Partial Evaluation of OCL

Daniel Larsson

KeY Workshop

Königswinter/Koblenz, June 2004

Overview

- Motivation
  - Automatically generated specifications
  - Pattern-driven generation of specifications
  - Need for simplification
- Partial Evaluation
- Example
- Implementation
- Results and future work

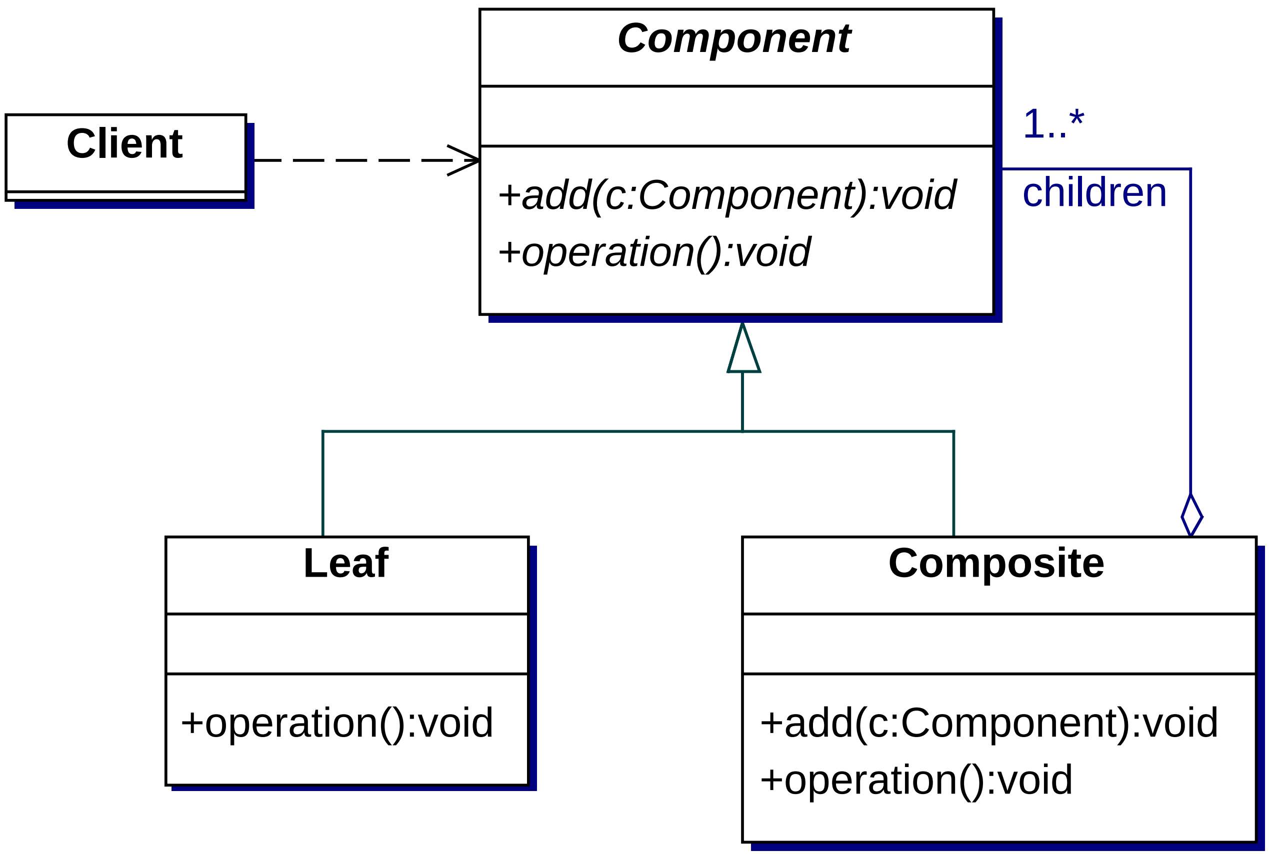
Automatically Generated Specs

- **Goal:** Make people use formal methods in software development
- **Problem:** Not trivial to write useful formal specifications
- **Solution:** Automatically generated specifications
  - *Ideally:*
    Informal specification
    $\Rightarrow$ Formal specification
  - *More realistic:*
    Informal specification
    $\Rightarrow$ Design pattern
    $\Rightarrow$ Formal specification
- Generated specifications need to be simplified
  $\Rightarrow$ Partial Evaluation

