

# *Software Design, Modelling and Analysis in UML*

## *Lecture 21: Meta-Modelling*

2016-02-11

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

– 21 – 2016-02-11 – main –

### *Contents & Goals*

#### **Last Lecture:**

- Liskov Substitution Principle
- Inheritance: Domain Inclusion Semantics

#### **This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is the idea of meta-modelling?
  - How does meta-modelling relate to UML?
- **Content:**
  - The UML Meta Model
  - Wrapup & Questions

– 21 – 2016-02-11 – Prelim –

## *Meta-Modelling: Idea*

## *Meta-Modelling: Why and What*

- **Meta-Modelling** is one major prerequisite for understanding
  - the standard documents [OMG \(2007a,b\)](#), 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 **model** UML models using a modelling language?
- In other words:  
Why not have a model  $\mathcal{M}_U$  such that
  - the set of legal instances of  $\mathcal{M}_U$is
  - the set of well-formed (!) UML models.

## Meta-Modelling: Example $D(V) = \{+, -, \neq\}$

For example, let's consider a class.

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

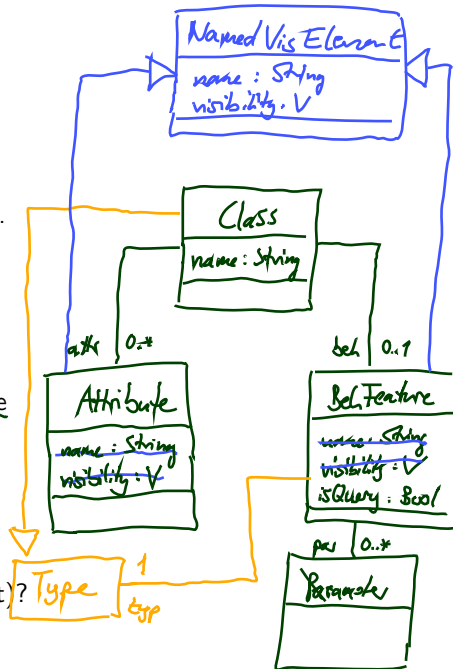
Each of the latter two has

- a **name** and
- a **visibility**.

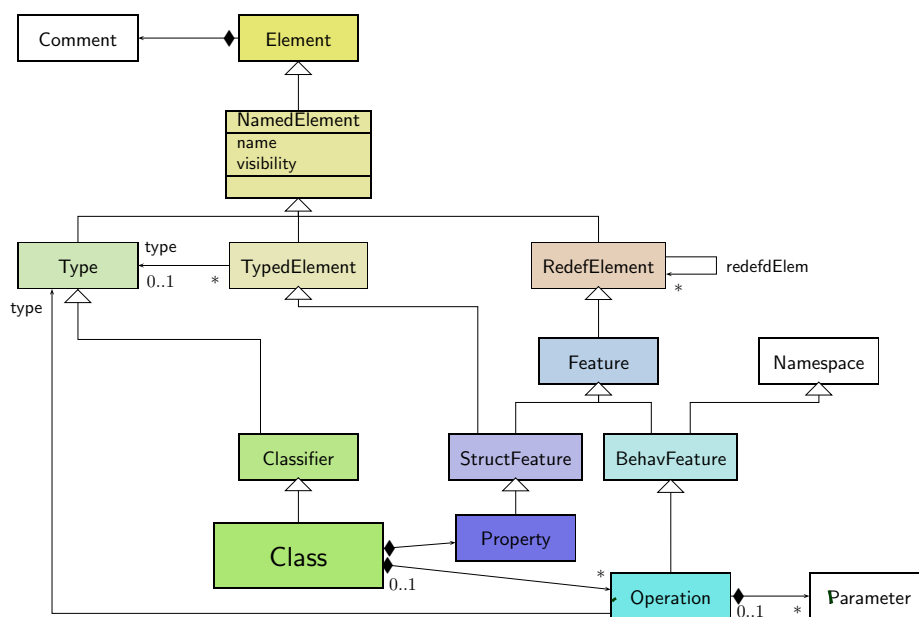
Behavioural features in addition have

- a boolean attribute **isQuery**,
- any number of parameters,
- a return type.

Can we model this (in UML, for a start)?

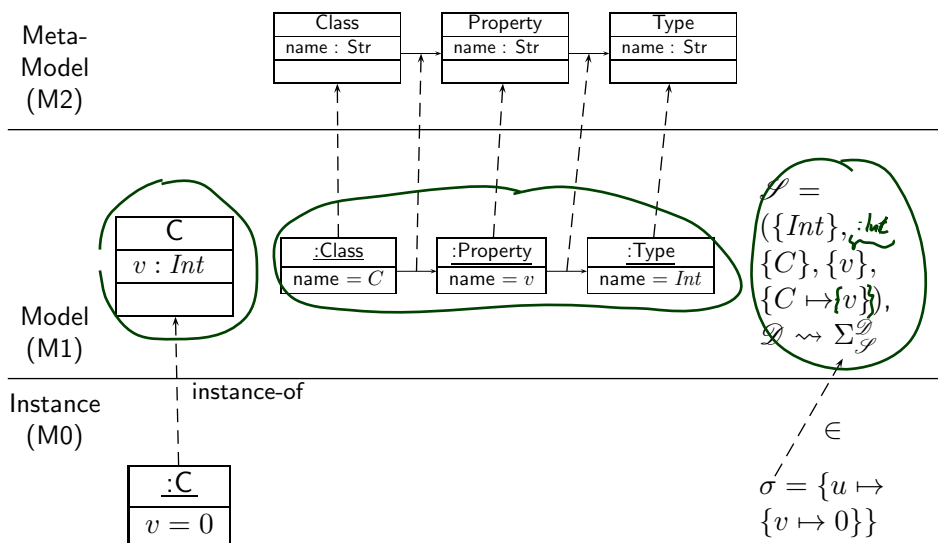


## UML Meta-Model: Extract from UML 2.0 Standard

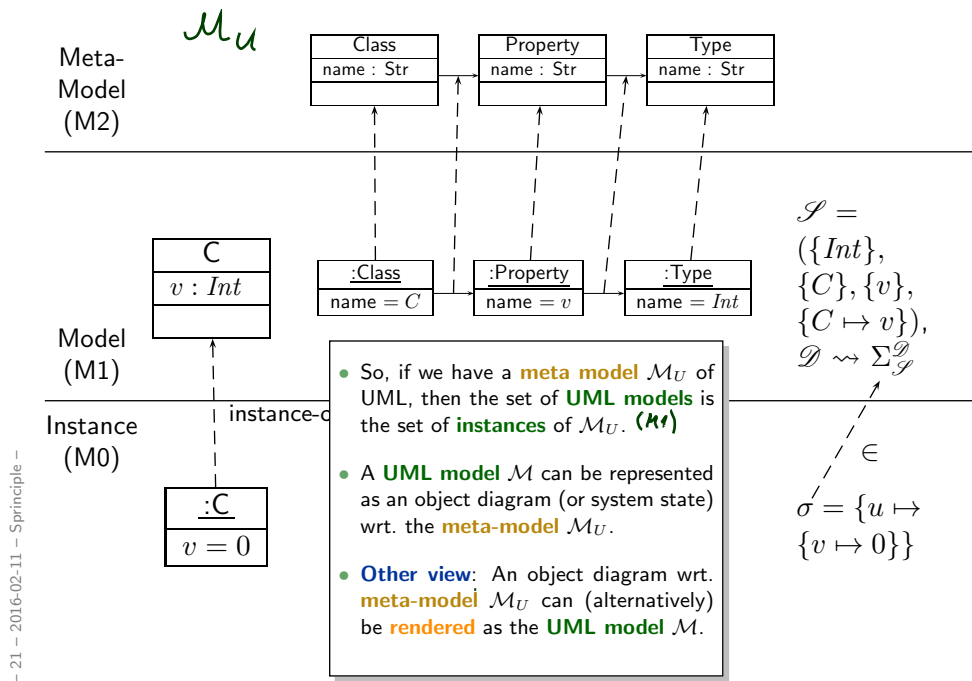


## Meta-Modelling: Principle

### Modelling vs. Meta-Modelling



## Modelling vs. Meta-Modelling



8/31

## 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.

not self . allParents() -> includes(self)" (OMG, 2007b, 53)

