Software Design, Modelling and Analysis in UML

Lecture 04: OCL Cont’d, Object Diagrams

2011-10-31

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:
• OCL Syntax

This Lecture:
• Educational Objectives: Capabilities for following tasks/questions.
  • 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)?
  • Is this an object diagram wrt. to that other thing?
  • How are system states and object diagrams related?
  • What does it mean that an OCL expression is satisfiable?
  • When is a set of OCL constraints said to be consistent?
  • Can you think of an object diagram which violates this OCL constraint?

• Content:
  • OCL Semantics
  • Object Diagrams
  • Example: Object Diagrams for Documentation
  • OCL: consistency, satisfiability
**OCL Semantics [OMG, 2006]**

### The Task

**OCL Syntax 1/4: Expressions**

| expr ::= | w : τ(w) |
| expr → expr2 | : τ × τ → Bool |
| oclIsUndefined(expr1) | : τ → Bool |
| {expr1, ..., expr_n} | : τ × · · · × τ → Set(τ) |
| size(expr1) | : Set(τ) → Int |
| allInstances_C | : Set(τ_C) |
| v(expr1) | : τ_C → τ(v) |
| r1(expr1) | : τ_C → τ_D |
| r2(expr1) | : τ_C → Set(τ_D) |

Where, given $\mathcal{P} = (\mathcal{F} \cup \mathcal{V}, \text{atr})$:

- $W = \{\text{self}\}$ is a set of typed logical variables, $w$ has type $τ(w)$
- $τ$ is any type from $\mathcal{F} \cup \mathcal{T}_B \cup \mathcal{T}_/BV$
- $\mathcal{T}_B$ is a set of basic types, in the following we use $\mathcal{T}_B = \{\text{Bool}, \text{Int}, \text{String}\}$
- $\mathcal{T}_/BV = \{τ_C | C ∈ \mathcal{V}\}$ is the set of object types
- Set($τ_0$) denotes the set-of-$τ_0$ type for $τ_0 ∈ \mathcal{T}_B \cup \mathcal{T}_/BV$
  (sufficient because of "flattening" (cf. standard))
- $v : τ(e) ∈ \text{atr}(C), τ(e) ∈ \mathcal{F}$
- $r_1 : D_{/BV} ∈ \text{atr}(C)$
- $r_2 : D_{/BV} ∈ \text{atr}(C)$
- $C, D ∈ \mathcal{V}$

- Given an OCL expression $expr$, a system state $σ ∈ Σ_{\mathcal{P}}$, and a valuation of logical variables $β$, define

$$I[\cdot](\cdot, \cdot) : \text{OCLExpressions}(\mathcal{P}) × Σ_{\mathcal{P}} × (W → I(\mathcal{F} \cup \mathcal{T}_B \cup \mathcal{T}_/BV)) → I(\text{Bool})$$

such that

$$I[expr](σ, β) ∈ \{\text{true, false, ⊥}_\text{Bool}\}.$$
Basically business as usual...

(i) Equip each OCL (!) basic type with a reasonable domain, i.e. define function

\[ I \] with \( \text{dom}(I) = \{ \text{true}, \text{false} \} \cup \{ \bot_{\text{Bool}} \} \)

(ii) Equip each object type \( \tau_C \) with a reasonable domain, i.e. define function

\[ I \] with \( \text{dom}(I) = \tau_C \)

(most reasonable: \( \mathcal{D}(C) \) determined by structure \( \mathcal{D} \) of \( \mathcal{S} \)).

(iii) Equip each set type \( \text{Set}(\tau_0) \) with reasonable domain, i.e. define function

\[ I \] with \( \text{dom}(I) = \{ \text{Set}(\tau_0) \mid \tau_0 \in \mathcal{T}_B \cup \mathcal{T}_W \} \)

(iv) Equip each arithmetical operation with a reasonable interpretation (that is, with a function operating on the corresponding domains).

\[ I \] with \( \text{dom}(I) = \{ +, -, \leq, \ldots \} \), e.g., \( I(+) \in I(\text{Int}) \times I(\text{Int}) \rightarrow I(\text{Int}) \)

(v) Set operations similar. \( I \) with \( \text{dom}(I) = \{ \text{isEmpty}, \ldots \} \)

(vi) Equip each expression with a reasonable interpretation, i.e. define function

\[ I : \text{Expr} \times \Sigma_{\mathcal{D}} \times (W \rightarrow I(\mathcal{T} \cup \mathcal{T}_B \cup \mathcal{T}_W)) \rightarrow I(\text{Bool}) \]

...except for OCL being a three-valued logic, and the "iterate" expression.