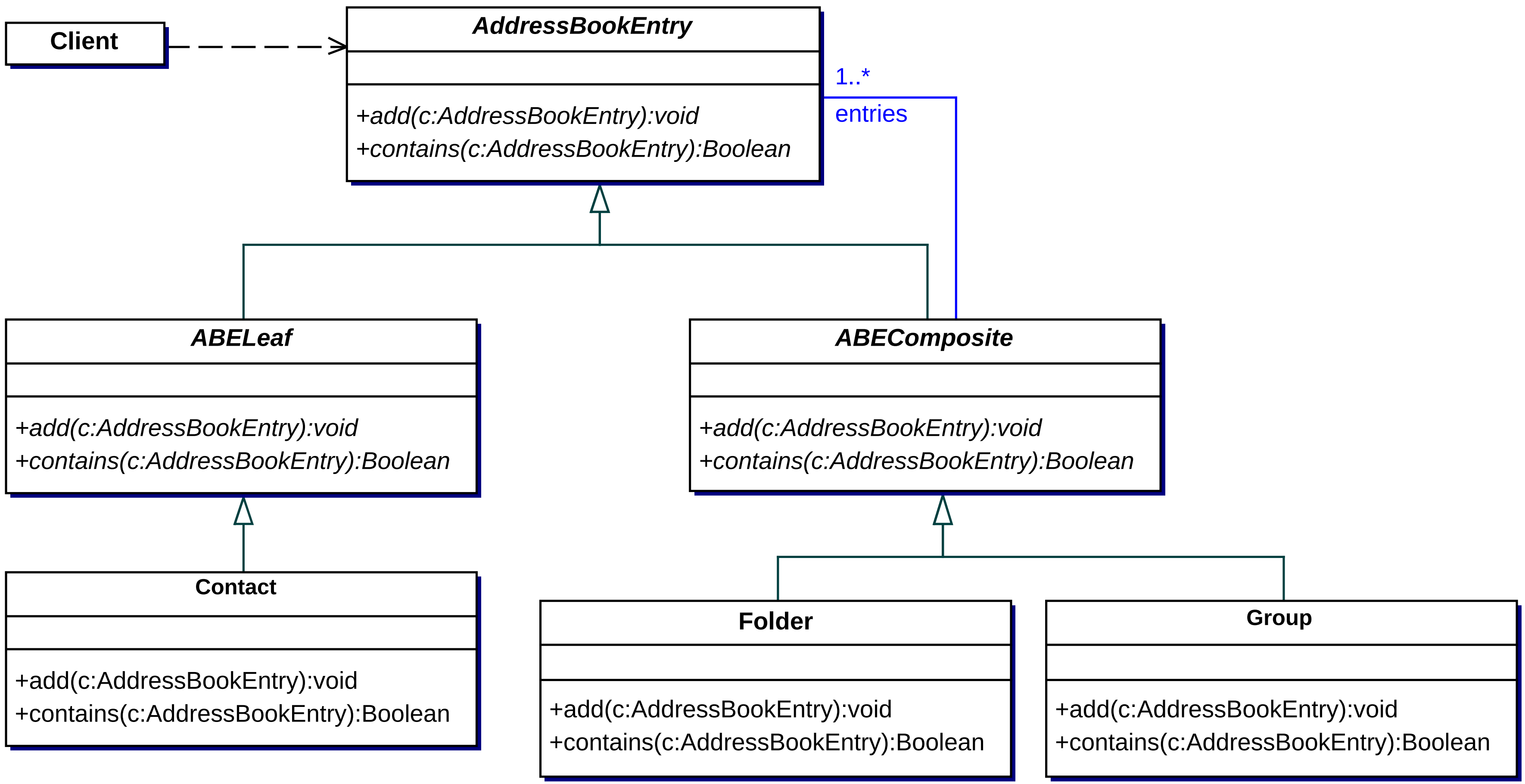
Generating Specs for Patterns

- Capture typical requirements associated with the pattern
- Don’t know beforehand ...
  - the *namespaces* of the modeled domains
  - what *structural modifications* the developer will perform
  - what *flavor* of the pattern the developer wants to use