- The other way round:

Given a **UML model**  $\mathcal{M}$ , unfold it into an object diagram  $O_1$  wrt.  $\mathcal{M}_U$ .

If  $O_1$  is a **valid** object diagram of  $\mathcal{M}_U$  (i.e. satisfies all invariants from  $Inv(\mathcal{M}_U)$ ), then  $\mathcal{M}$  is a well-formed UML model.

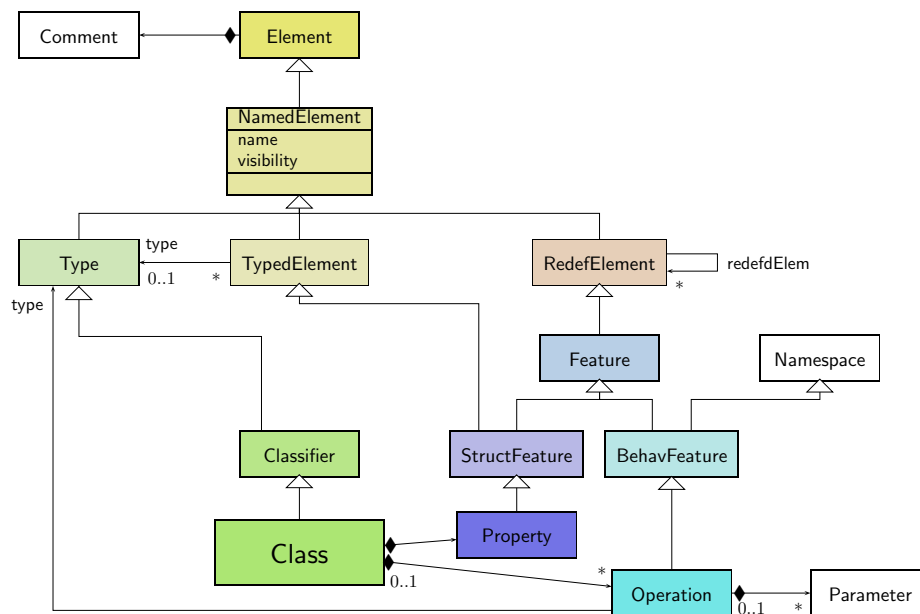
That is, if we have an object diagram **validity checker** for of the meta-modelling language, then we have a **well-formedness checker** for UML models.

