL.

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

Lecture 11

– 11 – 2015-12-10 – Sprelim –

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

Could you please map this signature to a class diagram?

UML standard: basic causality model

Lecture 6

Associations in OCL syntax.

Give a system state satisfying this constraint?

What is a class diagram?

Capabilities for these tasks/questions:

2

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

Content:

– 6 – 2015-11-12 – Sprelim –

Please explain this OCL constraint.

– 11 – 2015-12-10 – Sprelim –

Please explain this OCL constraint.

– 8 – 2015-11-26 – Sprelim –

Content:

Please formalise this constraint in OCL.

– 7 – 2015-11-17 – Sprelim –

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

Representing class diagrams as (extended) signatures — for the moment without

Can you think of an object diagram which violates this OCL constraint?

What does "navigability", "ownership", . . . mean?

Please un-abbreviate all abbreviations in this OCL expression.

Object Diagrams

Give a system state satisfying this constraint?

– 6 – 2015-11-12 – Sprelim –

Please explain this OCL constraint.

(Mostly) completed discussion of modelling

What is "multiplicity"? How did we treat them semantically?

Ether

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

Associations syntax and semantics.

2

Capabilities for following tasks/questions.

OCL: consistency, satisfiability

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

– 2 – 2015-10-22 – Sprelim –

Can you please model the following behaviour.

Visibility as an extension of well-typedness.

Educational Objectives:

2

What is visibility good for?

Step, Run-to-Completion Step

– 3 – 2014-10-29 – Sprelim –

Content:

How are system states and object diagrams related?

Basic Object System Signature

What's the purpose of a behavioural model?

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

– 03 – 2014-10-29 – Sprelim –

Content:

Introduction: Motivation, Content, Formalia

Study concrete syntax for "associations".

Capabilities for these tasks/questions:

What if things are missing?

Content:

– 7 – 2015-11-17 – Sprelim –

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

Representing class diagrams as (extended) signatures — for the moment without

Can you think of an object diagram which violates this OCL constraint?

What does "navigability", "ownership", . . . mean?

Please un-abbreviate all abbreviations in this OCL expression.

Object Diagrams

Give a system state satisfying this constraint?

– 6 – 2015-11-12 – Sprelim –

Please explain this OCL constraint.

(Mostly) completed discussion of modelling

What is "multiplicity"? How did we treat them semantically?

Ether

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

Associations syntax and semantics.

2

Capabilities for following tasks/questions.

OCL: consistency, satisfiability

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

– 2 – 2015-10-22 – Sprelim –

Can you please model the following behaviour.

Visibility as an extension of well-typedness.

Educational Objectives:

2

What is visibility good for?

Step, Run-to-Completion Step

– 3 – 2014-10-29 – Sprelim –

Content:

How are system states and object diagrams related?

Basic Object System Signature

What's the purpose of a behavioural model?

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

– 03 – 2014-10-29 – Sprelim –

Content:

Introduction: Motivation, Content, Formalia

Study concrete syntax for "associations".

Capabilities for these tasks/questions:

What if things are missing?
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Class Diagrams II
- Lecture 8: Class Diagrams III
- Lecture 9: Class Diagrams IV
- Lecture 10: State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Class Diagrams II
- Lecture 8: Class Diagrams III
- Lecture 9: Class Diagrams IV
- Lecture 10: State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
Lecture 1: Introduction

Lecture 2: Semantical Model

Lecture 3: Object Constraint Language (OCL)

Lecture 4: OCL Semantics

Lecture 5: Object Diagrams

Lecture 6: Class Diagrams I

Lecture 7: Class Diagrams II

Lecture 8: Class Diagrams III

Lecture 9: Class Diagrams IV

Lecture 10: State Machines Overview

Lecture 11: Core State Machines I

Lecture 12: Core State Machines II

Lecture 13: Core State Machines III

Lecture 14: Hierarchical State Machines I
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Class Diagrams II
- Lecture 8: Class Diagrams III
- Lecture 9: Class Diagrams IV
- Lecture 10: State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
- Lecture 14: Hierarchical State Machines I
- Lecture 15: Hierarchical State Machines II

Contents & Goals

Last Lecture:
- step, RTC-step, divergence
- initial state, UML model semantics (so far)
- create, destroy actions

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - What is simple state, OR-state, AND-state?
  - What is a legal state configuration?
  - What is a legal transition?
  - How is enabledness of transitions defined for hierarchical state machines?

