Software Design, Modelling and Analysis in UML
Lecture 5: Object Diagrams

2015-11-05
Prof. Dr. Andreas Podelski, Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:
• OCL Semantics

This Lecture:
• Educational Objectives:
  • 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

OCL Satisfaction Relation

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 $\sigma \in \Sigma_{DS}$ a system state.
We write
- $\sigma \models \phi$ if and only if $\llbracket \phi \rrbracket (\sigma, \emptyset) = \text{true}$.
- $\sigma \not\models \phi$ if and only if $\llbracket \phi \rrbracket (\sigma, \emptyset) = \text{false}$.

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

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 $S$ wrt. $D$ which satisfies all of them, i.e. if $\exists \sigma \in \Sigma_{DS}: \sigma \models \phi_1 \land \ldots \land \sigma \models \phi_n$ and inconsistent (or unsatisfiable) otherwise.

Example: OCL Consistent?

TeamMember
name : String
age : Integer

Location
participants 2..* meetings

Meeting
title : String
numParticipants : Integer
start : Date
duration: Time

move(newStart : Date)

• context Location inv: name = 'Lobby' implies meeting->isEmpty()
• context Meeting inv: title = 'Reception' implies location.name = 'Lobby'
• allInstances Meeting->exists (w: Meeting| w.title = 'Reception')
Deciding OCL Consistency

• Whether a set of OCL constraints is consistent 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.

OCL is as expressive as first-order logic over integers.

And now? Options:

Cabot and Claris´ o (2008)

• Constrain OCL, use a less rich fragment of OCL.

• Revert to finite domains — 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 M¨ uller (2003). (Or: sequence diagrams.)

• Real-Time: "Objects respond within 10s"

Proposals for fixes e.g. Cengarle and Knapp (2002).

• Reachability: "After insert operation, node shall be reachable."

Fix: add transitive closure.

What Is OCL Good For?
What's It Good For?

- Most prominent: Formalise requirements supposed to be satisfied by all system states. Example: "the choice panels of a VM should be consistent"

- Not unknown: Formalise pre/post-conditions of methods (Behavioural Features). Then evaluated over two system states (before/after executing the method). Example: "the dispense water method should decrement win"

- Common with State Machines: Guards in transitions.

- Lesser known: Specify operation bodies.

Mathematics

Object Diagrams

Recall: Graph

Definition. A node-labelled graph is a triple \( G = (N, E, f) \) consisting of

- nodes are identities (not necessarily alive), i.e. \( N \subseteq D(\text{C}) \)\( \text{finite} \),
- edges correspond to "links" of objects, i.e. \( E \subseteq N \times \{ v : T \in V | T \in \{ C_0, 1, C^* \} \} \)
- objects are labelled with attribute valuations, and non-alive identities with "X", i.e. \( X = \{ X \} \cup \{ V \rightarrow (D(T) \cup D(C^*)) \} \)

is called object diagram of \( \sigma \).
The empty set $\emptyset$ has value $\{\}$ explicitly.

Each (consistent) object diagram represents a set of system states, namely $X\mapsto\{\}$. The object complete $G$ and attribute complete $T_v$ are relative.

If somebody wants to write $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ node labelling $\sigma_i \in u$, $\forall i \in \{1, 2, \ldots, n\}$, $\exists X \in G, X \mapsto \{\}$.

We call $\langle G, T_x, T_v \rangle$ as an object diagram of system state with $\Sigma = (\{1, 2, \ldots, n\} \cup \{D\})$ if $\sigma(w) \in D$, $\forall w \in \sigma$. It consists of all alive and "linked" non-alive objects, i.e. for each $D \in u$, $\forall v \in V \in T_x$, $\sigma(D, v) \in \{\}$.

With $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, the object $\Sigma$ is finite.
Discussion

We slightly deviate from the standard (for reasons):

• We allow to show non-alive objects.

• 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.

• Easier to distinguish partial and complete object diagrams.

In the course, $C_0, C_1$ and $C_\ast$-typed attributes only have sets as values.

UML also considers multisets, that is, they can have $u_1 : C, u_2 : C$.

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.

From Object Diagram to Signature / Structure

• If we only have a diagram like $C_1 : C, C_2 : C, D : D$ we typically assume that it is meant to be an object diagram wrt. some signature and structure.

• In the example, we conclude that the author is referring to some signature $S = (T, C, V, atr)$ with at least:

  • $\{C, D\} \subseteq C$,
  • $T \in T$,
  • $\{x : C\ast, p : C\ast, z : T\} \subseteq V$,
  • $\{x\} \subseteq atr(C)$,
  • $\{p, z\} \subseteq atr(D)$,
  • $0 \in D(T)$.

Example: Object Diagrams for Documentation
Example: Data Structure

(Schumann et al., 2008)

BaseNode

+ parent : BaseNode *
+ prevSibling : BaseNode *
+ nextSibling : BaseNode *
+ firstChild : BaseNode *
+ lastChild : BaseNode *

Node

+ data : T
+ Node( data : T )

Iterator

+ operator ++() : Iterator
+ operator −−() : Iterator
+ operator ∗() : BaseNode

Forest

+ appendTopLevel( data: T )
+ appendChild( parent : Iterator, data : T )
+ remove( it : Iterator )
+ depth( it : Iterator ) : int
+ end() : Iterator
+ begin() : Iterator
+ empty() : bool
+ size() : int

References