# The UML 2.x Standard Revisited

- 21 - 2016-02-11 - main -

10/31

## Claim: Extract from UML 2.0 Standard



- 21 - 2016-02-11 - SumInm -

11/31

## Classes (OMG, 2007b, 32)

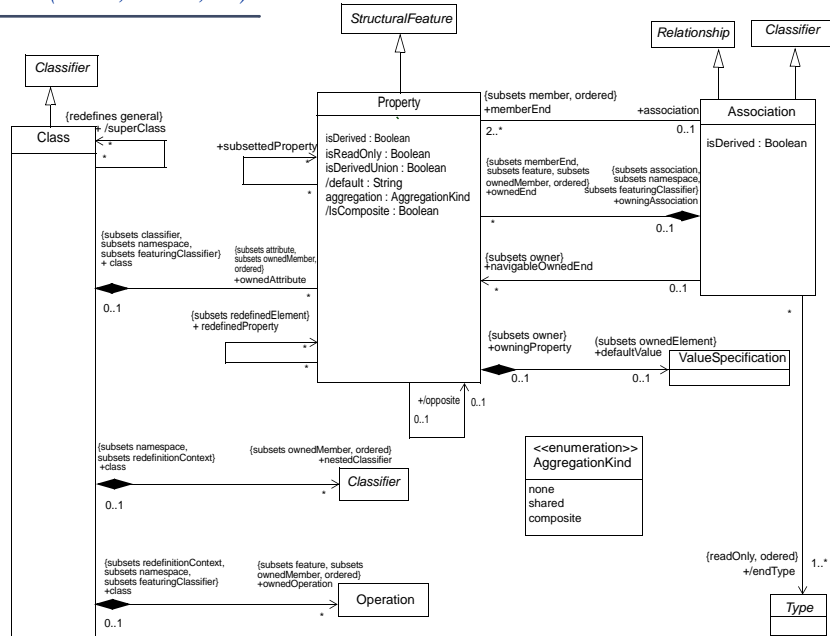


Figure 7.12 - Classes diagram of the Kernel package

- 21 - 2016-02-11 - Sumlmm -

## Operations (OMG, 2007b, 31)

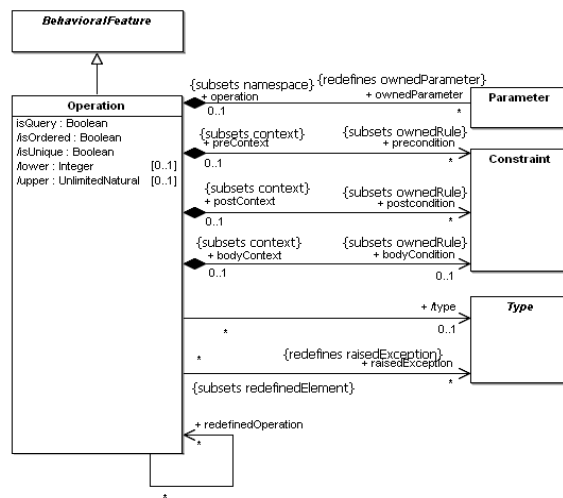


Figure 7.11 - Operations diagram of the Kernel package

- 21 - 2016-02-11 - Sumlmm -

## Operations (OMG, 2007b, 30)

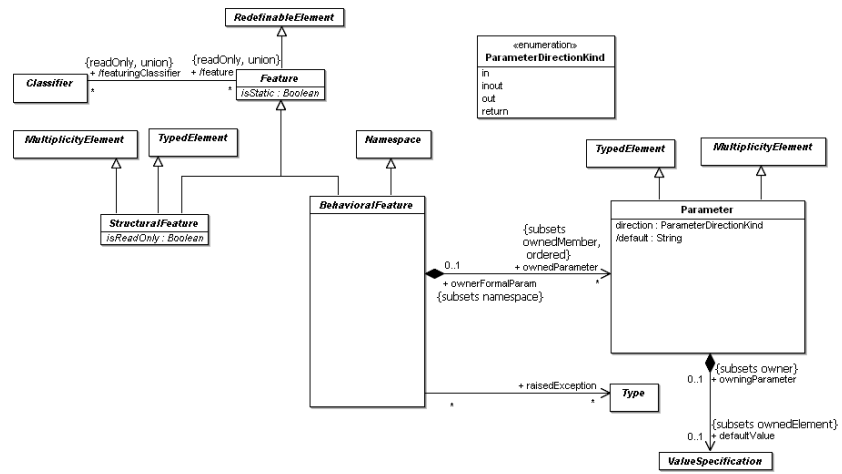


Figure 7.10 - Features diagram of the Kernel package

- 21 - 2016-02-11 - Sumlmm -

## Classifiers (OMG, 2007b, 29)

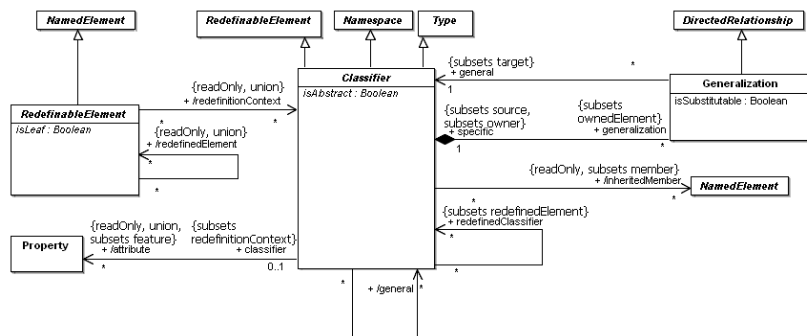


Figure 7.9 - Classifiers diagram of the Kernel package

- 21 - 2016-02-11 - Sumlmm -



## Namespaces (OMG, 2007b, 26)

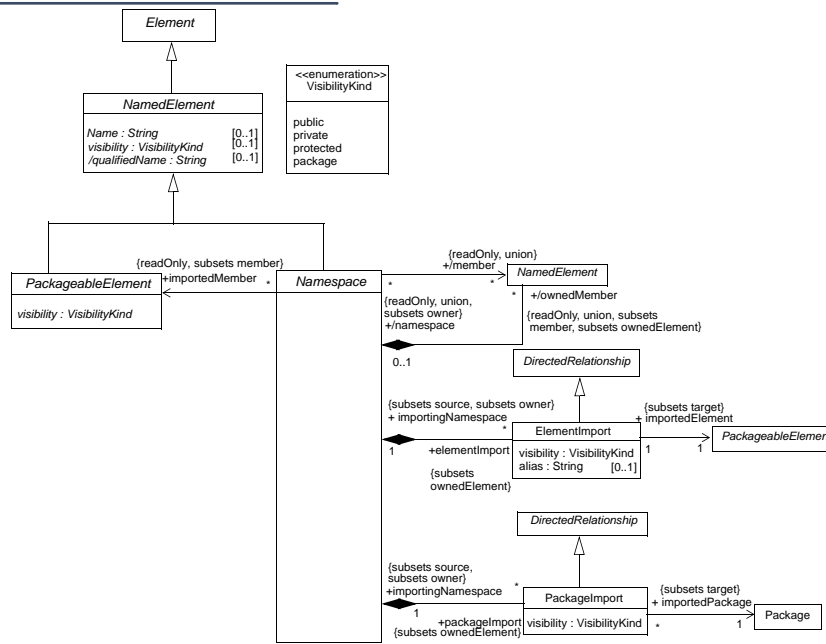


Figure 7.4 - Namespaces diagram of the Kernel package

- 21 - 2016-02-11 - Sumlmm -

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

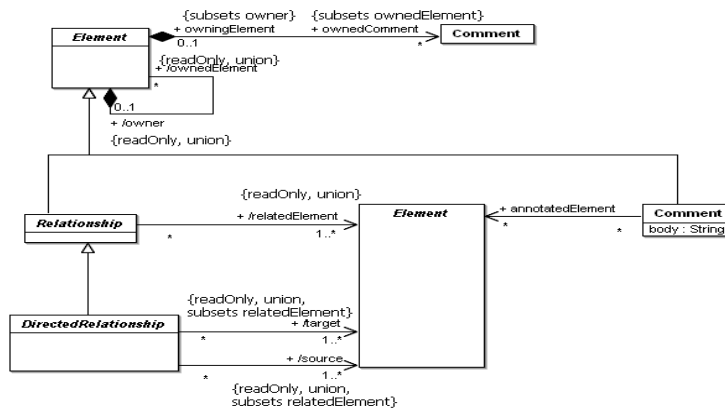


Figure 7.3 - Root diagram of the Kernel package

- 21 - 2016-02-11 - Sumlmm -

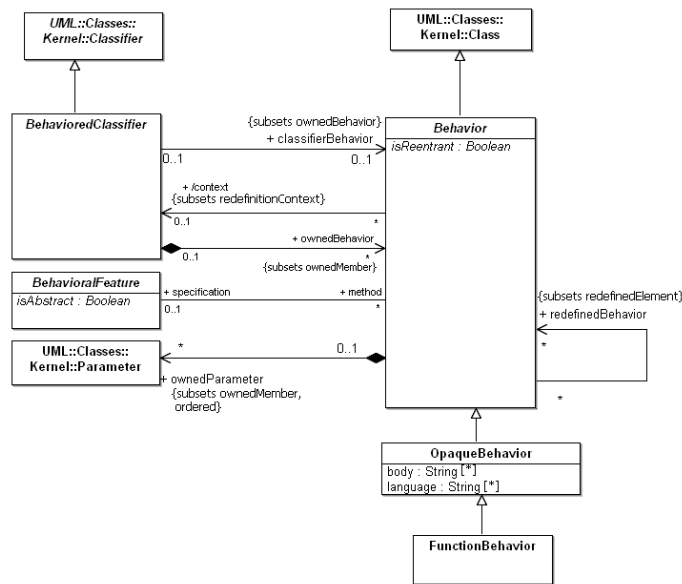


Figure 13.6 - Common Behavior

- 21 - 2016-02-11 - Summm -

### UML Architecture (OMG, 2003, 8)

- 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.

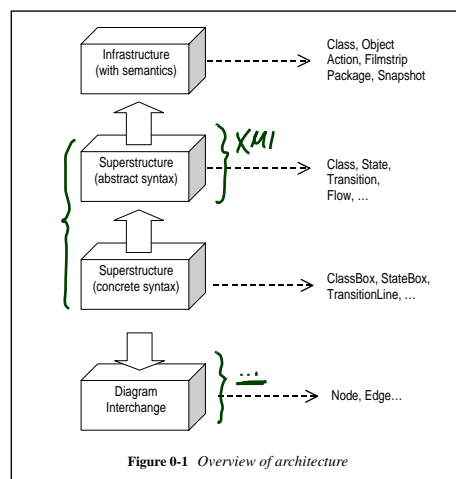
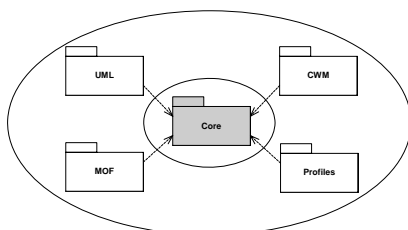


