4. Class Diagrams
4.1 Examples for Static Structure Diagrams

- To follow: examples are taken from the UML notation guide
- Static structure diagrams include class diagrams and object diagrams
Static structure diagram (3-20)

Window

- size: Area = (100, 100)
- visibility: Boolean = true
- default-size: Rectangle
- maximum-size: Rectangle
- xptr: XWindow*

Window

- display ()
- hide ()
- create ()
- attachXWindow(xwin:Xwindow*)

{abstract, author=Joe, status=tested}
Static structure diagram (3-21)

«controller»

PenTracker

{ leaf, author=”Mary Jones”}
Static structure diagram (3-22)

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1:Point</td>
</tr>
<tr>
<td>p2:Point</td>
</tr>
</tbody>
</table>

«constructor»
Rectangle(p1:Point, p2:Point)
«query»
area (): Real
aspect (): Real
...
«update»
move (delta: Point)
scale (ratio: Real)
...


### Static structure diagram (3-23)

<table>
<thead>
<tr>
<th>Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>operations</strong></td>
</tr>
<tr>
<td>guarantee()</td>
</tr>
<tr>
<td>cancel ()</td>
</tr>
<tr>
<td>change (newDate: Date)</td>
</tr>
<tr>
<td><strong>responsibilities</strong></td>
</tr>
<tr>
<td>bill no-shows</td>
</tr>
<tr>
<td>match to available rooms</td>
</tr>
<tr>
<td><strong>exceptions</strong></td>
</tr>
<tr>
<td>invalid credit card</td>
</tr>
</tbody>
</table>
Static structure diagram (3-24)

```
PoliceStation
  alert (Alarm)
    1 station
    *

BurglarAlarm
  isTripped: Boolean = false
  report()  --

{ if isTripped
  then station.alert(self) }
```
Static structure diagram (3-38)

triangle: Polygon

center = (0,0)
vertices = ((0,0),(4,0),(4,3))
borderColor = black
fillColor = white

triangle

:Polygon

scheduler
4.1 Examples for Static Structure Diagrams

Static structure diagram (3-39)

```
awindow : Window

horizontalBar : ScrollBar
verticalBar : ScrollBar

surface : Pane

title : TitleBar
```

moves

moves
4.1 Examples for Static Structure Diagrams

Static structure diagram (3-40)
4.1 Examples for Static Structure Diagrams

Static structure diagram (3-41)
Static structure diagram (3-42)

- Bank
  - account #
    * 0..1
    - Person

- Chessboard
  - rank: Rank
  - file: File
    - 1
    - 1
    - Square
Static structure diagram (3-43)
Static structure diagram (3-44)
Static structure diagram (3-45-1)
Static structure diagram (3-45-2)
Static structure diagram (3-46)

downhillSkiClub:Club

president

treasurer

Jill:Person

member

officer

Joe:Person

member

member

Chris:Person

officer
Static structure diagram (3-47)
4.1 Examples for Static Structure Diagrams

Static structure diagram (3-48)
Static structure diagram (3-49)

{disjoint, incomplete} species

Tree

Oak

Elm

Birch
4.2 Advanced Concepts in Class Diagrams

Starting situation

- UML complex language with many modeling features
- no overall agreed semantic foundation
- how to deal with the language element variety?

Study UML class and statechart diagrams and compare language elements

- explain advanced UML features by some more basic UML constructs
- use Object Constraint Logic OCL to express these advanced features
Benefits

• small set of features reduces complexity of design and facilitates communication
• smooth transition from utilizing basic set of language features to more sophisticated ones
• definition in terms of simple features helps identifying certain repeating patterns
Equivalence Rule for Cardinality Notation

\[\begin{align*}
&\text{A} \quad \text{ra} \quad \text{rb} \quad \text{B} \\
&\text{la..ha} \quad \text{lb..hb} \\
\rightarrow \\
&\text{A} \quad \text{ra} \quad \text{rb} \quad \text{B} \\
&\ast \quad \ast \\
&\text{+ constraint}
\end{align*}\]

\[\begin{align*}
\text{A.allInstances->forAll( } & \text{a } \mid \\
&\text{a.rb->size>=lb and a.rb->size<=hb )} \\
\text{and} \\
\text{B.allInstances->forAll( } & \text{b } \mid \\
&\text{b.ra->size>=la and b.ra->size<=ha )}
\end{align*}\]
Example for Cardinality Notation

```
<table>
<thead>
<tr>
<th>Rook</th>
<th>protectingRook</th>
<th>protectedSquare</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..2</td>
<td>2..14</td>
<td></td>
</tr>
</tbody>
</table>
```