Content:
- Legal state configurations
- Legal transitions
- Rules (i) to (v) for hierarchical state machines
### Contents

- **Lecture 1**: Introduction
- **Lecture 2**: Semantical Model
- **Lecture 3**: Object Constraint Language (OCL)
- **Lecture 4**: OCL Semantics
- **Lecture 5**: Object Diagrams
- **Lecture 6**: Class Diagrams I
- **Lecture 7**: Class Diagrams II
- **Lecture 8**: Class Diagrams III
- **Lecture 9**: Class Diagrams IV
- **Lecture 10**: State Machines Overview
- **Lecture 11**: Core State Machines I
- **Lecture 12**: Core State Machines II
- **Lecture 13**: Core State Machines III
- **Lecture 14**: Hierarchical State Machines I
- **Lecture 15**: Hierarchical State Machines II
- **Lecture 16**: Hierarchical State Machines III
Could you please map this signature to a class diagram?

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

What is a cut, fired-set, etc.?

What about methods?

Towards semantics

Give a system state satisfying this constraint?

Educational Objectives:

OCL: consistency, satisfiability

Representing class diagrams as (extended) signatures — for the moment without

Content:

Capabilities for these tasks/questions:

– 03 – 2014-10-29 – Sprelim –

Lecture 14

What if things are missing?

What does it mean that an OCL expression is satisfiable?

Capabilities for following tasks/questions.

Example: Object Diagrams for Documentation

Educational Objectives:

Content:

How did we treat "multiplicity" semantically?

What about junction, choice, terminate, etc.?

What is a step / run-to-completion step?

Educational Objectives:

Could you please map this class diagram to a signature?

Does this OCL constraint hold in this system state?

Transformers

Basic Object System Signatures

Study concrete syntax for "associations".

How are

How do entry / exit actions work? What about do-actions?

OCL Syntax

Please un-abbreviate all abbreviations in this OCL expression.

Object Diagrams

Lecture 2

– 12 – 2015-12-15 – Sprelim –

What is a legal transition?

Remaining pseudo-states; deferred events

Content:

Capabilities for following tasks/questions.

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

Please formalise this constraint in OCL.

– 8 – 2015-11-26 – Sprelim –

Passive reactive objects

System configuration cont'd

What is a class diagram?

Content:

What are constructive and reflective descriptions of behaviour?

How is enabledness of transitions defined for hierarchical state machines?

Content:

Legal state configurations

Which annotations of an association arrow are semantically relevant?

Transformers

UML Core State Machines

Lecture 3

Entry / exit / do actions, internal transitions

OCL Semantics (over system states)

What is a legal state configuration?

What is simple state, OR-state, AND-state?

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

What is the idea of deferred events?

System States

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

Lecture 16

What is the abstract syntax of this LSC?

How is enabledness of transitions defined for hierarchical state machines?

Content:

Step, Run-to-Completion Step

– 7 – 2015-11-17 – Sprelim –

Educational Objectives:

Capabilities for following tasks/questions.

Content:

Lecture 13

Visibility as an extension of well-typedness.

OCL Consistency and Satisfiability

What is divergence in the context of UML models?

Capabilities for following tasks/questions.

3

Educational Objectives:

What are UML Interactions?

Is this UML model consistent with that OCL constraint?

What is the effect of shallow / deep history pseudo-states?

What is the semantics of 'abstract'?

Content:

Legal transitions

Maybe: hierarchical state machines

Capabilities for following tasks/questions.

2

Educational Objectives:

Could you please map this class diagram to a signature?