Figure 0-1 Overview of architecture

- 21 - 2016-02-11 - Swhole -

# UML Superstructure Packages (OMG, 2007a, 15)

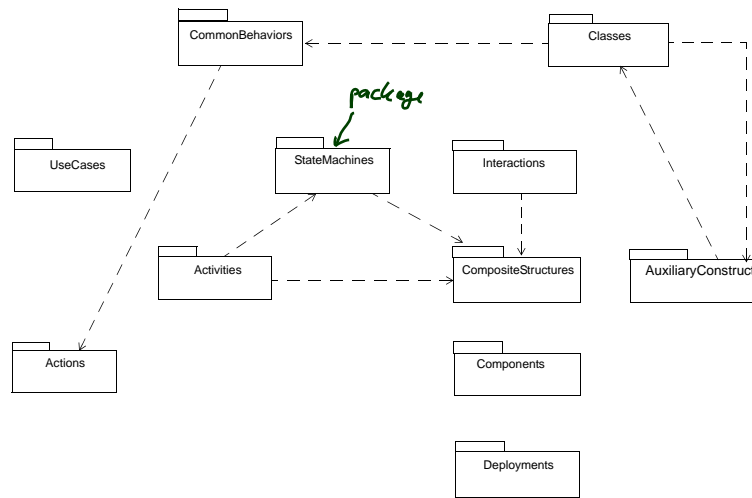


Figure 7.5 - The top-level package structure of the UML 2.1.1 Superstructure

– 21 – 2016-02-11 – Swhole –

## Reading the Standard

Table of Contents	
<b>1. Scope</b>	<b>1</b>
<b>2. Conformance</b>	<b>1</b>
2.1 Language Units	2
2.2 Compliance Levels	2
2.3 Meaning and Types of Compliance	6
2.4 Compliance Level Contents	8
<b>3. Normative References</b>	<b>10</b>
<b>4. Terms and Definitions</b>	<b>10</b>
<b>5. Symbols</b>	<b>10</b>
<b>6. Additional Information</b>	<b>10</b>
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
<b>Part I - Structure</b>	<b>21</b>
<b>7. Classes</b>	<b>23</b>

– 21 – 2016-02-11 – Sreading –

# Reading the Standard

<b>Table of Contents</b>	
<b>1. Scope</b>	7.1 Overview ..... 23
<b>2. Conformance</b>	7.2 Abstract Syntax ..... 24
2.1 Language Units	7.3 Class Descriptions ..... 38
2.2 Compliance Levels	7.3.1 Abstraction (from Dependencies) ..... 38
2.3 Meaning and Types	7.3.2 AggregationKind (from Kernel) ..... 38
2.4 Compliance Level C	7.3.3 Association (from Kernel) ..... 39
<b>3. Normative References</b>	7.3.4 AssociationClass (from AssociationClasses) ..... 47
<b>4. Terms and Definitions</b>	7.3.5 BehavioralFeature (from Kernel) ..... 48
<b>5. Symbols</b>	7.3.6 BehavedClassifier (from Interfaces) ..... 49
<b>6. Additional Information</b>	7.3.7 Class (from Kernel) ..... 49
6.1 Changes to Adopted	7.3.8 Classifier (from Kernel, Dependencies, PowerTypes) ..... 52
6.2 Architectural Alignme	7.3.9 Comment (from Kernel) ..... 57
6.3 On the Run-Time Se	7.3.10 Constraint (from Kernel) ..... 58
6.3.1 The Basic Premis	7.3.11 DataType (from Kernel) ..... 60
6.3.2 The Semantics A	7.3.12 Dependency (from Dependencies) ..... 62
6.3.3 The Basic Causal	7.3.13 DirectedRelationship (from Kernel) ..... 63
6.3.4 Semantics Descr	7.3.14 Element (from Kernel) ..... 64
6.4 The UML Metamodel	7.3.15 ElementImport (from Kernel) ..... 65
6.4.1 Models and What	7.3.16 Enumeration (from Kernel) ..... 67
6.4.2 Semantic Levels	7.3.17 EnumerationLiteral (from Kernel) ..... 68
6.5 How to Read this Sp	7.3.18 Expression (from Kernel) ..... 69
6.5.1 Specification form	7.3.19 Feature (from Kernel) ..... 70
6.5.2 Diagram format	7.3.20 Generalization (from Kernel, PowerTypes) ..... 71
6.6 Acknowledgements	7.3.21 GeneralizationSet (from PowerTypes) ..... 75
<b>Part I - Structure</b>	7.3.22 InstanceSpecification (from Kernel) ..... 82
<b>7. Classes</b>	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
	7.3.47 Relationship (from Kernel) ..... 132
	7.3.48 Slot (from Kernel) ..... 132
	7.3.49 StructuralFeature (from Kernel) ..... 133
	7.3.50 Substitution (from Dependencies) ..... 134
	7.3.51 Type (from Kernel) ..... 135
	7.3.52 TypedElement (from Kernel) ..... 136
	7.3.53 Usage (from Dependencies) ..... 137
	7.3.54 ValueSpecification (from Kernel) ..... 137
	7.3.55 VisibilityKind (from Kernel) ..... 139
	<b>7.4 Diagrams</b> ..... 140
	<b>8. Components</b> ..... 143
	8.1 Overview ..... 143
	8.2 Abstract syntax ..... 144
	8.3 Class Descriptions ..... 146
	8.3.1 Component (from BasicComponents, PackagingComponents) ..... 146
	8.3.2 Connector (from BasicComponents) ..... 154
	8.3.3 ConnectorKind (from BasicComponents) ..... 157
	8.3.4 ComponentRealization (from BasicComponents) ..... 157
	8.4 Diagrams ..... 159
	<b>9. Composite Structures</b> ..... 161
	9.1 Overview ..... 161
	9.2 Abstract syntax ..... 161
	9.3 Class Descriptions ..... 166
	9.3.1 Class (from StructuredClasses) ..... 166
	9.3.2 Classifier (from Collaborations) ..... 167
	9.3.3 Collaboration (from Collaborations) ..... 168
	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) ..... 179
	9.3.10 Parameter (from Collaborations) ..... 179
	9.3.11 Port (from Ports) ..... 179
	9.3.12 Property (from InternalStructures) ..... 183
	9.3.13 StructureClassifier (from InternalStructures) ..... 186
	9.3.14 Trigger (from InvocationActions) ..... 190
	9.3.15 Variable (from StructuredActivities) ..... 191
	9.4 Diagrams ..... 191
	<b>10. Deployments</b> ..... 193