Rook->forAll( r | r.protectedSquare->size>=2 and r.protectedSquare->size<=14 )

and

Square->forAll( s | s.protectingRook->size>=0 and s.protectingRook->size<=2 )
Equivalence Rule for Ternary Association Cardinality

\[
\text{A} \rightarrow \text{forall}(a \mid \\
\text{B} \rightarrow \text{forall}(b \mid \\
\text{C} \rightarrow \text{select}(c \mid \text{R}(a,b,c) \rightarrow \text{size} \geq l \text{ and } \text{(*)}) \\
\text{C} \rightarrow \text{select}(c \mid \text{R}(a,b,c) \rightarrow \text{size} \leq h))
\]

(*) Syntax \( \text{R}(a,b,c) \) not supported by OCL
Example for Cardinality Notation in N-ary Association

```
Student->forAll(st | Subject->forAll(su | Teacher->select(t | Examine(st,su,t))->size >= 0 and Teacher->select(t | Examine(st,su,t))->size <= 3 ))
```
Equivalence Rule for Ternary Association

\[ R \rightarrow \text{forAll}( r, r' | (r.ra=r'.ra \text{ and } r.rb=r'.rb \text{ and } r.rc=r'.rc ) \text{ implies } r=r') \]
Example for Equivalence Rule for Ternary Association

Examine->forAll( e, e' |
    ( e.examinee=e'.examinee and e.subject=e'.subject and e.examiner=e'.examiner ) implies e=e' )
Equivalence Rule for Qualifier

\[ A \rightarrow \text{forall}(a \mid a.ac \rightarrow \text{forall}(ac \mid a.rb \rightarrow \text{select}(b \mid b.ac \rightarrow \text{exists}(ac' \mid ac'.q = ac.q \\
\text{and } ac'.ra = a) \rightarrow \text{size } \geq l \\
\text{and } a.rb \rightarrow \text{select}(b \mid b.ac \rightarrow \text{exists}(ac' \mid ac'.q = ac.q \\
\text{and } ac'.ra = a) \rightarrow \text{size } \leq h) \) \]
Example for Qualifier

Bank->forall(b |
  b.account->forall(a |
    b.person->select(p |
      p.account->exists(a' | a'.account#=a.account#
        and a'.bank=b ) )->size >= 0
    and
    b.person->select(p |
      p.account->exists(a' | a'.account#=a.account#
        and a'.bank=b ) )->size <= 1 ) )
Equivalence Rule for Association Class

\[
C \rightarrow \text{forall} ( c \mid c.ra->size=1 \text{ and } c.rb->size=1 \text{ and } \\
C \rightarrow \text{forall} ( c' \mid ( c.ra=c'.ra \text{ and } c.rb=c'.rb ) \text{ implies } c=c' ))
\]
Example for Association Class

\[
\text{Account} \rightarrow \text{forall} ( a \mid \\
\quad a.\text{bank} \rightarrow \text{size}=1 \text{ and } a.\text{person} \rightarrow \text{size}=1 \\
\quad \text{and} \\
\quad \text{Account} \rightarrow \text{forall} ( a' \mid \\
\quad \quad ( a.\text{bank}=a'.\text{bank} \text{ and } a.\text{person}=a'.\text{person} ) \text{ implies } a=a' )
\]
Problems with UML Semantics document

Problem: Verbal semantics in UML Semantics document leads to ambiguities

Example: Aggregation and composition

An association may represent an aggregation (i.e., a whole/part relationship). In this case, the association-end attached to the whole element is designated, and the other association-end of the association represents the parts of the aggregation. Only binary associations may be aggregations. Composite aggregation is a strong form of aggregation which requires that a part instance be included in at most one composite at a time, although the owner may be changed over time. Furthermore, a composite implies propagation semantics (i.e., some of the dynamic semantics of the whole is propagated to its parts). For example, if the whole is copied or deleted, then so are the parts as well. A shared aggregation denotes weak ownership (i.e., the part may be included in several aggregates) and its owner may also change over time. However, the semantics of a shared aggregation does not imply deletion of the parts when one of its containers is deleted. Both kinds of aggregations define a transitive, antisymmetric relationship (i.e., the instances form a directed, non-cyclic graph). Composition instances form a strict tree (or rather a forest).