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

Deferred events

Transformers

UML Core State Machines

Lecture 3

Entry / exit / do actions, internal transitions

OCL Semantics (over system states)

What is a legal state configuration?

What is simple state, OR-state, AND-state?

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

What is the idea of deferred events?

System States

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

Lecture 16

What is the abstract syntax of this LSC?

How is enabledness of transitions defined for hierarchical state machines?

Content:

Step, Run-to-Completion Step

– 7 – 2015-11-17 – Sprelim –

Educational Objectives:

Capabilities for following tasks/questions.

Content:

Lecture 13

Visibility as an extension of well-typedness.

OCL Consistency and Satisfiability

What is divergence in the context of UML models?

Capabilities for following tasks/questions.

3

Educational Objectives:

What are UML Interactions?

Is this UML model consistent with that OCL constraint?

What is the effect of shallow / deep history pseudo-states?

What is the semantics of 'abstract'?

Content:

Legal transitions

Maybe: hierarchical state machines

Capabilities for following tasks/questions.

2

Educational Objectives:

Could you please map this class diagram to a signature?

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

Deferred events

Transformers

UML Core State Machines

Lecture 3

Entry / exit / do actions, internal transitions

OCL Semantics (over system states)

What is a legal state configuration?

What is simple state, OR-state, AND-state?

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

What is the idea of deferred events?

System States

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

Lecture 16

What is the abstract syntax of this LSC?

How is enabledness of transitions defined for hierarchical state machines?

Content:

Step, Run-to-Completion Step

– 7 – 2015-11-17 – Sprelim –

Educational Objectives:

Capabilities for following tasks/questions.

Content:

Lecture 13

Visibility as an extension of well-typedness.

OCL Consistency and Satisfiability

What is divergence in the context of UML models?
What's a role name? What's it good for?

Lecture 11 – 6 – 2015-11-12 – Sprelim –
Object Diagrams
UML Core State Machines
OCL Consistency and Satisfiability
Hierarchical state machines: the rest
Lecture 5

Content:
( Mostly) completed discussion of modelling

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

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

What is divergence in the context of UML models?

Study effect on OCL.

UML standard: basic causality model
Lecture 6

Could you please map this signature to a class diagram?

Introduction: Motivation, Content, Formalia
– 9 – 2015-12-01 – Sprelim –

Capabilities for following tasks/questions.

How are system states and object diagrams related?
Lecture 1
– 17 – 2016-01-21 – Sprelim –

2

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

2

Symbolic Automata
Entry / exit / do actions, internal transitions
What about junction, choice, terminate, etc.

Capabilities for following tasks/questions.
– 14 – 2016-01-12 – Sprelim –

2

What is "multiplicity"? How did we treat them semantically?
Object Diagrams
Capabilities for following tasks/questions.
Core State Machine syntax
Lecture 12
Capabilities for following tasks/questions.
Purposes of Behavioural Models
Educational Objectives:

– 10 – 2016-03-17 – Sprelim –

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

How are passive reactive objects treated in Rhapsody's UML semantics?
Lecture 10
– 2 – 2015-10-22 – Sprelim –

Stereotypes.

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

Could you please map this class diagram to a signature?
Lecture 8
Can you please model the following behaviour.

Content:
System configuration cont'd
Step, Run-to-Completion Step
– 03 – 2014-10-29 – Sprelim –

Behavioural features
structure

Is this UML model consistent with that OCL constraint?
Step, Run-to-Completion Step
Capabilities for these tasks/questions:
Capabilities for following tasks/questions.
Transformers
In what sense is OCL a three-valued logic? For what purpose?
Content:
Missing pieces: create / destroy transformer
What are UML Interactions?
Educational Objectives:
Lecture 9
Educational Objectives:

Content:
In what sense is OCL a three-valued logic? For what purpose?
Study UML syntax.
Step, Run-to-Completion Step
Please explain this OCL constraint.
Which annotations of an association arrow are semantically relevant?
Remaining pseudo-states; deferred events
Educational Objectives:
What is: Signal, Event, Ether, Transformer, Step, RTC.
– 7 – 2015-11-17 – Sprelim –