# Reading the Standard

<b>Table of Contents</b>	
<b>1. Scope</b>	7.1 Overview ..... 23
<b>2. Conformance</b>	7.2 Abstract Syntax ..... 24
2.1 Language Units	7.3 Class Descriptions ..... 38
2.2 Compliance Levels	7.3.1 Abstraction (from Dependencies) ..... 38
2.3 Meaning and Types	7.3.2 AggregationKind (from Kernel) ..... 38
2.4 Compliance Level C	7.3.3 Association (from Kernel) ..... 39
<b>3. Normative References</b>	7.3.4 AssociationClass (from AssociationClasses) ..... 47
<b>4. Terms and Definitions</b>	7.3.5 BehavioralFeature (from Kernel) ..... 48
<b>5. Symbols</b>	7.3.6 BehavedClass (from Interfaces) ..... 49
<b>6. Additional Information</b>	7.3.7 Class (from Kernel) ..... 49
6.1 Changes to Adopted	7.3.8 Classifier (from K) ..... 52
6.2 Architectural Alignme	7.3.9 Comment (from K) ..... 57
6.3 On the Run-Time Se	7.3.10 Constraint (from Kernel) ..... 58
6.3.1 The Basic Premis	7.3.11 DataType (from Kernel) ..... 60
6.3.2 The Semantics A	7.3.12 Dependency (fro
6.3.3 The Basic Causal	7.3.13 DirectedRelat
6.3.4 Semantics Descr	7.3.14 Element (from K
6.4 The UML Metamodel	7.3.15 ElementImport (f
6.4.1 Models and What	7.3.16 Enumeration (fr
6.4.2 Semantic Levels	7.3.17 EnumerationLit
6.5 How to Read this Sp	7.3.18 Expression (fr
6.5.1 Specification form	7.3.19 Feature (from K
6.5.2 Diagram format	7.3.20 Generalization (f
6.6 Acknowledgements	7.3.21 GeneralizationS
<b>Part I - Structure</b>	7.3.22 InstanceSpecifi
<b>7. Classes</b>	7.3.23 InstanceValue (f
	7.3.24 Interface (from I
	7.3.25 InterfaceRealiz
	7.3.26 LiteralBoolean (fr
	7.3.27 LiteralInteger (fr
	7.3.28 LiteralNull (from
	7.3.29 LiteralSpecificat
	7.3.30 LiteralString (fr
	7.3.31 LiteralUnlimitedE
	7.3.32 MultiplicityElem
	7.3.33 NamedElement
	7.3.34 Namespace (from
	7.3.35 OpaqueExpress
	7.3.36 Operation (from
	7.3.37 Package (from K
	7.3.38 PackageableEie
	7.3.39 PackageImport (
	7.3.40 PackageMerge (
	7.3.41 Parameter (from
	7.3.42 ParameterDirec
	7.3.43 PrimitiveType (f
	7.3.44 Property (from K
	7.3.45 Realization (from
	7.3.46 RedefinableEie
	7.3.47 Relationship (from Kernel) ..... 132
	7.3.48 Slot (from Kernel) ..... 132
	7.3.49 StructuralFeature (from Kernel) ..... 133
	7.3.50 Substitution (from Dependencies) ..... 134
	7.3.51 Type (from Kernel) ..... 135
	7.3.52 TypedElement (from Kernel) ..... 136
	7.3.53 Usage (from Dependencies) ..... 137
	7.3.54 ValueSpecification (from Kernel) ..... 137
	7.3.55 VisibilityKind (from Kernel) ..... 139
	<b>7.4 Diagrams</b> ..... 140
	<b>8. Components</b> ..... 143
	8.1 Overview ..... 143
	8.2 Abstract syntax ..... 144
	8.3 Class Descriptions ..... 146
	8.3.1 Component (from BasicComponents, PackagingComponents) ..... 146
	8.3.2 Connector (from BasicComponents) ..... 154
	8.3.3 ConnectorKind (from BasicComponents) ..... 157
	8.3.4 ComponentRealization (from BasicComponents) ..... 157
	8.4 Diagrams ..... 159
	<b>9. Composite Structures</b> ..... 161
	9.1 Overview ..... 161
	9.2 Abstract syntax ..... 161
	9.3 Class Descriptions ..... 166
	9.3.1 Class (from StructuredClasses) ..... 166
	9.3.2 Classifier (from Collaborations) ..... 167
	9.3.3 Collaboration (from Collaborations) ..... 168
	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) ..... 179
	9.3.10 Parameter (from Collaborations) ..... 179
	9.3.11 Port (from Ports) ..... 179
	9.3.12 Property (from InternalStructures) ..... 183
	9.3.13 StructureClassifier (from InternalStructures) ..... 186
	9.3.14 Trigger (from InvocationActions) ..... 190
	9.3.15 Variable (from StructuredActivities) ..... 191
	9.4 Diagrams ..... 191
	<b>10. Deployments</b> ..... 193

# Reading the Standard Cont'd

- 21 - 2016-02-11 - Reading -

Window
<pre> public size: Area = (100, 100) defaultSize: Rectangle protected visibility: Boolean = true private xWin: XWindow  public display() hide() private attach(XWin: XWindow)                     </pre>

**Figure 7.29 - Class notation: attributes and operations grouped according to visibility**

**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
- "RededefinableElement (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 metarelations 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.

52 UML Superstructure Specification, v2.1.2

# Reading the Standard Cont'd

- 21 - 2016-02-11 - Reading -

Window
<pre> public size: Area = (100, 100) defaultSize: Rectangle protected visibility: Boolean = true private xWin: XWindow  public display() hide() private attach(XWin: XWindow)                     </pre>

**Figure 7.29 - Class notation: attributes and operations grouped according to visibility**

**7.3.8 Classifier**

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

**Generalization**

- generalization: Generalization[\*]  
Specifies the Generalization relationships for this Classifier. These Generalizations navigate to more general classifiers in the generalization hierarchy. Subsets *Element:ownedElement*
- /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.  
general = 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.  
not self.allParents()->includes(self)
- [3] A classifier may only specialize classifiers of a valid type.  
self.parents()->forall(c | self.maySpecializeType(c))
- [4] 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)))

**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.  
Classifier::allFeatures(): Set(Feature);  
allFeatures = member->select(ocl::kindOf(Feature))
- [2] The query parents() gives all of the immediate ancestors of a generalized Classifier.  
Classifier::parents(): Set(Classifier);  
parents = generalization.general

52 UML Superstructure Specification, v2.1.2

# Reading the Standard Cont'd

52 UML Superstructure Specification, v2.1.2

Figure 7.29 - Class

**7.3.8 Class**

A classifier is a

**Generalization**

- "Name" : S
- "Redefin" : S
- "Type" : (f)

**Description**

A classifier is a

A classifier is a

A classifier is a

**Attributes**

- isAbstract : Boolean
- If true, classifier relation

**Associations**

- / attribute : P Refers Classifier
- / feature : F Specific parents = g
- / general : C Specific

Package PowerTypes

- generalization
- substitution
- redefinedClass

Package Dependencies

- substitution
- substitution

**Constraints**

(1) The generalization

(2) Generalization transitively

(3) A classifier

(4) The inheritance

(5) The Classifier

(6) The query in the specification

(7) The query to be redefined

(8) The query to the specified

**Semantics**

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 classifier 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.

- 21 - 2016-02-11 - Reading -

# Reading the Standard Cont'd

52 UML Superstructure Specification, v2.1.2

Figure 7.29 - Class

**7.3.8 Class**

A classifier is a

**Generalization**

- "Name" : S
- "Redefin" : S
- "Type" : (f)

**Description**

A classifier is a

A classifier is a

A classifier is a

**Attributes**

- isAbstract : Boolean
- If true, classifier relation

**Associations**

- / attribute : P Refers Classifier
- / feature : F Specific parents = g
- / general : C Specific

Package PowerTypes

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) have a common specific Classifier, and b) 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.

The name of an abstract Classifier is shown in italics.

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, AssociationClasses)" on page 123.