Text above is subject of many papers!

Further observation: Text above also mentions syntactical requirements (“Only binary associations may be aggregations.”)
Equivalence Rules for Aggregation and Composition (Part One)

Existential Dependency for the Part
\[ P \rightarrow \forall p \mid A \rightarrow \exists a \mid a.rp \rightarrow \text{includes}(p) \]

Existential Dependency for the Aggregate
\[ A \rightarrow \forall a \mid P \rightarrow \exists p \mid a.rp \rightarrow \text{includes}(p) \]
Equivalence Rules for Aggregation and Composition (Part Two)

Weak Form of Forbidding Sharing

\[ P \rightarrow \text{forall}(\ p \mid A \rightarrow \text{forall}(\ a, a' \mid \ (a.\rp\rightarrow\text{includes}(p) \ \text{and} \ a'.\rp\rightarrow\text{includes}(p) ) \ \text{implies} \ a=a')) \]
Equivalence Rules for Aggregation and Composition (Part Three)

Strong Form of Forbidding Sharing

\[
P \rightarrow \text{forall}(\ p \mid \\
A1 \rightarrow \text{forall}(\ a1 \mid \\
A2 \rightarrow \text{forall}(\ a2 \mid \\
(\ a1.\text{rp1} \rightarrow \text{includes}(p) \text{ and } a2.\text{rp2} \rightarrow \text{includes}(p) ) \\
\text{implies } a1 = a2))
\]
Equivalence Rules for Aggregation and Composition (Part Four)

\[
\text{parts: } P \rightarrow \text{Set}(P) \\
\text{aggregates: } P \rightarrow \text{Set}(P)
\]

Now possible but unwanted: \(p.\text{parts}->\text{includes}(p)\)

partsClosure(p) =
- \(\text{parts}(p) + \text{parts}(\text{parts}(p))' + \text{parts}(\text{parts}(\text{parts}(p)))' + \ldots\)

Forbidding Instance Reflexivity

\(P.\text{forAll}(\ p \ | \ not(\ p.\text{partsClosure}->\text{includes}(p) ) ) \)

Due to generalizations, reflexive associations may occur frequently!
## Association vs. Aggregation vs. Composition

<table>
<thead>
<tr>
<th></th>
<th>Association</th>
<th>Aggregation</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existential Dependency</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>for the Part</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Existential Dependency</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>for the Aggregate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weak Form of Forbidding Sharing</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Strong Form of Forbidding Sharing</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Forbidding Instance Reflexivity</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
‘Paper’ Example – Existential Dependency

Abstract\(\rightarrow_{\forall} a |\)
   Paper\(\rightarrow_{\exists} p | p.\text{abstract}\rightarrow\text{includes}(a)\)
Forbidding Sharing for Composition

Abstract->forAll( a | Paper->forAll( p, p' | ( p.abstract->includes(a) and p'.abstract->includes(a) ) implies p=p' ) )
‘Brain’ Example – Class Diagram

Fatherhood

KnowsByName

Fatherhood

KnowsByName

Human

Possession

Brain

father

child

father

Knower

known

known
‘Brain’ Example – Induced Requirements

Existential Part Dependency for Composition

Brain -> forall( b |
    Human -> exists( h | h.brain -> includes(b) ) )

Forbidding Sharing for Composition

Brain -> forall( b |
    Human -> forall( h, h' |
        (h.brain -> includes(b) and
        h'.brain -> includes(b)) implies h=h' ) )

Forbidding Instance Reflexivity for Aggregation

child: Human -> Set(Human)
childClosure: Human -> Set(Human) = offsprings

Human forall( h | not( h.childClosure -> includes(h) ) )
Human forall( h | not( h.offsprings -> includes(h) ) )

No restriction for Association!
‘Brain’ Example – Valid Object Diagram

Fatherhood: solid arc, KnowsByName: dashed arc, Possession: undirected line
‘Brain’ Example – Invalid Object Diagram

