Recall: Corner Cases

Closed Object Diagrams vs. Dangling References

Find the 10 differences! (Both diagrams shall be complete.)

1. C:
   C₅ C₆

2. C:
   C₅ C₆

Definition.

Let \( \sigma \) be a system state. We say attribute \( v \in V \) has a dangling reference in object \( u \in \text{dom}(\sigma) \) if and only if the attribute's value comprises an object which is not alive in \( \sigma \), i.e. if \( \sigma(u)(v) \not\subset \text{dom}(\sigma) \).

We call \( \sigma \) closed if and only if no attribute has a dangling reference in any object alive in \( \sigma \).
Closed Object Diagrams vs. Dangling References

Find the 10 differences! (Both diagrams shall be complete.)

\[ \text{Definition.} \]
Let \( \sigma \) be a system state. We say attribute \( v \in V_{0,1,\ast} \) has a dangling reference in object \( u \in \text{dom}(\sigma) \) if and only if the attribute's value comprises an object which is not alive in \( \sigma \), i.e., if \( \sigma(u)(v) \not\subset \text{dom}(\sigma) \).

We call \( \sigma \) closed if and only if no attribute has a dangling reference in any object alive in \( \sigma \).

**Observation:** Let \( G \) be the (complete) object diagram of a closed system state \( \sigma \).
Then the nodes in \( G \) are labelled with \( T \)-typed attributes.

**Special Notation**

- \( S = (\{\text{Int}\}, \{C\}, \{n,p: C\ast\} , \{C \mapsto \{n,p\}\}) \).
- Instead of \( 1C:C 5C:C n \) we want to write \( 1C:C p = \emptyset 5C:C p = \emptyset n \) or \( 1C:C p = n | p | p \) to explicitly indicate that attribute \( p:C\ast \) has value \( \emptyset \) (also for \( p:C0,1 \ast \)).

**Aftermath**

- We slightly deviate from the standard (for reasons):
  - In the course, \( C0,1\ast \)-typed attributes only have sets as values.
    UML also considers multisets, that is, they can have \( u_1:C u_2:C u_3:D \).
    This is not an object diagram in the sense of our definition because of the requirement on the edges \( E \) between in-object graphs and the object graph (1).
  - We allow to give the evaluation of \( C0,1 \)-or \( C\ast \)-typed attributes in the values compartment.
    Allows us to indicate that a certain \( n \) is not referring to another object.
    Allows us to represent "dangling references", i.e., references to objects which are not alive in the current system state.
  - We introduce a graphical representation of values.

**Example: Object Diagrams for Documentation**

The Other Way Round

- In this example, we can conclude by "goodwill" that the author is referring to some signature \( S = (T, C, V, \text{atr}) \) with at least:
  - \( \{C,D\} \subseteq C \),
  - \( T \in T \),
  - \( \{x:C\ast, p:C\ast, z:T\} \subseteq V \),
  - \( \{x\} \subseteq \text{atr}(C) \),
  - \( \{p,z\} \subseteq \text{atr}(D) \),
  - and a structure with
    - \( \{u_1,u_2\} \subseteq D(C) \),
    - \( u_3 \in D(D) \),
    - \( 0 \in D(T) \).
In the following, $S$ denotes a signature and $D$ a structure of $S$.

**Definition (Satisfaction Relation).**

Let $\phi$ be an OCL constraint over $S$ and $\sigma \in \Sigma^D_S$ a system state. We write

- $\sigma \models \phi$ if and only if $I/llbracket \phi/rrbracket(\sigma, \emptyset) = true$.
- $\sigma \nmodels \phi$ if and only if $I/llbracket \phi/rrbracket(\sigma, \emptyset) = false$.

**Note:** In general we can't conclude from $\neg(\sigma \models \phi)$ to $\sigma \nmodels \phi$ or vice versa.

**Object Diagrams and OCL**

- Let $G$ be an object diagram of signature $S$ wrt. structure $D$.
  - Let $\text{expr}$ be an OCL expression over $S$.
    - We say $G$ satisfies $\text{expr}$, denoted by $G \models \text{expr}$, if and only if $\forall \sigma \in G^->: \sigma \models \text{expr}$.
    - If $G$ is complete, we can also talk about "$\nmodels \text{expr}$". (Otherwise, to avoid confusion, avoid "$\nmodels \text{expr}$": $G^-$ could comprise system states in which $\text{expr}$ evaluates to $true$, $false$, and $\bot$.)

**Example:**

- $C$: $C.p = \emptyset$
- $C$: $C.n = \emptyset p = \emptyset$
- $D$: $D.x = 23$
- $\text{context } C \text{ inv}: n \rightarrow \text{isEmpty}()$
- $\text{context } C \text{ inv}: p.n \rightarrow \text{isEmpty}()$
- $\text{context } D \text{ inv}: x \neq 0$
OCL Consistency

Definition (Consistency).