**Presentation Options**

Any compartment may be suppressed. A separator line is not drawn for a suppressed compartment. If a compartment is suppressed, no inference can be drawn about the presence or absence of elements in it. Compartment names can be used to remove ambiguity, if necessary.

An abstract Classifier can be shown using the keyword [abstract] after or below the name of the Classifier.

The type, visibility, default, multiplicity, property string may be suppressed from being displayed, even if there are values in the model.

The individual properties of an attribute can be shown in columns rather than as a continuous string.

**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 of each word 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.

- 21 - 2016-02-11 - Reading -

**Package PowerTypes**

generalizationSet

public size: Area = (...)

hide()

private attach(XWindow)

public display()

private attach(XWindow)

Figure 7.29 - Class

**7.3.8 Class**

A classifier is a

**Generalization**

- "NameSet"
- "Redefined"
- "Type (if not self.allParents)"

**Description**

A classifier is a

A classifier is a

A classifier is a

**Attributes**

- isAbstract: If true, classifier relation

**Associations**

- attribute: P Refers to Class
- feature: F Specifies
- general: C Specifies

**Package PowerTypes**

The notion of p subsets. In esset a Classifier with for that class.

**Semantic Vari**

The precise life

**Notation**

Classifier is an in one place a default notation compartments 4 classifier can be

The name of an

An attribute can (from Kernel, A

**Presentation O**

Any compartme suppressed, no i to remove ambi

An abstract Cla

The type, visibi in the model.

The individual

**Style Guidelin**

- Attribute and using
- Center th
- Center ke
- For those with an u
- Left justi
- Begin att
- Show full

**ClassA**

```

name: String
shape: Rectangle
+ size: Integer [0..1]
/ area: Integer (readOnly)
height: Integer=5
width: Integer
    
```

**ClassB**

```

id (redefines name)
shape: Square
height = 7
width
    
```

**Figure 7.30 - Examples of attributes**

The attributes in Figure 7.30 are explained below.

- ClassA: name is an attribute with type String.
- ClassA: shape is an attribute with type Rectangle.
- ClassA: size is a public attribute of type Integer with multiplicity 0..1.
- ClassA: area is a derived attribute with type Integer. It is marked as read-only.
- ClassA: height is an attribute of type Integer with a default initial value of 5.
- ClassA: width is an attribute of type Integer.
- ClassB: id is an attribute that redefines ClassA: name.
- ClassB: shape is an attribute that redefines ClassA: shape. It has type Square, a specialization of Rectangle.
- ClassB: height is an attribute that redefines ClassA: height. It has a default of 7 for ClassB instances that overrides the ClassA default of 5.
- ClassB: width is a derived attribute that redefines ClassA: width, which is not derived.

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

**Figure 7.31 - Association-like notation for attribute**

**Package PowerTypes**

generalizationSet

public size: Area = (...)

hide()

private attach(XWindow)

public display()

private attach(XWindow)

Figure 7.29 - Class

**7.3.8 Class**

A classifier is a

**Generalization**

- "NameSet"
- "Redefined"
- "Type (if not self.allParents)"

**Description**

A classifier is a

A classifier is a

A classifier is a

**Attributes**

- isAbstract: If true, classifier relation

**Associations**

- attribute: P Refers to Class
- feature: F Specifies
- general: C Specifies

**Package PowerTypes**

The notion of p subsets. In esset a Classifier with for that class.

**Semantic Vari**

The precise life

**Notation**

Classifier is an in one place a default notation compartments 4 classifier can be

The name of an

An attribute can (from Kernel, A

**Presentation O**

Any compartme suppressed, no i to remove ambi

An abstract Cla

The type, visibi in the model.

The individual

**Style Guidelin**

- Attribute and using
- Center th
- Center ke
- For those with an u
- Left justi
- Begin att
- Show full

**Class**

```

name: String
shape: Rectangle
+ size: Integer [0..1]
/ area: Integer
height: Integer=5
width: Integer
    
```

**Class**

```

id (redefines name)
shape: Square
height = 7
width
    
```

**Figure 7.30 - Ex**

The attributes in

- ClassA: name
- ClassA: shape
- ClassA: size
- ClassA: area
- ClassA: height
- ClassA: width
- ClassB: id
- ClassB: shape
- ClassB: height
- ClassB: width

**Figure 7.31 - As**

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.

## Meta Object Facility (MOF)

### Open Questions...

- Now you've been **"tricked"**.
  - We didn't tell what the **modelling language** for meta-modelling is.
  - We didn't tell what the **is-instance-of** relation of this language 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 ?**



## MOF Semantics

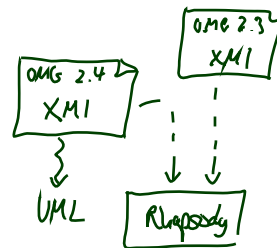
- One approach:
  - Treat it with **our signature-based theory**
  - This is (in effect) the right direction, but may require new (or extended) signatures for each level.
- Other approach:
  - Define a **generic, graph based** “is-instance-of” relation.
  - Object diagrams (that **are** graphs) then **are** the system states — not **only graphical representations** of system states.
  - If this works out, good: We can easily experiment with different language designs, e.g. different flavours of UML that immediately have a semantics.
  - Most interesting: also do generic definition of behaviour within a closed modelling setting, but this is clearly still research, e.g. [Buschermöhle and Oelerink \(2008\)](#).

– 21 – 2016-02-11 – Smof –

25/31

## Benefits

- In particular:
  - Benefits for **Modelling Tools**.
  - Benefits for **Language Design**.
  - Benefits for **Code Generation and MDA**.

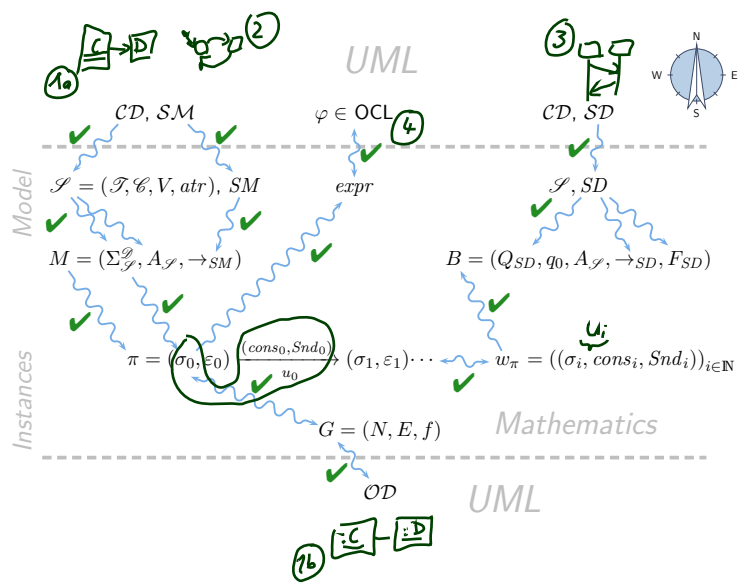


– 21 – 2016-02-11 – Sbenefits –

26/31

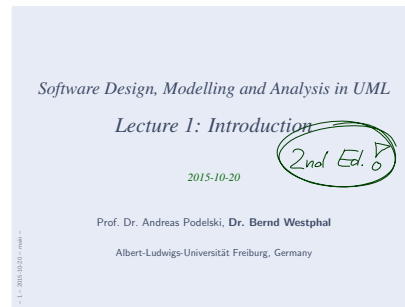
And That's It!