Objects and links drawn fat contribute to invalidity

Fatherhood: solid arc, KnowsByName: dashed arc, Possession: undirected line
Equivalence Rule for Generalization

\[ S \rightarrow \text{forAll}( s, s' \mid s \neq s' \implies s.rg \neq s'.rg ) \]

or equivalently

\[ S \rightarrow \text{forAll}( s, s' \mid s.rg = s'.rg \implies s = s' ) \]
4.2 Advanced Concepts in Class Diagrams

Equivalence Rule for Disjoint Generalization

\[ S1 \rightarrow \forall s1 \mid S2 \rightarrow \forall s2 \mid s1.g \not= s2.g \]

Diagram:

```
S1
{disjoint}
S2
  \rightarrow
G
  \rightarrow
S1
\ 0..1

\ 1..1

S2
\ 0..1

+ constraint
```
Equivalence Rule for Complete Generalization

\[ G \rightarrow \forall g' \left( S_1 \rightarrow \exists s_1 \mid s_1.g = g' \right) \text{ or } S_2 \rightarrow \exists s_2 \mid s_2.g = g' \]
Examples for (1) Plain Generalization and (2) Disjoint and Complete Generalization

\[\text{Car} \rightarrow \text{forAll}(c, c' \mid c \neq c' \implies c.\text{vehicle} \neq c'.\text{vehicle})\]

\[\text{Female} \rightarrow \text{forAll}(f \mid \text{Male} \rightarrow \text{forAll}(m \mid f.\text{person} \neq m.\text{person})\)\]

\[\text{Person} \rightarrow \text{forAll}(p \mid \text{Female} \rightarrow \text{exists}(f \mid f.\text{person} = p) \text{ or } \text{Male} \rightarrow \text{exists}(m \mid m.\text{person} = p)\)\]
Example for Disjoint and Incomplete Generalization

\[
\text{Oak} \rightarrow \text{forAll} ( o | \\
\text{Elm} \rightarrow \text{forAll} ( e | \\
\text{Birch} \rightarrow \text{forAll} ( b | \\
\quad o.\text{tree} \not\sim e.\text{tree} \hspace{0.2cm} \text{and} \\
\quad o.\text{tree} \not\sim b.\text{tree} \hspace{0.2cm} \text{and} \\
\quad e.\text{tree} \not\sim b.\text{tree} ) ) )
\]
Example for Overlapping and Complete Generalization

Vehicle->forall( v | LandVehicle->exists( l | l.vehicle=v ) )  
or  
WaterVehicle->exists( w | w.vehicle=v )  
or  
AirVehicle->exists( a | a.vehicle=v )  

\[
\text{Vehicle} \rightarrow \forall v \left( \text{LandVehicle} \rightarrow \exists l \left( l.\text{vehicle} = v \right) \right) \\
\text{or} \\
\text{WaterVehicle} \rightarrow \exists w \left( w.\text{vehicle} = v \right) \\
\text{or} \\
\text{AirVehicle} \rightarrow \exists a \left( a.\text{vehicle} = v \right)
\]
Example for Overlapping and Incomplete Generalization

No constraints!
4.3 Basic Concepts in Class Diagrams

- To follow: formal definition of basic concepts in class diagrams based on set theory
- Formal abstract syntax for the these basic concepts
- Formal interpretation for these basic concepts
Example Class Diagram
4.3 Basic Concepts in Class Diagrams

Definition 1 (Names)
Let \( \mathcal{N} \) be a set of Names.

Definition 2 (Basic Types)
We assume that there is a signature \( \Sigma = (T, \Omega) \) with \( T \) being a set of basic type names, and \( \Omega \) being a set of operations over types in \( T \):

\[
\omega : t_1 \times \cdots \times t_n \rightarrow t
\]
4.3 Basic Concepts in Class Diagrams

Example 1 (Basic Types and Operations)

*Predefined OCL Basic Types:*

\[ \{\text{Integer}, \text{Real}, \text{Boolean}, \text{String}\} \subseteq T \]

*Operations in $\Omega$ include, for example, the usual arithmetic operations $+, -, *, /$, etc. for integers, e.g.*

\[ + : \text{Integer} \times \text{Integer} \rightarrow \text{Integer} \]

**Definition 3 (Collection Types)**

*Collection types over the basic types are available for describing collections of values, for example, $\text{Set(} \text{String} \text{)}, \text{Bag(} \text{Integer} \text{)},$ and $\text{Sequence(} \text{Real} \text{)}, \text{Set(} \text{Sequence(} \text{Integer} \text{)} \text{)}.**
Example 2 (Collection Types)
In our example model, the attribute email of class Person has the type Set(String) to allow multiple email addresses for each person.