(i) Domains of Basic Types

Recall:
- \( \mathcal{T}_B = \{ \text{Bool}, \text{Int}, \text{String} \} \)

We set:
- \( I(\text{Bool}) := \{ \text{true}, \text{false} \} \cup \{ \bot_{\text{Bool}} \} \)
- \( I(\text{Int}) := \mathbb{Z} \cup \{ \bot_{\text{Int}} \} \)
- \( I(\text{String}) := \ldots \cup \{ \bot_{\text{String}} \} \)

We may omit index \( \tau \) of \( \bot_{\tau} \) if it is clear from context.
(ii) Domains of Object and (iii) Set Types

- Now we need a structure \( \mathcal{D} \) of our signature \( \mathcal{S} = (\mathcal{F}, \mathcal{C}, V, \text{atr}) \).
- Recall: \( \mathcal{D} \) assigns an (infinite) domain \( \mathcal{D}(C) \) to each class \( C \in \mathcal{C} \).

- Let \( \tau_C \) be an (OCL) object type for a class \( C \in \mathcal{C} \).
- We set
  \[
  I(\tau_C) := \mathcal{D}(C) \cup \{\bot_{\tau_C}\}
  \]

- Let \( \tau \) be a type from \( T_B \cup T_C \).
- We set
  \[
  I(\text{Set}(\tau)) := \mathcal{2}^{I(\tau)} \cup \{\bot_{\text{Set}(\tau)}\}
  \]

Note: in the OCL standard, only finite subsets of \( I(\tau) \). But infinity doesn’t scare us, so we simply allow it.

(iv) Interpretation of Arithmetic Operations

- Literals map to fixed values:
  \[
  I(\text{true}) := \text{true}, \quad I(\text{false}) := \text{false}, \quad I(0) := 0, \quad I(1) := 1, \ldots
  \]
  \[
  I(\text{OclUndefined}) := \bot
  \]

- Boolean operations (defined point-wise for \( x_1, x_2 \in I(\tau) \)):  
  \[
  I(\text{=}) (x_1, x_2) := \begin{cases} 
  \text{true}, & \text{if } x_1 \neq \bot \neq x_2 \text{ and } x_1 = x_2 \\
  \text{false}, & \text{if } x_1 \neq \bot \neq x_2 \text{ and } x_1 \neq x_2 \\
  \bot_{\text{Bool}}, & \text{otherwise}
  \end{cases}
  \]

- Integer operations (defined point-wise for \( x_1, x_2 \in I(\text{Int}) \)):
  \[
  I(+) (x_1, x_2) := \begin{cases} 
  x_1 + x_2, & \text{if } x_1 \neq \bot \neq x_2 \\
  \bot, & \text{otherwise}
  \end{cases}
  \]

Note: There is a common principle. Namely, the interpretation of an operation \( \omega : \tau_1 \times \ldots \times \tau_n \rightarrow \tau \) is a function \( I(\omega) : I(\tau_1) \times \cdots \times I(\tau_n) \rightarrow I(\tau) \) on corresponding semantical domain(s).
(iv) Interpretation of OclIsUndefined

- The **is-undefined** predicate (defined point-wise for \( x \in I(\tau) \)):

\[
I(\text{oclIsUndefined}_\tau)(x) := \begin{cases} 
true & \text{, if } x = \bot_{\tau} \\
false & \text{, otherwise}
\end{cases}
\]

(v) Interpretation of Set Operations

Basically the same principle as with arithmetic operations...

Let \( \tau \in T_B \cup T_{\text{op}} \).

- **Set comprehension** \((x_1, \ldots, x_n \in I(\tau))\):

\[
I(\{\}^n)(x_1, \ldots, x_n) := \{x_1, \ldots, x_n\}
\]

for all \( n \in \mathbb{N}_0 \)

- **Empty-ness check** \((x \in I(\text{Set}(\tau)))\):

\[
I(\text{isEmpty}_\tau)(x) := \begin{cases} 
true & \text{, if } x = \emptyset \\
\bot_{\text{Bool}} & \text{, if } x = \bot_{\text{Set}(\tau)} \\
false & \text{, otherwise}
\end{cases}
\]

- **Counting** \((x \in I(\text{Set}(\tau)))\):

\[
I(\text{size}_\tau)(x) := |x| \text{ if } x \neq \bot_{\text{Set}(\tau)} \text{ and } \bot_{\text{Int}} \text{ otherwise}
\]
(vi) Putting It All Together

Valuations of Logical Variables

- **Recall**: we have typed logical variables \( (w \in W) \), \( \tau(w) \) is the type of \( w \).