A set \( \text{Inv} = \{ \phi_1, \ldots, \phi_n \} \) of OCL constraints over \( S \) is called consistent (or satisfiable) if and only if there exists a system state of \( S \) w.r.t. \( D \) which satisfies all of them, i.e. if \( \exists \sigma \in \Sigma_D^S : \sigma \models \phi_1 \land \ldots \land \sigma \models \phi_n \) and inconsistent (or unrealizable) otherwise.

Deciding OCL Consistency

- Whether a set of OCL constraints is satisfiable or not is in general not as obvious as in the made-up example.
- Wanted: A procedure which decides the OCL satisfiability problem.
- Unfortunately: In general undecidable.

Otherwisewe could, for instance, solve diophantine equations:

\[
c_1 x_1^{n_1} + \cdots + c_m x_m^{n_m} = d.
\]

**Encoding in OCL:**

\[
\text{allInstances } C \rightarrow \exists (w : C | c_1 \ast w.x_1^{n_1} + \cdots + c_m \ast w.x_m^{n_m} = d).
\]

- And now?
- Options: [Cabot and Claris´ o, 2008]
  - Constrain OCL, use a less rich fragment of OCL.
  - Revert to finitedomains — basic types vs. number of objects.
OCL Critique

• Expressive Power:
  • "Pure OCL expressions only compute primitive recursive functions, but not recursive functions in general."
  
  [Cengarle and Knapp, 2001]

• Evolution over Time:
  • "finally self.x > 0"
  
  Proposals for fixes e.g. [Flake and Müller, 2003]. (Or: sequencediagrams.)

• Real-Time:
  • "Objects respond within 10s"
  
  Proposals for fixes e.g. [Cengarle and Knapp, 2002].

• Reachability:
  • "After insert operation, nodes shall be reachable."
  
  Fix: add transitive closure.

• Concrete Syntax:
  • "The syntax of OCL has been criticized – e.g., by the authors of Catalysis [...] – for being hard to read and write."
  • OCL's expressions are stacked in the style of Smalltalk, which makes it hard to see the scope of quantified variables.
  • Navigations are applied to atoms and not sets of atoms, although there is a collect operation that maps a function over a set.
  • Attributes, [...], are partial functions in OCL, and result in expressions with undefined value."
  
  [Jackson, 2002]

UML Class Diagrams: Stocktaking

Wanted: places in the signature to represent the information from the picture.

Extended Signature
Recall: Signature

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

where

- \((\text{basic})\) types \(T\) and classes \(C\), (both finite),
- typed attributes \(V\), \(\tau\) from \(T\) or \(C\), \(0, 1\) or \(C^*\), \(C \in C\),
- \(\text{atr}: C \to 2^V\) mapping classes to attributes.

Too abstract to represent class diagram, e.g. no "place" to put class stereotypes or attribute visibility.

So:

- Extend definition for classes and attributes: Just as attributes already have types, we will assume that
  - classes have (among other things) stereotypes
  - attributes have (in addition to a type and other things) a visibility.

Extended Classes

From now on, we assume that each class \(C \in C\) has:

- a finite (possibly empty) set \(S_C\) of stereotypes,
- a boolean flag \(a \in B\) indicating whether \(C\) is abstract,
- a boolean flag \(t \in B\) indicating whether \(C\) is active.

We use \(S_C\) to denote the set \(\bigcup C \in C S_C\) of stereotypes in \(S\).

(Alternatively, we could add a set \(St\) as 5-th component to \(S\) to provide the stereotypes (names of stereotypes) to choose from. But: too unimportant to care.)

Convention:

- We write \(\langle C, S_C, a, t \rangle \in C\) when we want to refer to all aspects of \(C\).
- If the new aspects are irrelevant (for a given context), we simply write \(C \in C\) i.e. old definitions are still valid.

Extended Attributes

- From now on, we assume that each attribute \(v \in V\) has (in addition to the type):
  - a visibility \(\xi \in \{\text{public}, \text{private}, \text{protected}, \text{package}\}\),
  - an initial value \(\text{expr}_0\) given as a word from language for initial values, e.g. OCL expressions.
  (If using Java as action language (later) Java expressions would be fine.)
  - a finite (possibly empty) set of properties \(P_v\).

We define \(P_C\) analogously to stereotypes.
From now on, we assume that each attribute \( v \in V \) has (in addition to the type):

- a visibility \( \xi \in \{ \text{public}, \text{private}, \text{protected}, \text{package} \} \)

- an initial value \( \text{expr}_0 \) given as a word from the language for initial values, e.g., OCL expressions. (If using Java as action language (later) Java expressions would be fine.)