Definition 4 (Classes)
The set of classes is given by a finite set of names

\[ \text{CLASS} \subseteq \mathcal{N} \]

Example 3 (Class Definition)
The example class diagram defines the following set of classes:

\[ \text{CLASS} = \{ \text{Person, Check, ServiceDepot, Customer, Employee, Car, Branch, CarGroup, Rental} \} \]
Definition 5 (Attributes)

Let \( t \in T \) be a basic or collection type. The attributes of a class \( c \in \text{CLASS} \) are defined as a set \( \text{ATT}_c \) of signatures \( a : t_c \to t \) where the attribute name \( a \) is an element of \( \mathcal{N} \), and \( t_c \in T \) is the type of class \( c \).

Example 4 (Attribute)

\[
\text{ATT}_{\text{Person}} = \{ \text{firstname} : \text{Person} \to \text{String}, \\
\qquad \text{lastname} : \text{Person} \to \text{String}, \\
\qquad \text{age} : \text{Person} \to \text{Integer}, \\
\qquad \text{isMarried} : \text{Person} \to \text{Boolean}, \\
\qquad \text{email} : \text{Person} \to \text{Set} (\text{String}) \} 
\]
Definition 6 (Operations)
Let $t$ and $t_1, \ldots, t_n \in T$ be types. Operations of a class $c \in \text{CLASS}$ with type $t_c \in T$ are defined by a set of $\text{OP}_c$ of signatures $\omega : t_c \times t_1 \times \cdots \times t_n \rightarrow t$ with operation symbols $\omega \in \mathcal{N}$.

Example 5 (Operation)

\[
\text{OP}_{\text{Car}} = \{\text{description} : \text{Car} \rightarrow \text{String}\}
\]

\[
\text{OP}_{\text{Branch}} = \{\text{rentalsForDay} : \text{Branch} \times \text{String} \rightarrow \text{Set(Rental)}\}
\]

\[
\text{OP}_{\text{Employee}} = \{\text{raiseSalary} : \text{Employee} \times \text{Real} \rightarrow \text{Real}\}
\]
Definition 7 (Associations)
The set of associations is given by:

1. a finite set of names \( \text{ASSOC} \subseteq N \)
2. a function

\[
\text{associates} : \text{ASSOC} \rightarrow \text{CLASS}^+
\]

\[
as \mapsto \langle c_1, \ldots, c_n \rangle \text{ with } (n \geq 2)
\]
Example 6 (Association)

\[ \text{Assoc} = \\{ \text{Booking, Management, Employment, Fleet, Classification, Assignment, Offer, Quality, Provider, Reservation} \} \]

\[
\begin{align*}
\text{associates(Assignment)} & = \langle \text{Rental, Car} \rangle \\
\text{associates(Booking)} & = \langle \text{Rental, Customer} \rangle \\
\text{associates(Classification)} & = \langle \text{CarGroup, Car} \rangle \\
\text{associates(Employment)} & = \langle \text{Branch, Employee} \rangle \\
\text{associates(Fleet)} & = \langle \text{Branch, Car} \rangle \\
\text{associates(Maintenance)} & = \langle \text{ServiceDepot, Check, Car} \rangle \\
\end{align*}
\]

\[
\ldots
\]
Definition 8 (Role names)

Let \( as \in \text{Assoc} \) be an association with associates\((as) = \langle c_1, \ldots, c_n \rangle\).

Role names for an association are defined by a function

\[
\text{roles} : \text{Assoc} \rightarrow \mathbb{N}^+
\]

\[
as \mapsto \langle r_1, \ldots, r_n \rangle
\]

where all role names must be distinct, i.e.,

\[
\forall i, j \in \{1, \ldots, n\} : i \neq j \implies r_i \neq r_j.
\]

If role names are not specified in the class diagram, then implicit role names consisting of the class name with a lower case first letter are used.