- By \( \beta \), we denote a valuation of the logical variables, i.e. for each \( w \in W \),

\[
\beta(w) \in I(\tau(w)).
\]

\[
\beta; W \rightarrow \bigcup_{\tau(w) \in \mathcal{T}} I(\tau(w))
\]

\[
\beta(w) \in I(\tau(w)) = \mathcal{D}(C) \cup \{ 0 \}
\]
(vi) Putting It All Together...

\[
\text{expr ::= } w \mid \omega(\text{expr}_1, \ldots, \text{expr}_n) \mid \text{allInstances}_C \mid v(\text{expr}_1) \mid r_1(\text{expr}_1) \\
\mid r_2(\text{expr}_1) \mid \text{expr}_1 \rightarrow \text{iterate}(v_1 : \tau_1 ; v_2 : \tau_2 = \text{expr}_2 \mid \text{expr}_3)
\]

- \(I[w](\sigma, \beta) := \beta(w)\)
- \(I[\omega(\text{expr}_1, \ldots, \text{expr}_n)](\sigma, \beta) := I(\omega)(I[\text{expr}_1](\sigma, \beta), \ldots, I[\text{expr}_n](\sigma, \beta))\)
- \(I[\text{allInstances}_C](\sigma, \beta) := \text{dom}(\sigma) \cap \mathbb{D}(C)\)

Note: in the OCL standard, \(\text{dom}(\sigma)\) is assumed to be finite.

Again: doesn’t scare us.

- \(I[v(\text{expr}_1)](\sigma, \beta) := \begin{cases} \sigma(u)(v), & \text{if } u \in \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases}\)
- \(I[r_1(\text{expr}_1)](\sigma, \beta) := \begin{cases} u, & \text{if } u \in \text{dom}(\sigma) \text{ and } \sigma(u)(r_1) = \{u\} \\ \bot, & \text{otherwise} \end{cases}\)
- \(I[r_2(\text{expr}_1)](\sigma, \beta) := \begin{cases} \sigma(u)\{v_2\}, & \text{if } u \notin \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases}\)

(Recall: \(\sigma\) evaluates \(r_2\) of type \(C_\ast\) to a set)
(vi) Putting It All Together...

\[
expr ::= w \mid \omega(expr_1, \ldots, expr_n) \mid \text{allInstances}_C \mid v(expr_1) \mid r_1(expr_1) \mid r_2(expr_1) \mid expr_1 \rightarrow \text{iterate}(v_1 : \tau_1; v_2 : \tau_2 = expr_2 \mid expr_3)
\]

\[I[expr_1 \rightarrow \text{iterate}(v_1 : \tau_1; v_2 : \tau_2 = expr_2 \mid expr_3)](\sigma, \beta) := \begin{cases} I[expr_3](\sigma, \beta), & \text{if } I[expr_1](\sigma, \beta) = \emptyset \\ \text{iterate}(hlp, v_1, v_2, expr_3, \sigma, \beta'), & \text{otherwise} \end{cases}
\]

where \(\beta' = \beta[hlp \mapsto I[expr_1](\sigma, \beta), v_2 \mapsto I[expr_3](\sigma, \beta)]\) and

\[I[\text{iterate}(hlp, v_1, v_2, expr_3, \sigma, \beta')](\sigma, \beta'') := \begin{cases} I[expr_3](\sigma, \beta'[v_1 \mapsto x]), & \text{if } \beta'(hlp) = \{x\} \\ I[expr_3](\sigma, \beta''), & \text{if } \beta'(hlp) = X \cup \{x\} \text{ and } X \neq \emptyset \end{cases}
\]

where \(\beta'' = \beta'[v_1 \mapsto x, v_2 \mapsto \text{iterate}(hlp, v_1, v_2, expr_3, \sigma, \beta'[hlp \mapsto X])]\)

Quiz: Is (our) \(I\) a function?
Where Are We?

You Are Here.

