OCL: Bridging the Gap between Semi-Formal and Formal Specification

Achim D. Brucker
Albert-Ludwigs Universität Freiburg, Germany

September 30, 2002

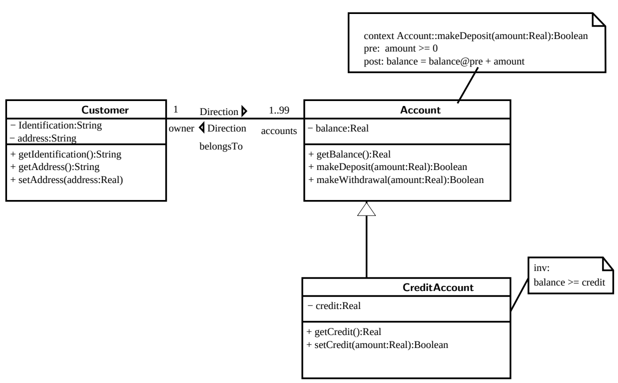
Motivation

- Why specify?
  - Complex software systems require a precise specification of architecture and components.
  - Semi-formal methods (like UML) are not strong enough.
- Why UML/OCL?
  - UML is the standard modeling language in OO development.
  - OCL is part of the OMG UML standard.

Specification should not only generate documentation!

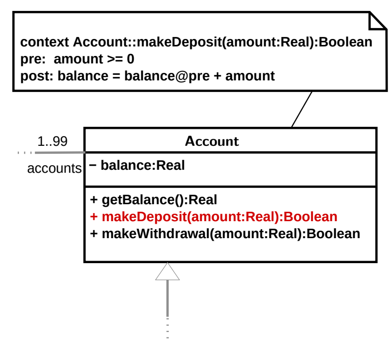
The Unified Modeling Language (UML)

- diagrammatic OO modeling language
- many diagram types, e.g.
  - class diagrams (static)
  - state charts (dynamic)
  - use cases
- semantics currently standardized by the OMG
- we expect wide use in SE-Tools (ArgoUML, Rational Rose,...)



The Object Constraint Language (OCL)

- designed for annotating UML diagrams (and give foundation for injectivities, ...)
- based on logic and set theory
- in the context of class–diagrams:
  - preconditions
  - postconditions
  - invariants
- can also be used for other diagram types



Why There is a Need for a "more" Formal UML

- **The short answer:**
  - UML is not powerful enough for supporting formal reasoning over specifications.

Softech Freiburg Motivation 4

Why There is a Need for a "more" Formal UML

- **The short answer:**
  - UML is not powerful enough for supporting formal reasoning over specifications.
  - OCL should close this gap.
- **The long answer:**

Softech Freiburg Motivation 4

Why There is a Need for a "more" Formal UML

- **The short answer:**
  - UML is not powerful enough for supporting formal reasoning over specifications.
  - OCL should close this gap.
- **The long answer:**
  - We want to be able to
    - verify
    - validate
    - refine

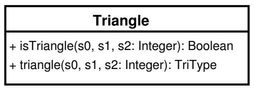
    UML/OCL specifications, e.g. for **proving security constraints** or **automatic test data generation**.
  - The OCL semantics is not formally defined and needs clarification of several issues.

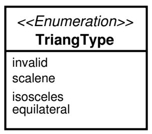
HOL-OCL: A Shallow Embedding of OCL into HOL

- is build on top of Isabelle/HOL.
- provides a consistent (machine checked) OCL semantics.
- allows the examination of OCL features.
- builds the basis for OCL tool development.
- follows OCL 1.4 and the proposal for OCL 2.0

HOL-OCL Application: Test Data Generation

Based on a UML/OCL specification a minimal set of test data is calculated which can be used for validating an implementation.

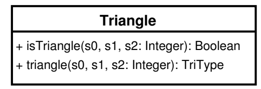


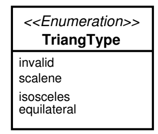



```
context
 Triangle::isTriangle(s0,s1,s2:Integer):Boolean

pre:
 (s0 > 0) and (s1 > 0) and (s2 > 0)

post:
 result =     (s2 < (s0 + s1))
          and (s0 < (s1 + s2))
          and (s1 < (s0 + s2))
```


HOL-OCL Application: Test Data Generation

Based on a UML/OCL specification a minimal set of test data is calculated which can be used for validating an implementation.






```
context
 Triangle::triangle(s0,s1,s2:Integer):TriangType

pre:
 (s0 > 0) and (s1 > 0) and (s2 > 0)

post:
 result = if (isTriangle(s0,s1,s2)) then
            if (s0 = s1) then
               if (s1 = s2) then
                 Equilateral::TriangType
               else
                 Isosceles::TriangType endif
            else
              if (s1 = s2) then
                Isosceles::TriangType
              else
                if (s0 = s2) then
                  Isosceles::TriangType
                else
                  Scalene::TriangType
              endif endif endif
          else
            Invalid::TriangType endif
```


HOL-OCL Application: Test Data Generation

1. Reduce all logical operation to the basis operators:

   and, **or**, und **not**

2. Determine disjunctive normal Form (DNF):

   x and $(y$ or $z)$ $\leadsto$ $(x$ and $y)$ or $(x$ and $z)$

3. Eliminate unsatisfiable sub-formulae, e.g.:

   scalene and invalid

4. Select test data with respect to boundary cases.

Partitioning of the Test Data

1. Input describes **no** triangle.
2. Input describes an **equilateral** triangle.
3. Input describes an **isoscalene** triangle:
   (a) with s_0 equals s_1.
   (b) with s_0 equals s_2.
   (c) with s_1 equals s_2.
4. Input describes an **scalene** triangle.

For each partition, concrete test data has to be selected with respect to boundary cases (e.g. max./min. Integers, ...).

Conclusion

- OCL can be seen as formal specification language.
- OCL can be used for further tool support, e.g.:
  - run-time checking, validating or proving (security) properties.
  - automatic test data generation.
  - reasoning over specifications.
- OCL offers a possibility for stepwise introducing Formal Methods into UML based, industrial software development processes.