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?
Abstract Syntax

A signature with inheritance is a tuple

$$\mathcal{S} = (\mathcal{T}, \mathcal{C}, V, \text{atr}, E, F, mth, \triangleright)$$

where

- $$(\mathcal{T}, \mathcal{C}, V, \text{atr}, E)$$ is a signature with signals and behavioural features ($$F/mth$$ are methods, analogous to $$V/\text{atr}$$ attributes), and
- $$\triangleright \subseteq (\mathcal{C} \times \mathcal{C}) \cup (E \times E)$$ is an acyclic generalisation relation, i.e. $$C \triangleright^+ C$$ 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
Common graphical representations of $\leq \{ (C, D_1), (C, D_2) \}$:

```
C -> D_1
C -> D_2
```

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

```
C_0 \triangleleft C_1
\downarrow
C_1 \triangleleft D_2
\downarrow
C_2
```

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:


“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.”

---

**Static Sub-Typing**

In FrontEnd’s state machine:

```
01
 itsSession := new Session

02
 /itsSession := new VIPSession

03
 /itsSession.dump()
```

```
\begin{align*}
S_0 \quad & S_1 \\
\text{OK, } & T_1 \downarrow T_2
\end{align*}
```
System States with Inheritance

**Wanted:** a formal representation of "if $C \sqsubseteq^* D$ then $D$ is a $C$", that is,

1. $D$ has the same attributes and behavioural features as $C$, and
2. $D$ objects (identities) can replace $C$ objects.

**Two approaches to semantics:**

- **Domain-inclusion Semantics**

![Diagram of Domain-inclusion Semantics]

- **Uplink Semantics**

![Diagram of Uplink Semantics]
Inheritance and State-Machines: Example

\[ \varepsilon = (\varepsilon, \varepsilon, \varepsilon) \]

\[ \eta = (\varepsilon, \varepsilon, \varepsilon) \]

\[ \eta = (\varepsilon, \varepsilon, \varepsilon) \]

\[ n = (\varepsilon, \varepsilon, \varepsilon) \]

\[ n = (\varepsilon, \varepsilon, \varepsilon) \]

\[ \eta = (\varepsilon, \varepsilon, \varepsilon) \]

\[ \eta = (\varepsilon, \varepsilon, \varepsilon) \]

\[ \eta = (\varepsilon, \varepsilon, \varepsilon) \]

\[ \eta = (\varepsilon, \varepsilon, \varepsilon) \]
(ii) Dispatch

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

if

- \( u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \wedge \exists u_E \in \mathcal{D}(E) : u_E \in \text{ready}(\varepsilon, u) \)
- \( u \) 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.

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

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

\[
\epsilon, \delta \in (\mathcal{E}(\text{cons}(u, \text{params}_E \mapsto u_E) / u_E))
\]

and

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

\[
(\sigma'', \varepsilon') \in t_{\text{act}}[u](\bar{\sigma}, \varepsilon \ominus u_E), \quad \sigma' = (\sigma''[u, \text{stable} \mapsto \text{stable}] \ominus u_E)
\]

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

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

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

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:

"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**

- Teacher
- Student
- Genius
- Polite
- Clown
- GenStWorker

**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
Classes (OMG, 2007b, 32)

Figure 7.12 - Classes diagram of the Kernel package
Operations (OMG, 2007b, 31)

Figure 7.10 - Features diagram of the Kernel package

Figure 7.11 - Operations diagram of the Kernel package
Classifiers (OMG, 2007b, 29)

Figure 7.9 - Classifiers diagram of the Kernel package
**Namespaces** (OMG, 2007b, 26)

Figure 7.4 - Namespaces diagram of the Kernel package

**Root Diagram** (OMG, 2007b, 25)

Figure 7.3 - Root diagram of the Kernel package
Reading the Standard

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

UML Superstructure Specification, v2.1.2
A classifier is a type and can own generalizations, thereby making it possible to define generalization relationships to
other classes. A classifier is redefinable element, meaning that it is possible to redefine nested classifiers.