Composite Pattern



Instantiation of Composite



Schema for (part of) Composite

```
schema childrenSetOrBag(String flavor)
 ocl: context Composite inv:
        if flavor = 'set'
         then self.children->size
               = self.children->asSet->size
         else true
        endif
```

Schema for Composite cont'd

```
schema childrenSetOrBag(String flavor)
 ocl: Composite.allSubtypes->
        forAll(s | s.allInstances->
         forAll(i |
            if flavor = 'set'
             then i.children->size
                   = i.children->asSet->size
             else true
            endif))
```

Generated Specification

```
ABEComposite.allSubtypes->
 forAll(s | s.allInstances->
  forAll(i |
     if 'set' = 'set'
      then i.entries->size
              = i.entries->asSet->size
      else true
     endif))
```

Simplification Needed

- Schema becomes parameterized
  - Elements from pattern's namespace
  - Different flavors of pattern — explicit parameters
- Structural modifications have to be taken into account
- Generated specification contains redundant information
- $\Rightarrow$ Need for simplification
- $\Rightarrow$ Partial Evaluation

Partial Evaluation

- Normally applied to computer programs
- Given a program and *some* of its input
  $\Rightarrow$ Produce a more specialized program
- Motivation w.r.t. programs: execution speedup
- Motivation w.r.t. formal specifications:
  - enhance understandability
  - make it easier to prove properties about them
- So far — just simplification
- Idea — apply more sophisticated p.e. techniques

Generated Specification — again

```
ABEComposite.allSubtypes->
 forAll(s | s.allInstances->
  forAll(i |
     if 'set' = 'set'
      then i.entries->size
              = i.entries->asSet->size
      else true
     endif))
```

OCL Simplification

```
ABEComposite.allSubtypes->
 forAll(s | s.allInstances->
  forAll(i |
     if true
      then i.entries->size
               = i.entries->asSet->size
      else true
     endif))
```

OCL Simplification cont'd

```
ABEComposite.allSubtypes->
 forAll(s | s.allInstances->
  forAll(i | i.entries->size
              = i.entries->asSet->size))
```

OCL Simplification cont'd

```
Set{Group, Folder, ABEComposite}->
 forAll(s | s.allInstances->
  forAll(i | i.entries->size
               = i.entries->asSet->size))
```

OCL Simplification cont'd

```
Group.allInstances->
   forAll(i | i.entries->size
               = i.entries->asSet->size)
and
Folder.allInstances->
   forAll(i | i.entries->size
               = i.entries->asSet->size)
and
ABEComposite.allInstances->
   forAll(i | i.entries->size
               = i.entries->asSet->size)
```

OCL Simplification cont'd

```
Group.allInstances->
   forAll(i | i.entries->size
                  = i.entries->asSet->size)
and
Folder.allInstances->
   forAll(i | i.entries->size
                  = i.entries->asSet->size)
```

OCL Simplification cont'd

```
context Group inv:
  self.entries->size
   = self.entries->asSet->size

context Folder inv:
  self.entries->size
   = self.entries->asSet->size
```

OCL Simplification cont'd

```
context Group inv:
  entries->size
   = entries->asSet->size

context Folder inv:
  entries->size
   = entries->asSet->size
```

Implementation

- Already have a rule-engine
- Taclet machinery!
- Re-write taclets

OCL Taclets

```
ocl_equals {find(e = e) replacewith(true)}

ocl_if_true {
    find(if true then e1 else e2 endif)
    replacewith(e1)}

ocl_allsubtypes {find(c.allSubtypes)
                    replacewith(#allsubtypes(c))}
```

Recipe

1. Express OCL using Term datastructure
2. Wrap the “term” in a formula
3. Put formula in sequent (succedent)
4. Apply taclets to sequent

Results and Future Work

- Results
  - Know how to express OCL using Term datastructure
  - Can handle bound variables
  - Have performed evaluation steps in example
- Future work
  - Deal with types
  - Write the taclets
  - More partial evaluation techniques to be evaluated
  - Connection to OCL parser/type checker
  - Integration with pattern mechanism in KeY