UML

Model

Instances

Mathematics

CD, SM

φ ∈ OCL

expr

CD, SD

\( \mathcal{P} = (\mathcal{F}, \mathcal{E}, V, \text{attr}) \), SM

\( M = (\Sigma_{\mathcal{P}}, A_{\mathcal{P}}, \neg SM) \)

\( \pi = (\sigma_0, \varepsilon_0, \text{cons}_0, \text{Snd}_0) \xrightarrow{w_0} (\sigma_1, \varepsilon_1, \cdots) \xrightarrow{w_n} ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \)

G = (N, E, f)

OD

UML

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

\( \mathcal{G} \subset \mathcal{OD} \)

\( \mathcal{G} \subset \mathcal{OD} \)

\( \mathcal{G} \subset \mathcal{OD} \)

\( \mathcal{G} \subset \mathcal{OD} \)
Definition. A node labelled graph is a triple

\[ G = (N, E, f) \]

consisting of

- vertexes \( N \),
- edges \( E \),
- node labeling \( f : N \to X \), where \( X \) is some label domain,
**Object Diagrams**

**Definition.** Let $\mathcal{D}$ be a structure of signature $\mathcal{D} = (\mathcal{E}, \mathcal{C}, V, \text{atr})$ and $\sigma \in \Sigma_\mathcal{D}$ a system state.

Then any graph $G = (N, E, f)$ with
- nodes are identities (not necessarily alive), i.e. $N \subseteq \mathcal{D}(\mathcal{E})$ finite,
- edges correspond to "links" of objects, i.e. $E \subseteq N \times \{ v : \tau \in V \mid \tau \in \{ C_0, 1, C^* \mid C \in \mathcal{C} \} \} \times N$,
- objects are labelled with attribute valuations and non-alive identities marked with "X", i.e. $X = \{ X \} \cup \{ V \not\rightarrow (\mathcal{D}(\mathcal{C}) \cup \mathcal{D}(\mathcal{B})) \}$

is called **object diagram** of $\sigma$. 

\[
\begin{align*}
N &= \{ 1_{34}, 3_{34}, 5_{34} \} \\
E &= \{ (3_{34}, \text{participants}, 1_{34}), \\
& \quad (3_{34}, \text{participants}, 5_{34}) \} \\
\text{f} &= \{ 1_{34} \rightarrow \{ \text{age} = 21 \}, \\
& \quad 3_{34} \rightarrow \{ \text{participants} = \{ 1_{34}, 8_{34}, 5_{34} \}, \text{married} = \{ 3_{34} \} \} \}
\end{align*}
\]
**Graphical Representation of Object Diagrams**

- Assume $\mathcal{P} = \{\{\text{Int}\}, \{C\}, \{v_1 : \text{Int}, v_2 : \text{Int}, r : \mathcal{C}\}, \{C \mapsto \{v_1, v_2, r\}\}\}$.  
- Consider $\sigma = \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2, r \mapsto \emptyset\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4, r \mapsto \emptyset\}\}$.  
- Then $G = (N, E, f) = (\{u_1, u_2\}, \{(u_1, r, u_2)\}, \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4\}\})$, is an object diagram of $\sigma$ wrt. $\mathcal{P}$ and any $\mathcal{P}$ with $\mathcal{P} (\text{Int}) \supseteq \{1, 2, 3, 4\}$.  
- We may equivalently (!) represent $G$ graphically as follows:

---

**UML Notation for Object Diagrams**

- Use this if $f(u) = \emptyset$ or $f(u) = X$
- "compartment"
- mandatory
- optional
- "class"
- optional
- different "boxes" represent different objects
- different "boxes" represent different objects
Object Diagrams: More Examples

\[ N \subset \mathcal{P}(\mathcal{C}) \text{ finite, } E \subset N \times V_{0,1,\ldots} \times N, \quad X = \{X\} \cup (V \rightarrow (\mathcal{P}(\mathcal{F}) \cup \mathcal{P}(\mathcal{C}))) \]
\[ u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r), \quad f(u) \subseteq \sigma(u) \text{ or } f(u) = \{X\} \]

\[ \sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5\}_C\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{x \mapsto 23\}\} \]

vs.

\[
\begin{array}{|c|c|}
\hline
N & 5C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\} \\
\hline
n & 5C \mapsto \emptyset \\
\hline
p & 5C \mapsto \emptyset \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
1C & p = \emptyset \\
\hline
n & 1C \mapsto \emptyset \\
\hline
p & 1C \mapsto \emptyset \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
1D & x = 23 \\
\hline
\end{array}
\]

Complete vs. Partial Object Diagram

Definition. Let \( G = (N, E, f) \) be an object diagram of system state \( \sigma \in \Sigma_{G} \).