• /attribute: Property [*]

Type (from Kernel) on page 135

If classifier is intended to be used by other classifiers (e.g., as the target of general metarelationships or generalization relationships), the Classifier does not provide a complete declaration and can typically not be instantiated. An abstract classifier is a classifier that does not provide a complete declaration and can typically not be instantiated.
A classifier is a classification of instances, it describes a set of instances that have features in common. 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**
  - \textit{isAbstract}: Boolean
  - \textit{general} : Classifier[*]
  - \textit{feature} : Feature [*]
  - \textit{general} : Classifier
  - \textit{feature} : Feature

- **Associations**
  - \textit{inheritedMember}: NamedElement[*]
  - \textit{general} : Classifier
  - \textit{feature} : Feature

- **Package Dependencies**
  - \textit{substitutes}: Substitution
  - \textit{substituted}: Substitution
  - \textit{inherits}: Inheritance

- **Constraints**
  - The general classifiers are the classifiers referenced by the generalization relationships.
  - Generalization function must be declared and used. A classifier must be a specific or general classifier of the same classifier.

- **Additional Operations**
  - The query \texttt{allParents()} gives all of the feature values of the classifier. In general, through mechanisms such as inheritance, this will be a larger set than feature. Classifier\{\texttt{allParents()}\} [\texttt{inherit}]

- **Semantics**
  - A classifier is a classification of instances according to their features.

- **References**
  - "Namespace (from Kernel)" on page 99
  - "RedefinableElement (from Kernel)" on page 130

- **Designates**
  - The GeneralizationSet of which the associated Classifier is a power type.

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Designates**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets

- **Refers to**
  - All of the Properties that are direct (i.e., not inherited or imported) attributes of the classifier. Subsets
7.3.8 Classifier (from Kernel, Dependencies, PowerTypes)

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

Generalizations
• isAbstract: Boolean

Attributes
Attributes

Associations
• / general : Classifier[*]
• / feature : Feature[*]

xWin: XWindow
protected
defaultSize: Rectangle
private
hide()
public

"RedefinableElement (from Kernel)" on page 130
"Namespace (from Kernel)" on page 99

If classifier is intended to be used by other classifiers (e.g., as the target of general metarelationships or generalization of another classifier), it may define a generalization hierarchy. A classifier can specify a generalization hierarchy by referencing its general classifiers.

Objects

If an object is a generalized classifier, it can inherit from these generalizations. The set of generalizations for a classifier is the union of all generalizations from its general classifiers and its general classifiers (if any).

Additional Operations
Classifier::parents(): Set(Classifier);
Classifier::allFeatures(): Set(Feature);
Classifier::inherit(inhs: Set(NamedElement)): Set(NamedElement);
Classifier::inheritableMembers(c: Classifier): Set(NamedElement);
Classifier::conformsTo(type: Classifier): Boolean;
Classifier::hasVisibilityOf(n: NamedElement): Boolean;

Semantic Variation Points
The precise lifecycle semantics of aggregation is a semantic variation point. It is not specified here and is intended to be left flexible.

Style Guidelines

Attributes
• Attribute names typically begin with a lowercase letter. Multi-word names are often formed by concatenating the words (e.g., name shortened with an uppercase character).
• Show full attributes and operations when needed and suppress them in other contexts or references.

Operations
• Operations on classifiers may be shown using OCL for the operations and the argument type(s) of the operation.
• Methods (e.g., accessor or mutator methods) may be shown as operations.

Notation
• Center keyword (including stereotype names) in plain face within guillemets above the classifier name.
• Show all classifiers that begin a block of OCL (e.g., a query).

References
• The query parents() gives all of the immediate ancestors of a generalized Classifier.
• The query allParents() gives all of the direct and indirect ancestors of a generalized Classifier.
• The query inheritableMembers() gives all of the members of a classifier that may be inherited in one of its descendants.
• The query conformsTo() gives true for a classifier that defines a type that conforms to another. This is used, for example, to determine if a classifier is a supertype of another classifier.
• The query hasVisibilityOf() determines whether a named element is visible in the classifier. By default all are visible. It is not necessary to use this query for instance names.
• The query area: Integer {readOnly}

Figure 7.31 - Association-like notation for attribute
A classifier is a redefinable element, meaning that it is possible to redefine nested classifiers.