The Map



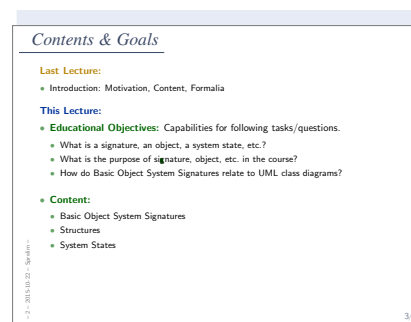
## Content

- **Lecture 1:** Introduction



## Content

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



## Content

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

*Contents & Goals*  
**Contents & Goals**

**Last Lecture:**

- Basic Object System Signature  $\mathcal{S}$  and Structure  $\mathcal{S}$ , 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  $\mathcal{S}(C)$  and  $T_C$  related?
- **Content:**
  - OCL Syntax
  - OCL Semantics (over system states)

– 03 – 2016-02-09 – System –

2/38

## Content

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

*Contents & Goals*  
**Contents & Goals**

**Last Lecture:**

- OCL Syntax

**This Lecture:**

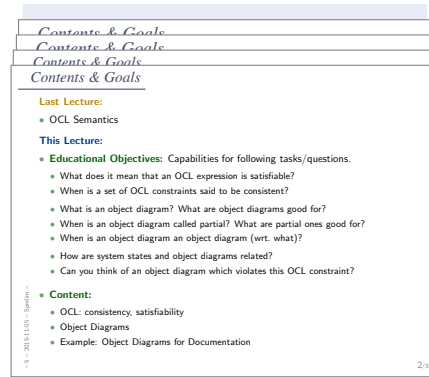
- **Educational Objectives:** Capabilities for these tasks/questions:
  - Please un-abbreviate all abbreviations in this OCL expression. ✓
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this system state?
  - Give a system state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

– 4 – 2016-11-03 – System –

2/38

## Content

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



Content & Goals  
Content & Goals  
Content & Goals  
Content & Goals  
Content & Goals

**Last Lecture:**

- OCL Semantics

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does it mean that an OCL expression is satisfiable?
  - When is a set of OCL constraints said to be consistent?
  - What is an object diagram? What are object diagrams good for?
  - When is an object diagram called partial? What are partial ones good for?
  - When is an object diagram an object diagram (wrt. what)?
  - How are system states and object diagrams related?
  - Can you think of an object diagram which violates this OCL constraint?

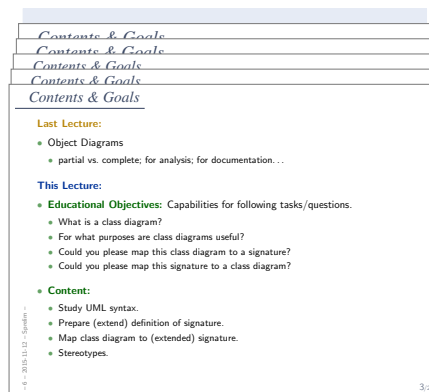
**Content:**

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

5. - 2016-02-11 - Spring - 2/31

## 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 & Goals  
Content & Goals  
Content & Goals  
Content & Goals  
Content & Goals

**Last Lecture:**

- Object Diagrams
  - partial vs. complete; for analysis; for documentation...

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a class diagram?
  - For what purposes are class diagrams useful?
  - Could you please map this class diagram to a signature?
  - Could you please map this signature to a class diagram?

**Content:**

- Study UML syntax.
- Prepare (extend) definition of signature.
- Map class diagram to (extended) signature.
- Stereotypes.

6. - 2016-02-11 - Spring - 3/31

## 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

*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*

**Last Lecture:**

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

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Could you please map this class diagram to a signature?
  - What if things are missing?
  - Could you please map this signature to a class diagram?
  - What is the semantics of 'abstract'?
  - What is visibility good for?
- **Content:**
  - Map class diagram to (extended) signature cont'd.
  - Stereotypes – for documentation.
  - Visibility as an extension of well-typedness.

© 2016, D. S. Schmidt

2/31

## 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

*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*

**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?

© 2016, D. S. Schmidt

2/34

## 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

*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*

**Last Lecture:**

- Associations syntax and semantics.
- Associations in OCL syntax.

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Compute the value of a given OCL constraint in a system state with links.
  - How did we treat "multiplicity" semantically?
  - What does "navigability", "ownership", ... mean?
  - ...
- **Content:**
  - Associations and OCL: semantics.
  - Associations: the rest.

– 9 – 2016-02-11 – Semantics –

2/49

## 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

*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*  
*Contents & Goals*

**Last Lecture:**

- (Mostly) completed discussion of modelling **structure**.

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the purpose of a behavioural model?
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
- **Content:**
  - For completeness: Modelling Guidelines for Class Diagrams
  - Purposes of Behavioural Models
  - UML Core State Machines

– 10 – 2016-02-11 – Semantics –

2/31

## 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

Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals

**Last Lecture:**

- What makes a class diagram a good class diagram?
- Core State Machine syntax

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: Signal, Event, Ether, Transformer, Step, RTC.
- **Content:**
  - UML standard: basic causality model
  - Ether
  - Transformers
  - Step, Run-to-Completion Step

– 11 – 2016-02-11 – Sem4 – 2/34

## 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

Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals  
 Contents & Goals

**Last Lecture:**

- Basic causality model
- Ether/event pool
- System configuration

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: Signal, Event, Ether, Transformer, Step, RTC.
- **Content:**
  - System configuration cont'd
  - Transformers
  - Step, Run-to-Completion Step

– 12 – 2016-02-11 – Sem4 – 2/47



## 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

Thumbnail of slide 29. The slide title is "Content". It lists lectures 1 through 13. Below the list, it shows a stack of "Contents & Goals" slides. The main content area includes:

- Last Lecture:**
  - System configuration cont'd
  - Action language and transformer
- This Lecture:**
  - **Educational Objectives:** Capabilities for following tasks/questions.
    - What does this State Machine mean? What happens if I inject this event?
    - Can you please model the following behaviour.
    - What is: Signal, Event, Ether, Transformer, Step, RTC.
  - **Content:**
    - Step, Run-to-Completion Step

Small text at the bottom left: - 13 - 2016-02-11 - System -

Page number: 29

## 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:** Core State Machines IV

Thumbnail of slide 30. The slide title is "Content". It lists lectures 1 through 14. Below the list, it shows a stack of "Contents & Goals" slides. The main content area includes:

- Last Lecture:**
  - Transitions by Rule (i) to (v).
- This Lecture:**
  - **Educational Objectives:** Capabilities for following tasks/questions.
    - What is a step / run-to-completion step?
    - What is divergence in the context of UML models?
    - How to define what happens at "system / model startup"?
    - What are roles of OCL constraints in behavioural models?
    - Is this UML model consistent with that OCL constraint?
    - What do the actions create / destroy do? What are the options and our choices (why)?
  - **Content:**
    - Step / RTC-Step revisited, Divergence
    - Initial states
    - Missing pieces: create / destroy transformer
    - A closer look onto code generation
    - Maybe: hierarchical state machines

Small text at the bottom left: - 14 - 2016-02-11 - System -

Page number: 30

## 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:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I

Thumbnail of slide 29/31. The slide title is "Contents & Goals". It lists "Last Lecture" topics: step, RTC-step, divergence; initial state, UML model semantics (so far); create, destroy actions. It lists "This Lecture" topics: 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.