Content:
Give a system state satisfying this constraint?
Example: Object Diagrams for Documentation
Lecture 2
step, RTC-step, divergence

OCL Semantics (over system states)
What is a legal transition?
Step / RTC-Step revisited, Divergence
Deferred events
Educational Objectives:
This Lecture:

Capabilities for following tasks/questions.
Could you please map this signature to a class diagram?
Educational Objectives:
Basic causality model
Map class diagram to (extended) signature cont'd.
Prepare (extend) definition of signature.
– 16 – 2016-01-19 – Sprelim –

LSC syntax
What is: Signal, Event, Ether, Transformer, Step, RTC.
What is a cut, fired-set, etc.
Capabilities for following tasks/questions.
initial state, UML model semantics (so far)
Lecture 18
3
Contents & Goals

Last Lecture:
- Symbolic Büchi Automata
- Language of a UML Model
- Cuts

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - How is the semantics of LSCs constructed?
  - What is a cut, fired-set, etc.?
  - Construct the TBA for this LSC.
  - Give one example which (non-)trivially satisfies this LSC.

- Content:
  - Cut Examples, Firedset
  - Automaton construction
  - Transition annotations
  - Forbidden scenarios

Lecture 1: Introduction
Lecture 2: Semantical Model
Lecture 3: Object Constraint Language (OCL)
Lecture 4: OCL Semantics
Lecture 5: Object Diagrams
Lecture 6: Class Diagrams I
Lecture 7: Class Diagrams II
Lecture 8: Class Diagrams III
Lecture 9: Class Diagrams IV
Lecture 10: State Machines Overview
Lecture 11: Core State Machines I
Lecture 12: Core State Machines II
Lecture 13: Core State Machines III
Lecture 14: Hierarchical State Machines I
Lecture 15: Hierarchical State Machines II
Lecture 16: Hierarchical State Machines III
Lecture 17: Live Sequence Charts I
Lecture 18: Live Sequence Charts II
Lecture 19: Live Sequence Charts III

What does "navigability", "ownership", . . . mean?
Can you please model the following behaviour.
Representing class diagrams as (extended) signatures — for the moment without

Educational Objectives:
Content:
When is a set of OCL constraints said to be consistent?
Give a system state satisfying this constraint?
UML standard: basic causality model
Rules (i) to (v) for hierarchical state machines
Purposes of Behavioural Models
What is a cut, fired-set, etc.?
UML Core State Machines
Visibility as an extension of well-typedness.
How is the semantics of LSCs constructed?
Basic Object System Signature
How is enabledness of transitions defined for hierarchical s tate machines?
Prepare (extend) definition of signature.
Ether
Interactions: Live Sequence Charts
Transitions by Rule (i) to (v).
Action language and transformer
Step / RTC-Step revisited, Divergence
Example: Object Diagrams for Documentation
When is an object diagram called partial? What are partial on es good for?
Legal transitions
Content:
OCL Syntax
Entry / exit / do actions, internal transitions
How are
initial state, UML model semantics (so far)
Lecture 7
Capabilities for following tasks/questions.
What is divergence in the context of UML models?
2 – 16 – 2016-01-19 – Sprelim –
Associations syntax and semantics.
What is: Signal, Event, Ether, Transformer, Step, RTC.
2
What is "reading direction", "navigability", "ownership" , . . . ?
Object Diagrams
Associations: the rest.
Educational Objectives:
Legal state configurations
2
What is the idea of deferred events?
Educational Objectives:
Capabilities for following tasks/questions.