- **isAbstract**: Boolean
- **attributes**: Property [*]
- **general**: Classifier [*]

Generalization hierarchies must be directed and acyclical. A classifier cannot be both a transitively general and

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

```
self.inheritedMember->includesAll(self.inherit(self.parents()->collect(p | p.inheritableMembers(self))))
```

### Style Guidelines
- **name**: Capitalize the first letter of each word but the first.
- **center keyword (including stereotype names)**: Within guillemets above the classifier name.
- **left justify attributes and operations**: In plain face.

### Examples
- **ClassA**: A classifier is a redefinable element. It is used to group similar things.
  - **shape**: Rectangle
  - **name**: String
  - **area**: Integer, read-only

- **ClassB**: A classifier that redefines `ClassA`.
  - **shape**: Square, specialization of `Rectangle`
  - **id**: String

### Presentation
```
Classifier::allFeatures():
  self.inheritedMember->includesAll(self.inherit(self.parents()->collect(p | p.inheritableMembers(self))))
```

### Design Guidelines
- **any compartment**: May be suppressed. A separator line is not drawn for a suppressed compartment. If a compartment is

## Meta Object Facility (MOF)

Meta Object Facility (MOF) is a facility for defining and managing metamodels.
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 ...

Benefits

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

\[ C \vdash v : \text{Int} \]

\[ S = (\{ \text{Int} \}, \{C\}, \{v\}, \{C \mapsto v\}) \]

\[ \in \]

\[ \sigma = \{u \mapsto \{v \mapsto 0\}\} \]
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 acyclical. A classifier cannot be both a transitively general and transitively specific classifier of the same classifier.

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

The Map

UML

Model

Instances

\varphi \in \text{OCL}

\mathcal{S} = (T, C, V, \text{atr})

\mathcal{M} = (\Sigma D, A S, \rightarrow \mathcal{M})

\pi = (\sigma_0, \epsilon_0, \text{cons} 0, \text{Snd} 0) \rightarrow \cdots \rightarrow u_0 (\sigma_1, \epsilon_1, \text{cons} 1, \text{Snd} 1)

\mathcal{G} = (N, E, f)

\mathcal{S} = (\mathcal{P}, \mathcal{E}, V, \text{atr}), \mathcal{M}

Mathematics

\odot

UML

\varphi \in \text{OCL}

\mathcal{S} = (T, C, V, \text{atr})

\mathcal{M} = (\Sigma D, A S, \rightarrow \mathcal{M})

\pi = (\sigma_0, \epsilon_0, \text{cons} 0, \text{Snd} 0) \rightarrow \cdots \rightarrow u_0 (\sigma_1, \epsilon_1, \text{cons} 1, \text{Snd} 1)

\mathcal{G} = (N, E, f)

\mathcal{S} = (\mathcal{P}, \mathcal{E}, V, \text{atr}), \mathcal{M}

\varphi \in \text{OCL}

\mathcal{S} = (T, C, V, \text{atr})

\mathcal{M} = (\Sigma D, A S, \rightarrow \mathcal{M})

\pi = (\sigma_0, \epsilon_0, \text{cons} 0, \text{Snd} 0) \rightarrow \cdots \rightarrow u_0 (\sigma_1, \epsilon_1, \text{cons} 1, \text{Snd} 1)

\mathcal{G} = (N, E, f)

\mathcal{S} = (\mathcal{P}, \mathcal{E}, V, \text{atr}), \mathcal{M}

\varphi \in \text{OCL}

\mathcal{S} = (T, C, V, \text{atr})

\mathcal{M} = (\Sigma D, A S, \rightarrow \mathcal{M})

\pi = (\sigma_0, \epsilon_0, \text{cons} 0, \text{Snd} 0) \rightarrow \cdots \rightarrow u_0 (\sigma_1, \epsilon_1, \text{cons} 1, \text{Snd} 1)

\mathcal{G} = (N, E, f)