In case of ambiguities, for example, in the case where two associations between two classes are present, role names must be specified in the class diagram.
Example 7 (Role names)

\[
\begin{align*}
\text{roles(Assignment)} &= \langle \text{rental, car} \rangle \\
\text{roles(Booking)} &= \langle \text{rental, customer} \rangle \\
\text{roles(Classification)} &= \langle \text{carGroup, car} \rangle \\
\text{roles(Employment)} &= \langle \text{employer, employee} \rangle \\
\text{roles(Fleet)} &= \langle \text{branch, car} \rangle \\
\text{roles(Maintenance)} &= \langle \text{serviceDepot, check, car} \rangle \\
\text{roles(Management)} &= \langle \text{managedBranch, manager} \rangle \\
\text{roles(Offer)} &= \langle \text{branch, carGroup} \rangle \\
\text{roles(Provider)} &= \langle \text{rental, branch} \rangle \\
\text{roles(Quality)} &= \langle \text{lower, higher} \rangle \\
\text{roles(Reservation)} &= \langle \text{rental, carGroup} \rangle
\end{align*}
\]
Definition 9 (Multiplicities)

Let $as \in Assoc$ be an association with $\text{associates}(as) = \langle c_1, \ldots, c_n \rangle$. The function $\text{multiplicities}(as) = \langle M_1, \ldots, M_n \rangle$ assigns each class $c_i$ participating in the association a non-empty set $M_i \subset \mathbb{N}_0$ with $M_i \neq \{0\}$ for all $1 \leq i \leq n$.

Example 8 (Multiplicities)

For example, the multiplicities of the Maintenance association are defined as $\text{multiplicities}(\text{Maintenance}) = \langle \{0, 1\}, \mathbb{N}_0, \mathbb{N}_0 \rangle$. 
Definition 10 (Generalization hierarchy)

A generalization hierarchy $\prec$ is a partial order (transitive, non-reflexive relation) on the set of classes $\text{CLASS}$.

Example 9 (Generalization hierarchy)

The example class diagram contains two generalizations. Person is the parent class of Customer and Employee.

$\prec = \{(\text{Customer}, \text{Person}), (\text{Employee}, \text{Person})\}$
Definition 11 (Parents)

\[ parents : \text{CLASS} \rightarrow \mathcal{P}(\text{CLASS}) \]
\[ c \mapsto \{ c' | c' \in \text{CLASS} \land c \prec c' \} \]

Definition 12 (Full set of attributes)

\[ \text{ATT}_c^* = \text{ATT}_c \cup \bigcup_{c' \in \text{parents}(c)} \text{ATT}_{c'} \]
4.3 Basic Concepts in Class Diagrams

Definition 13 (Inherited user-defined operations)

\[ OP_c^* = OP_c \cup \bigcup_{c' \in \text{parents}(c)} OP_{c'} \]

Definition 14 (Navigable role names)

\[ \text{navends}^*(c) = \text{navends}(c) \cup \bigcup_{c' \in \text{parents}(c)} \text{navends}(c') \]
Definition 15 (Full descriptor of a class)

The full descriptor of a class $c \in \text{CLASS}$ is a structure $\text{FD}_c = (\text{ATT}^*_c, \text{OP}^*_c, \text{navends}^*(c))$ containing all attributes, user-defined operations, and navigable role names defined for the class and all of its parents.
Additional well-formedness rules

1. In a full class descriptor, attributes are defined in exactly one class
2. In a full class descriptor, an operation may only be defined once
3. In a full class descriptor, role names are defined exactly in one class
4. Attribute names and role names must not conflict
Overview on Formal Syntax

Definition 16 (Syntax of object models)

The syntax of an object model is a structure

\[ M = (\text{CLASS}, \text{ATT}_c, \text{OP}_c, \text{ASSOC}, \text{associates}, \text{roles}, \text{multiplicities}, \prec) \]

where

1. \text{CLASS} is a set of classes.

2. \text{ATT}_c is a set of operation signatures for functions mapping an object of class \( c \) to an associated attribute value.

3. \text{OP}_c is a set of signatures for user-defined operations of a class \( c \).
4. \textbf{Assoc} is a set of association names.

(a) \textit{associates} is a function mapping each association name to a list of participating classes.

(b) \textit{roles} is a function assigning each end of an association a role name.

(c) \textit{multiplicities} is a function assigning each end of an association a multiplicity specification.

5. \textit{≺} is a partial order on \texttt{CLASS} reflecting the generalization hierarchy of classes.
Definition 17 (Object Identifiers)

1. The set of object identifiers of a class $c \in \text{CLASS}$ is defined by an infinite set $\text{oid}(c) = \{c_1, c_2, \ldots\}$.