- a finite (possibly empty) set of properties \( P_v \).

We define \( P_C \) analogously to stereotypes.

Convention:

- We write \( \langle v: \tau, \xi, \text{expr}_0, P_v \rangle \in V \) when we want to refer to all aspects of \( v \).

- Write only \( v: \tau \) or \( v \) if details are irrelevant.

- Note: All definitions we have up to now essentially still apply as they are.

- The other way round: most of the newly added aspects don't contribute to the constitution of system states or object diagrams.
What If Things Are Missing?

There is no "novisibility"—an attribute has no initial value, no invisibility has
tribute to the constitution of system states or object diagrams.

The other way round:

- We view a class box from Class Diagrams to Extended Signatures

\[
\begin{align*}
\text{From Class Diagram to Extended Signatures} & \quad \text{What If Things Are Missing?}
\end{align*}
\]
IstheMapping aFunction?

• Is $S(CD)$ well-defined?

Two possible sources for problems:

(1) A class $C$ may appear in multiple class diagrams:

(i) $C_v: \text{Int}$

(ii) $C_v: \text{Bool}$

Simply forbid the case (ii)—easy syntactical check on diagram.

(2) An attribute $v$ may appear in multiple classes:

$C_v: \text{Bool}$

$D_v: \text{Int}$

Two approaches:

• Require unique attribute names. This requirement can easily be established (implicitly, behind the scenes) by viewing $v$ as an abbreviation for $C::v$ or $D::v$ depending on the context. ($C::v: \text{Bool}$ and $D::v: \text{Int}$ are unique.)

• Subtle, formalist's approach: observe that $\langle v: \text{Bool} , \ldots \rangle$ and $\langle v: \text{Int} , \ldots \rangle$ are different things in $\Sigma$. But we don't follow that path.

ClassDiagramSemantics

• The semanticsofasetof class diagrams $CD$ firstof all isthe induced (extended) signature $S(CD)$.

• The signature gives rise to a set of system states given a structure $D$.

• Do we need to redefine/extend $D$?

No. (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type $\tau$, i.e., the set $D(\tau)$, would be determined by the class diagram, and not free for choice.)

• What is the effect on $\Sigma^{DS}$?
The semantics of a set of class diagrams CD first of all is the induced (extended) signature $S(CD)$.

The signature gives rise to a set of system states given a structure $D$.

• Do we need to redefine/extend $D$? No. (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type $\tau$, i.e., the set $D(\tau)$, would be determined by the class diagram, and not free for choice.)

What is the effect on $\Sigma DS$? Little.

For now, we only remove abstract class instances, i.e., $\sigma: D(C) \not\rightarrow (V \not\rightarrow (D(T) \cup D(C^*) ))$ is now only called system state if and only if, for all $\langle C,S,C,1,t \rangle \in C$, $\text{dom}(\sigma) \cap D(C) = \emptyset$.

With $a = 0$ as default "abstractness", the earlier definitions apply directly. We'll revisit this when discussing inheritance.

What about the rest?

• Classes:
  • Active: not represented in $\sigma$. Later: relevant for behaviour, i.e., how system states evolve over time.
  • Stereotypes: in a minute.

• Attributes:
  • Initial value: not represented in $\sigma$. Later: provides an initial value as effect of "creation action".
  • Visibility: not represented in $\sigma$. Later: viewed as additional typing information for well-formedness of system transformers; and with inheritance.

• Properties: such as $\text{readOnly}$, $\text{ordered}$, $\text{composite}$ (Deprecated in the standard).
  • $\text{readOnly}$ — later treated similar to visibility.
  • $\text{ordered}$ — too fine for our representation.
  • $\text{composite}$ — cf. lecture on associations.

Stereotypes as labels or tags

• So, a class is $\langle C,S,C,a,t \rangle$ with $a$ the abstractness flag, $t$ the activeness flag, and $S$ a set of stereotypes.

What are Stereotypes?

• Not represented in system states.
• Not contributing to typing rules.
(cf. later lecture on type theory for UML)

• [Oestereich, 2006]: View stereotypes as (additional) "labelling" ("tags") or as "grouping". Useful for documentation and MDA.

• Documentation: e.g., layers of an architecture. Sometimes, packages (cf. the standard) are sufficient and "right".

• Model-Driven Architecture (MDA): later.