\mathcal{S} = (\mathcal{P}, \mathcal{E}, V, \text{atr}), \mathcal{M}

\varphi \in \text{OCL}

\mathcal{S} = (T, C, V, \text{atr})

\mathcal{M} = (\Sigma D, A S, \rightarrow \mathcal{M})

\pi = (\sigma_0, \epsilon_0, \text{cons} 0, \text{Snd} 0) \rightarrow \cdots \rightarrow u_0 (\sigma_1, \epsilon_1, \text{cons} 1, \text{Snd} 1)

\mathcal{G} = (N, E, f)
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
Contents & Goals

Last Lecture:
- Basic Object System Signature \( S \) and Structure \( D \), System State \( \sigma \in \Sigma \)

This Lecture:
- Educational Objectives: Capabilities for these tasks/questions:
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does the 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 Syntax
- OCL Semantics (over system states)

Contents & Goals

Last Lecture:
- OCL Syntax

This Lecture:
- Educational Objectives: Capabilities for these tasks/questions:
  - Please unabbreviate all abbreviations in this OCL expression.
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does the 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 Syntax
- OCL Semantics

Contents & Goals

Last Lecture:
- OCL Syntax

This Lecture:
- Educational Objectives: Capabilities for these tasks/questions:
  - Please unabbreviate all abbreviations in this OCL expression.
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does the 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 Syntax
- 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 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

Educational Objectives:
- OCL Semantics
- Basic Object System Signatures
- OCL: consistency, satisfiability
- Object Diagrams
- How are system states and object diagrams related?
- How do Basic Object System Signatures relate to UML class diagrams?
- Example: Object Diagrams for Documentation

Last Lecture:
- OCL Syntax
- OCL Consistency and Satisfiability
- Study UML syntax.
- When is an object diagram an object diagram (wrt. what)?
- What is a class diagram?
- What is an object diagram? What are object diagrams good for?
- How do these tasks/questions:
  - Map class diagram to (extended) signature.
  - Prepare (extended) definition of signature.
  - What is the purpose of signature, object, etc. in the course?
  - How are these tasks/questions:
    - Map class diagram to (extended) signature.
    - Prepare (extended) definition of signature.
    - What is the purpose of signature, object, etc. in the course?

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - What does it mean that an OCL expression is satisfiable?
  - What is the purpose of signature, object, etc. in the course?
  - What is the purpose of signature, object, etc. in the course?
  - What is the purpose of signature, object, etc. in the course?
  - Could you please map this class diagram to a signature?
  - Could you please map this class diagram to a signature?
  - Could you please map this class diagram to a signature?
  - Could you please map this class diagram to a signature?
  - Could you please map this class diagram to a signature?

Content:
- OCL: consistency, satisfiability
- Object Diagrams
- Example: Object Diagrams for Documentation

Please formalise this constraint in OCL.

Please un-abbreviate all abbreviations in this OCL expression.

Please un-abbreviate all abbreviations in this OCL expression.

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

Example: Object Diagrams for Documentation

Prepare (extend) definition of signature.

Describe the relationship between Basic Object System Signatures and UML class diagrams.

Give a system state satisfying this constraint?

Does this OCL constraint hold in this system state?

Could you please map this signature to a class diagram?

Example: Object Diagrams for Documentation

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

Lecture 1
Introduction: Motivation, Content, Formalia
- OCL: consistency, satisfiability
- Object Diagrams

Lecture 2
Object Diagrams
- How are system states and object diagrams related?
- How do Basic Object System Signatures relate to UML class diagrams?
- Example: Object Diagrams for Documentation

Lecture 3
Object Constraint Language (OCL)
- What is an object diagram? What are object diagrams good for?
- How do these tasks/questions:
  - Map class diagram to (extended) signature.
  - Prepare (extended) definition of signature.
  - What is the purpose of signature, object, etc. in the course?
  - How are these tasks/questions:
    - Map class diagram to (extended) signature.
    - Prepare (extended) definition of signature.
    - What is the purpose of signature, object, etc. in the course?

Lecture 4
OCL Semantics
- What is the purpose of signature, object, etc. in the course?
- Could you please map this class diagram to a signature?
- Could you please map this class diagram to a signature?
- Could you please map this class diagram to a signature?
- Could you please map this class diagram to a signature?
- Could you please map this class diagram to a signature?