We call \( G \) complete wrt. \( \sigma \) if and only if

- \( G \) is object complete, i.e.
  - \( G \) consists of all alive objects, i.e. \( N = \text{dom}(\sigma) \),

- \( G \) is attribute complete, i.e.
  - \( G \) comprises all "links" between alive objects, i.e. if \( u_2 \in \sigma(u_1)(r) \) for some \( u_1, u_2 \in \text{dom}(\sigma) \) and \( r \in V \), then \( (u_1, r, u_2) \in E \), and
  - each node is labelled with the values of all \( \mathcal{F} \)-typed attributes, i.e. for each \( u \in \text{dom}(\sigma) \),
    \[
    f(u) \supseteq \sigma(u) \upharpoonright V_{\mathcal{F}} \cup \{r \mapsto (\sigma(u)(r) \setminus N) \mid r \in V : \sigma(u)(r) \setminus N \neq \emptyset\}
    \]
    where \( V_{\mathcal{F}} := \{v : \tau \in V \mid \tau \in \mathcal{F}\} \).

Otherwise we call \( G \) partial.
Complete vs. Partial Examples

- $N = \text{dom}(\sigma)$, if $u_2 \in \sigma(u_1)(r)$, then $(u_1, r, u_2) \in E$,
- $f(u) = \sigma(u)|_{V_F} \cup \{r \mapsto (\sigma(u)(r) \setminus N) \cup \sigma(u)(r) \setminus N\}$

Complete or partial?

$\sigma = \{1_C \mapsto \{p \mapsto 0, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto 0, n \mapsto \emptyset\}, 1_D \mapsto \{x \mapsto 23\}\}$

Complete/Partial is Relative

- **Claim:**
  - Each finite system state has exactly one complete object diagram.
  - A finite system state can have many partial object diagrams.

- Each object diagram $G$ represents a set of system states, namely
  
  $G^{-1} := \{\sigma \in \Sigma_\mathcal{P} \mid G \text{ is an object diagram of } \sigma\}$

- **Observation:** If somebody tells us, that a given (consistent) object diagram $G$ is complete, we can uniquely reconstruct the corresponding system state.
  
  In other words: $G^{-1}$ is then a singleton.
**Corner Cases**

**Closed Object Diagrams vs. Dangling References**

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

![Diagram](image)

**Definition.** Let $\sigma$ be a system state. We say attribute $v \in V_{0,1,\sigma}$ 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 $\mathcal{F}$-typed attribute/value pairs only.
Special Notation

- $\mathcal{S} = (\{\text{Int}\}, \{C\}, \{n, p : C_*\}, \{C \mapsto \{n, p\}\})$.

- Instead of

  ```
  \[ \begin{array}{c}
  \text{1c : } C \\
  \text{n}
  \end{array} \quad \begin{array}{c}
  \text{2c : } C \\
  \text{n}
  \end{array} \]
  ```

  we want to write

  ```
  \[ \begin{array}{c}
  \text{1c : } C \\
  \text{n} \\
  \text{p = 0}
  \end{array} \quad \begin{array}{c}
  \text{2c : } C \\
  \text{n} \\
  \text{p = 0}
  \end{array} \]
  ```

  or

  ```
  \[ \begin{array}{c}
  \text{p} \\
  \text{1c : } C \\
  \text{n}
  \end{array} \quad \begin{array}{c}
  \text{p} \\
  \text{2c : } C \\
  \text{n}
  \end{array} \]
  ```

  to explicitly indicate that attribute $p : C_*$ has value $\emptyset$ (also for $p : C_{0,1}$).

Aftermath

We slightly deviate from the standard (for reasons):
- In the course, $C_{0,1}$ and $C_*$-typed attributes only have sets as values. UML also considers multisets, that is, they can have

  ```
  \[ \begin{array}{c}
  \text{y_1 : } C \\
  \text{n}
  \end{array} \quad \begin{array}{c}
  \text{y_2 : } C \\
  \text{n}
  \end{array} \]
  ```

  (This is not an object diagram in the sense of our definition because of the requirement on the edges $E$. Extension is straightforward but tedious.)

- We allow to give the valuation of $C_{0,1}$- or $C_*$-typed attributes in the values compartment.
  - Allows us to indicate that a certain $r$ 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 $\emptyset$ values.
References