2. The domain of a class $c \in \text{CLASS}$ is defined as

$$I_{\text{CLASS}}(c) = \bigcup \{\text{oid}(c') \mid c' \in \text{CLASS} \land c' \preceq c\}.$$
Example 10 (Object Identifiers)

\( l(\text{Branch}) = \{ \overline{\text{Branch}}_1, \overline{\text{Branch}}_2, \ldots \} \)

\( l(\text{Customer}) = \{ \overline{\text{cu}}_1, \overline{\text{cu}}_2, \ldots \} \)

*(Class names may be abbreviated)*

\( l(\text{Employee}) = \{ \overline{\text{e}}_1, \overline{\text{e}}_2, \ldots \} \)

\( l(\text{Person}) = \{ \overline{\text{p}}_1, \overline{\text{p}}_2, \ldots \} \cup l(\text{Customer}) \cup l(\text{Employee}) \)
Generalization

The above definition implies that a generalization hierarchy induces a subset relation on the semantic domain of classes.

\[ \forall c_1, c_2 \in \text{CLASS} : c_1 \prec c_2 \implies I(c_1) \subset I(c_2). \]

Example 11 (Inheritance)

\[ I(\text{Customer}) \subset I(\text{Person}) \]
\[ I(\text{Employee}) \subset I(\text{Person}) \]
Definition 18 (Links)
Each association $as \in \text{ASSOC}$ with associates$(as) = \langle c_1, \ldots, c_n \rangle$ is interpreted as the Cartesian product of the sets of object identifiers of the participating classes:

$$I_{\text{ASSOC}}(as) = I_{\text{CLASS}}(c_1) \times \ldots \times I_{\text{CLASS}}(c_n)$$

A link denoting a connection between objects is an element $l_{as} \in I_{\text{ASSOC}}(as)$. 
Example 12 (Links)

\[ I(\text{Assignment}) = I(\text{Rental}) \times I(\text{Car}) \]
\[ I(\text{Maintenance}) = I(\text{ServiceDepot}) \times I(\text{Check}) \times I(\text{Car}) \]
\[ I(\text{Quality}) = I(\text{CarGroup}) \times I(\text{CarGroup}) \]
Definition 19 (System State)
A system state for a model $\mathcal{M}$ is a structure
$\sigma(\mathcal{M}) = (\sigma_{\text{CLASS}}, \sigma_{\text{ATT}}, \sigma_{\text{ASSOC}})$.

1. Finite sets $\sigma_{\text{CLASS}}(c)$ contain all objects of a class $c \in \text{CLASS}$ existing in the system state:
   $\sigma_{\text{CLASS}}(c) \subset \text{oid}(c)$.

2. Functions $\sigma_{\text{ATT}}$ assign attribute values to each object:
   $\sigma_{\text{ATT}}(a) : \sigma_{\text{CLASS}}(c) \rightarrow I(t)$ for each $a : t_c \rightarrow t \in \text{ATT}_c^*$. 
3. Finite sets $\sigma_{\text{ASSOC}}$ contain links connecting objects. For each $as \in \text{ASSOC}$: $\sigma_{\text{ASSOC}}(as) \subset I_{\text{ASSOC}}(as)$. A link set must satisfy all multiplicity specifications defined for an association (the function $\pi_i(l)$ projects the $i$th component of a tuple or list $l$, whereas the function $\bar{\pi}_i(l)$ projects all but the $i$th component):

$$\forall i \in \{1, \ldots, n\}, \forall l \in \sigma_{\text{ASSOC}}(as):$$

$$|\{l' \mid l' \in \sigma_{\text{ASSOC}}(as) \land (\bar{\pi}_i(l') = \bar{\pi}_i(l))\}| \in \pi_i(\text{multiplicities}(() = as))$$
Example 13 (System State)

\[ \sigma(Employee) = \{e_1, e_2\} \]
\[ \sigma(Branch) = \{b_1\} \]
\[ \sigma(Employment) = \{(b_1, e_1), (b_1, e_2)\} \]
\[ \sigma(Management) = \{(b_1, e_1)\} \]
Definition 20 (Interpretation of object models)

The interpretation of an object model $\mathcal{M}$ is the set of all possible system states $\sigma(\mathcal{M})$. 
Syntax and Semantics as Meta-model