Lecture 5
Object Diagrams
- How are system states and object diagrams related?
- How do Basic Object System Signatures relate to UML class diagrams?
- Example: Object Diagrams for Documentation

Lecture 6
Class Diagrams I
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

OCL Semantics

- Last Lecture:
  - Representing class diagrams as (extended) signatures — for the moment without associations

- This Lecture:
  - Educational Objectives: Capabilities for following tasks/questions:
    - Could you please map this class diagram to a signature?
    - Could you please map this signature to a class diagram?
    - What is the semantics of “abstract”?
    - What is variability good for?

  - Content:
    - Map class diagram to (extended) signature cont’d
    - Stereotypes: for documentation
    - Visibility as an extension of well-typedness
    - Completing class diagrams — except for associations

- This Lecture:
  - Educational Objectives: Capabilities for following tasks/questions:
    - Could you please map this signature to a class diagram?
    - Could you please map this class diagram to a signature?

- This Lecture:
  - Educational Objectives: Capabilities for these tasks/questions:
    - What does it mean that an OCL expression is satisfiable?
    - How are system states and object diagrams related?
    - What if things are missing?
    - Does this OCL constraint hold in this system state?
**Contents & Goals**

**Last Lecture:**
- Association syntax and semantics.
- Association in OCL syntax.

**Title Lecture:**
- Educational Objectives: Capabilities for following tasks/questions.
- What is “multiplicity”? How did we treat them semantically?
- What does “navigability”, “ownership”, … mean?
- …

**Content:**
- Association and OCL semantics.
- Associations: the rest.

---

**Contents & Goals**

**Last Lecture:**
- ( Mostly) completed discussion of modeling structure.

**Title Lecture:**
- Educational Objectives: Capabilities for following tasks/questions.
- What is the purpose of a behavioral model?
- How do we treat “multiplicity” semantically?
- What does “navigability”, “ownership”, … mean?
- …

**Content:**
- For completeness: Modelling Guidelines for Class Diagrams
- Programs of Behavioral Models
- UML Core State Machines

---

**Contents & Goals**

**Last Lecture:**
- Association syntax and semantics.
- Association in OCL syntax.

**Title Lecture:**
- Educational Objectives: Capabilities for following tasks/questions.
- What is “multiplicity”? How did we treat them semantically?
- What does “navigability”, “ownership”, … mean?
- …

**Content:**
- Association and OCL semantics.
- Associations: the rest.

---

**Contents & Goals**

**Last Lecture:**
- Association syntax and semantics.
- Association in OCL syntax.

**Title Lecture:**
- Educational Objectives: Capabilities for following tasks/questions.
- What is “multiplicity”? How did we treat them semantically?
- What does “navigability”, “ownership”, … mean?
- …

**Content:**
- Association and OCL semantics.
- Associations: the rest.

---

**Contents & Goals**

**Last Lecture:**
- Association syntax and semantics.
- Association in OCL syntax.

**Title Lecture:**
- Educational Objectives: Capabilities for following tasks/questions.
- What is “multiplicity”? How did we treat them semantically?
- What does “navigability”, “ownership”, … mean?
- …

**Content:**
- Association and OCL semantics.
- Associations: the rest.
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
Educational Objectives:

Associations in OCL syntax.

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

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

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

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

Content:
- Legal state configurations
- Legal transitions
- Rule (i) to (v) for hierarchical state machines

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

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

Content:
- Legal state configurations
- Legal transitions
- Rule (i) to (v) for hierarchical state machines
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 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

Rhapsody code generation

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

Structures

Content:

Map class diagram to (extended) signature.

OCL Semantics (over system states)

How did we treat "multiplicity" semantically?

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

Behavioural features

Capabilities for following tasks/questions.

Lecture 19

Associations and OCL: semantics.

Can you please model the following behaviour.

Missing pieces: create / destroy transformer

What is a legal state configuration?

Lecture 10

Content:

Lecture 16

create, destroy actions

Language of a UML Model

Interactions: Live Sequence Charts

Can you please model the following behaviour.