## 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:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I
- **Lecture 16:** Hierarchical State Machines II

Thumbnail of slide 29/31. The slide title is "Contents & Goals". It lists "Last Lecture" topics: Legal state configurations; Legal transitions; Rules (i) to (v) for hierarchical state machines. It lists "This Lecture" topics: Educational Objectives (Capabilities for following tasks/questions): How do entry / exit actions work? What about do-actions?; What is the effect of shallow / deep history pseudo-states?; What about junction, choice, terminate, etc.?; What is the idea of deferred events?; How are passive reactive objects treated in Rhapsody's UML semantics?; What about methods?; Content: Entry / exit / do actions, internal transitions; Remaining pseudo-states; deferred events; Passive reactive objects; Behavioural features.

## 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:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I
- **Lecture 16:** Hierarchical State Machines II
- **Lecture 17:** Live Sequence Charts I

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

**Last Lecture:**

- Hierarchical state machines: the rest
- Deferred events
- Passive reactive objects

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What are constructive and reflective descriptions of behaviour?
  - What are UML Interactions?
  - What is the abstract syntax of this LSC?
  - How is the semantics of LSCs constructed?
  - What is a cut, fired-set, etc.?
- **Content:**
  - Rhapsody code generation
  - Interactions: Live Sequence Charts
  - LSC syntax
  - Towards semantics

– 17 – 2016-02-11 – Spring –

2/45

## 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:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I
- **Lecture 16:** Hierarchical State Machines II
- **Lecture 17:** Live Sequence Charts I
- **Lecture 18:** Live Sequence Charts II

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

**Last Lecture:**

- Rhapsody code generation
- Interactions: Live Sequence Charts
- LSC syntax

**This Lecture:**

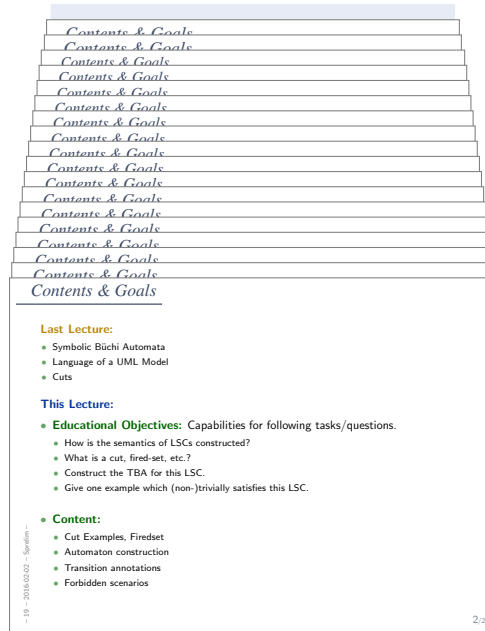
- **Educational Objectives:** Capabilities for following tasks/questions.
  - How is the semantics of LSCs constructed?
  - What is a cut, fired-set, etc.?
  - Construct the TBA for this LSC.
  - Give one example which (non-)trivially satisfies this LSC.
- **Content:**
  - Symbolic Automata
  - Firedset, Cut
  - Automaton construction
  - Transition annotations

– 18 – 2016-02-11 – Spring –

2/45

## 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:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I
- **Lecture 16:** Hierarchical State Machines II
- **Lecture 17:** Live Sequence Charts I
- **Lecture 18:** Live Sequence Charts II
- **Lecture 19:** Live Sequence Charts III



*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

**Last Lecture:**

- Symbolic Buchi Automata
- Language of a UML Model
- Cuts

**This Lecture:**

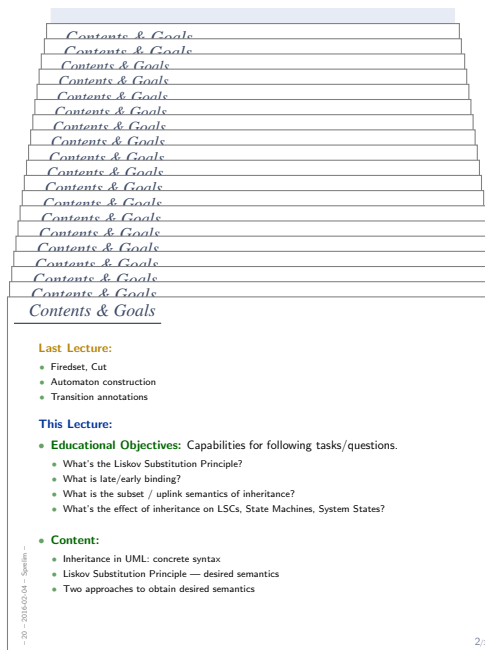
- **Educational Objectives:** Capabilities for following tasks/questions.
  - How is the semantics of LSCs constructed?
  - What is a cut, fired-set, etc.?
  - Construct the TBA for this LSC.
  - Give one example which (non-)trivially satisfies this LSC.
- **Content:**
  - Cut Examples, Firedset
  - Automaton construction
  - Transition annotations
  - Forbidden scenarios

- 19 - 2016-02-09 - System -

2/28

## 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:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I
- **Lecture 16:** Hierarchical State Machines II
- **Lecture 17:** Live Sequence Charts I
- **Lecture 18:** Live Sequence Charts II
- **Lecture 19:** Live Sequence Charts III
- **Lecture 20:** Inheritance



*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

**Last Lecture:**

- Firedset, Cut
- Automaton construction
- Transition annotations

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the Liskov Substitution Principle?
  - What is late/early binding?
  - What is the subset / uplink semantics of inheritance?
  - What's the effect of inheritance on LSCs, State Machines, System States?
- **Content:**
  - Inheritance in UML: concrete syntax
  - Liskov Substitution Principle — desired semantics
  - Two approaches to obtain desired semantics

- 20 - 2016-02-09 - System -

2/30

## Content

- **Lecture 1:** Introduction
- **Lecture 2:** Semantical Model
- **Lecture 3:** Object Constraint Language (OCL)
- **Lecture 4:** OCL Semantics (4)
- **Lecture 5:** Object Diagrams (5)
- **Lecture 6:** Class Diagrams I
- **Lecture 7:** Class Diagrams II (6)
- **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 (2)
- **Lecture 14:** Core State Machines IV
- **Lecture 15:** Hierarchical State Machines I
- **Lecture 16:** Hierarchical State Machines II
- **Lecture 17:** Live Sequence Charts I
- **Lecture 18:** Live Sequence Charts II (3)
- **Lecture 19:** Live Sequence Charts III
- **Lecture 20:** Inheritance
- **Lecture 21:** Meta-Modelling

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

*Contents & Goals*

**Last Lecture:**

- Liskov Substitution Principle
- Inheritance: Domain Inclusion Semantics

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is the idea of meta-modelling?
  - How does meta-modelling relate to UML?
- **Content:**
  - The UML Meta Model
  - Wrapup & Questions

- 21 - 2016-02-11 - Spring -

2/31

## References

## *References*

---

Buschermöhle, R. and Oelerink, J. (2008). Rich meta object facility. In *Proc. 1st IEEE Int'l workshop UML and Formal Methods*.

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

OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.