What is the purpose of signature, object, etc. in the course?
Capabilities for following tasks/questions.
Lecture 17
Capabilities for following tasks/questions.
Step, Run-to-Completion Step
Content:
Does this OCL constraint hold in this system state?
Symbolic Automata
Capabilities for following tasks/questions.
Please un-abbreviate all abbreviations in this OCL express ion.
Construct the TBA for this LSC.
– 7 – 2015-11-17 – Sprelim –
Capabilities for following tasks/questions.
Which annotations of an association arrow are semantically relevant?
Please explain this OCL constraint.
How is the semantics of LSCs constructed?
What makes a class diagram a good class diagram?
– 03 – 2014-10-29 – Sprelim –
Lecture 15
Capabilities for following tasks/questions.
Educational Objectives:
(Mostly) completed discussion of modelling
Core State Machine syntax
Content:
Passive reactive objects
How do Basic Object System Signatures relate to UML class dia grams?
Lecture 5
Lecture 9
– 10 – 2015-12-03 – Sprelim –
Please un-abbreviate all abbreviations in this OCL express ion.
System configuration cont’d
– 2 – 2015-10-22 – Sprelim –
How are passive reactive objects treated in Rhapsody's UML s emantics?
What's a role name? What's it good for?
Lecture 12
Content:
What is a legal state configuration?
Lecture 8
– 12 – 2015-12-15 – Sprelim –
– 14 – 2016-01-12 – Sprelim –
Compute the value of a given OCL constraint in a system state w ith links.
What is visibility good for?
Content:
Study effect on OCL.
What does this State Machine mean? What happens if I inject th is event?
Educational Objectives:
Can you think of an object diagram which violates this OCL con straint?
Capabilities for following tasks/questions.
Transformers
Capabilities for following tasks/questions.
What is the abstract syntax of this LSC?
– 17 – 2016-01-21 – Sprelim –
Legal transitions
– 3 – 2016-01-19 – Sprelim –

Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Class Diagrams II
- Lecture 8: Class Diagrams III
- Lecture 9: Class Diagrams IV
- Lecture 10: State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
- Lecture 14: Hierarchical State Machines I
- Lecture 15: Hierarchical State Machines II
- Lecture 16: Hierarchical State Machines III
- Lecture 17: Live Sequence Charts I
- Lecture 18: Live Sequence Charts II
- Lecture 19: Live Sequence Charts III
- Lecture 20: Live Sequence Charts IV
Content

- **Lecture 1**: Introduction
- **Lecture 2**: Semantical Model
- **Lecture 3**: Object Constraint Language (OCL)
- **Lecture 4**: OCL Semantics
- **Lecture 5**: Object Diagrams
- **Lecture 6**: Class Diagrams I
- **Lecture 7**: Class Diagrams II
- **Lecture 8**: Class Diagrams III
- **Lecture 9**: Class Diagrams IV
- **Lecture 10**: State Machines Overview
- **Lecture 11**: Core State Machines I
- **Lecture 12**: Core State Machines II
- **Lecture 13**: Core State Machines III
- **Lecture 14**: Hierarchical State Machines I
- **Lecture 15**: Hierarchical State Machines II
- **Lecture 16**: Hierarchical State Machines III
- **Lecture 17**: Live Sequence Charts I
- **Lecture 18**: Live Sequence Charts II
- **Lecture 19**: Live Sequence Charts III
- **Lecture 20**: Live Sequence Charts IV
- **Lecture 21**: MBSE & Inheritance
Content

- Lecture 1: Introduction
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Class Diagrams II
- Lecture 8: Class Diagrams III
- Lecture 9: Class Diagrams IV
- Lecture 10: State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
- Lecture 14: Hierarchical State Machines I
- Lecture 15: Hierarchical State Machines II
- Lecture 16: Hierarchical State Machines III
- Lecture 17: Live Sequence Charts I
- Lecture 18: Live Sequence Charts II
- Lecture 19: Live Sequence Charts III
- Lecture 20: Live Sequence Charts IV
- Lecture 21: MBSE & Inheritance
- Lecture 22: Meta-Modelling
References


OMG (2003). Uml 2.0 proposal of the 2U group, version 0.2, [http://www.2uworks.org/uml2submission](http://www.2uworks.org/uml2submission).