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

Give one example which (non-)trivially satisfies this LSC.

Does this OCL constraint hold in this system state?

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

2

What are UML Interactions?

Lecture 9

Purposes of Behavioural Models

Legal state configurations

What is a class diagram?

Stereotypes – for documentation.

Transformers

Educational Objectives:

Could you please map this signature to a class diagram?

Deferred events

Content:

Purposes of Behavioural Models

Could you please map this class diagram to a signature?

Lecture 13

Object Diagrams

Capabilities for following tasks/questions.

Can you please model the following behaviour.

Hierarchical state machines: the rest

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

Capabilities for following tasks/questions.

For what purposes are class diagrams useful?

Action language and transformer

Btw.: where do we put OCL constraints?

2

Please formalise this constraint in OCL.

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

For what purposes are class diagrams useful?

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

What is the subset / uplink semantics of inheritance?

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

What is divergence in the context of UML models?

What about junction, choice, terminate, etc.?

Lecture 7

Does this OCL constraint hold in this system state?

2

Capabilities for following tasks/questions.

What is the abstract syntax of this LSC?

Please formalise this constraint in OCL.

Is this UML model consistent with that OCL constraint?

Educational Objectives:

(G)Mostly) completed discussion of modelling

2

Lecture 18

Transition annotations

Lecture 14

Transitions by Rule (i) to (v).

Content:

What are UML Interactions?

How is enabledness of transitions defined for hierarchical state machines?

Content:

Content:

Please un-abbreviate all abbreviations in this OCL expression.

Capabilities for following tasks/questions.

2

What about methods?

Educational Objectives:

Basic Object System Signatures

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

What makes a class diagram a good class diagram?

Capabilities for these tasks/questions:

Lecture 3

Capabilities for following tasks/questions.

Step / RTC-Step revisited, Divergence

What if things are missing?

Content:

Educational Objectives:

This Lecture:

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

Capabilities for following tasks/questions.

2

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

For what purposes are class diagrams useful?

What is the State Name? What happens if I inject this event?

What is the subset / uplink semantics of inheritance?

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

What is divergence in the context of UML models?

What about junction, choice, terminate, etc.?

Lecture 7

Does this OCL constraint hold in this system state?

2

Capabilities for following tasks/questions.

What is the abstract syntax of this LSC?

Please formalise this constraint in OCL.

Is this UML model consistent with that OCL constraint?
What is: Signal, Event, Ether, Transformer, Step, RTC.

Educational Objectives:
Basic causality model
Initial states

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

Purposes of Behavioural Models
Rhapsody code generation
Representing class diagrams as (extended) signatures — for the moment without Transition annotations

What is divergence in the context of UML models?
What is a cut, fired-set, etc.?

What if things are missing?

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

The UML Meta Model

Educational Objectives:

OCL Consistency and Satisfiability

Educational Objectives:

Passive reactive objects
Two approaches to obtain desired semantics

Educational Objectives:
How are system states and object diagrams related?

Capabilities for following tasks/questions.
Legal state configurations
Associations and OCL: semantics.
When is an object diagram an object diagram (wrt. what)?

Educational Objectives:

What is the abstract syntax of this LSC?

Lecture 22
What are constructive and reflective descriptions of behaviour?
Can you please model the following behaviour.
How are passive reactive objects treated in Rhapsody's UML semantics?

Capabilities for following tasks/questions.
Visibility as an extension of well-typedness.
Give a system state satisfying this constraint?
Cuts

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

Give one example which (non-)trivially satisfies this LSC.

Transformers

Content:

Symbolic Büchi Automata

UML standard: basic causality model
For completeness: Modelling Guidelines for Class Diagrams

How is enabledness of transitions defined for hierarchical state machines?
step, RTC-step, divergence

Lecture 15
Lecture 12
Lecture 5
Lecture 10
Lecture 9
Lecture 8
Lecture 7
Lecture 6
Lecture 5
Lecture 4
Lecture 3
Lecture 2
Lecture 1

Last Lecture:

Contents & Goals

This Lecture:

Contents & Goals
References