Example: Stereotypes for Documentation

Core View
Application/Qt
Trace
sort
move
filter
jump
zoom
View/Qt

• Example: Timing Diagram Viewer [Schumann et al., 2008]

Architecture of four layers:
• core, data layer
• abstract view layer
• toolkit-specific view layer/widget
• application using widget

• Stereotype "= layer" = colour

Example: Stereotypes as Inheritance

• Another view (due to whom?): distinguish
• Technical Inheritance
If the target platform, such as the programming language for the implementation of the blueprint, is object-oriented, assume a 1-on-1 relation between inheritance in the model and on the target platform.

• Conceptual Inheritance
Only meaningful with a common idea of what stereotypes stand for. For instance, one could label each class with the team that is responsible for realizing it. Or with licensing information (e.g., LGPL and proprietary).

Or one could have labels understood by code generators (cf. lecture on MDSE).

• Confusing:
• Inheritance is often referred to as the "isa"-relation. Sharing a stereotype also expresses "being something".

• We can always (ab-)use UML-inheritance for the conceptual case, e.g.

Example of Stereotypes in Documentation
Recall: In lecture 03, we introduced OCL expressions with types, for instance:

\[ \text{expr} ::= w : \tau . . \text{logical variable} \]
\[ \text{true} \mid \text{false} : \text{Bool}. . \text{constants} \]
\[ 0 \mid -1 \mid 1 . . : \text{Int}. . \text{constants} \]
\[ \text{expr}_1 + \text{expr}_2 : \text{Int} \times \text{Int} \to \text{Int}. . \text{operation} \]
\[ \text{size}(\text{expr}_1) : \text{Set}(\tau) \to \text{Int} \]

Wanted: A procedure to tell well-typed, such as \((w : \text{Bool})\) not \(w\) from not well-typed, such as, \(\text{size}(w)\).

Approach: Derivation System, that is, a finite set of derivation rules. We then say \(\text{expr}\) is well-typed if and only if we can derive \(A, C \vdash \text{expr} : \tau\) (read: "expression \(\text{expr}\) has type \(\tau\)"), for some OCL type \(\tau\), i.e., \(\tau \in T_B \cup T_C \cup \{\text{Set}(\tau_0) \mid \tau_0 \in T_B \cup T_C\}\), \(C \in C\).
• If \( \text{expr} \) is a boolean constant, then \( \text{expr} \) is of type \( \text{Bool} \): \[
\text{(BOOL)} \vdash B : \text{Bool}, \quad B \in \{ \text{true}, \text{false} \}
\]

• If \( \text{expr} \) is an integer constant, then \( \text{expr} \) is of type \( \text{Int} \): \[
\text{(INT)} \vdash N : \text{Int}, \quad N \in \{ 0, 1, -1, \ldots \}
\]

• If \( \text{expr} \) is the application of operation \( \omega : \tau_1 \times \cdots \times \tau_n \rightarrow \tau \) to expressions \( \text{expr}_1, \ldots, \text{expr}_n \) which are of type \( \tau_1, \ldots, \tau_n \), then \( \text{expr} \) is of type \( \tau \): \[
\vdash \text{expr}_1 : \tau_1, \ldots, \vdash \text{expr}_n : \tau_n, \quad \vdash \omega(\text{expr}_1, \ldots, \text{expr}_n) : \tau,
\quad \omega / \in \text{atr}(\mathcal{C})
\]

\[(\text{Note: this rule also covers'=} \tau', 'isEmpty', and'size'.)\]
### Example

τ

*isoftype* as denoted by C for an object of τ of an attribute of type

*expr*:

w ⊢ A

∀w ∈ τ, w ⊢ ∈ w

C

*occurs in*:

• set all instances

C

all instances refer to

• type environment: in any

C

*Environment Introduction and Logical Variables*

C

*Type Environment Example*


Example

\[ \begin{align*}
\{ x & \mid x \in \mathbb{R} \} = \mathbb{R} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x > 0 \} = \mathbb{R}^+ \\
\{ x & \mid x \in \mathbb{R} \text{ and } -1 < x < 1 \} = (-1, 1) \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \in \mathbb{Q} \} = \mathbb{Q} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \in \mathbb{Z} \} = \mathbb{Z} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is an integer} \} = \mathbb{Z} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is a real number} \} = \mathbb{R} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is a complex number} \} = \mathbb{C} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is a rational number} \} = \mathbb{Q} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is an integer and } x \text{ is even} \} = \mathbb{Z}_{\text{even}} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is an integer and } x \text{ is odd} \} = \mathbb{Z}_{\text{odd}} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is a real number and } x \text{ is a rational number} \} = \mathbb{Q} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is a real number and } x \text{ is an integer} \} = \mathbb{Z} \\
\{ x & \mid x \in \mathbb{R} \text{ and } x \text{ is a real number and } x \text{ is a rational number} \} = \mathbb{Q} \\
\end{align*} \]