

Lecture 9

UML Model architecture Object Constraint Language

12/10/98

AOO / UML / OCL/Strategies

1

UML language architecture

- UML metamodel defines meaning of UML models
- Defined in a metacircular manner, using a subset of UML to specify itself
- UML metamodel bootstraps itself. Similar:
 - compiler compiles itself
 - grammar defines itself
 - class dictionary defines itself

12/10/98

AOO / UML / OCL/Strategies

2

4 layer metamodel architecture

- UML metamodel one of the layers
- Why four layers?
- Proven architecture for complex models
- Validates core constructs by using them to define themselves

12/10/98

AOO / UML / OCL/Strategies

3

Four layer architecture

- meta-metamodel
 - language for specifying metamodels
- metamodel
 - language for specifying models
- model
 - language for specifying objects in some domain
- user objects

12/10/98

AOO / UML / OCL/Strategies

4

Four levels

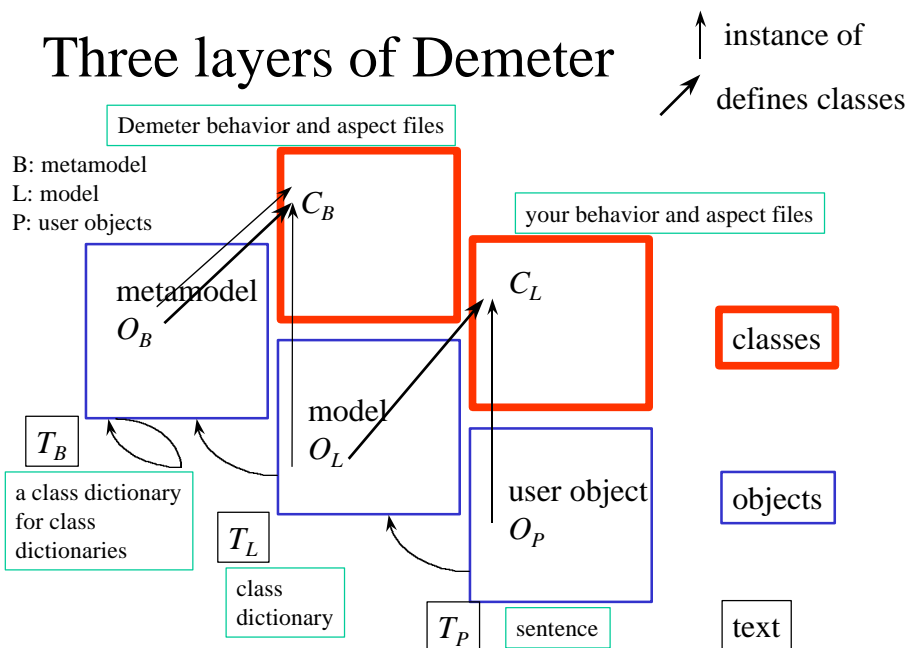
- User Objects in running system
 - check run-time constraints
- Model of System under design
 - specify run-time constraints
- Meta-model
 - specify constraints on use of constructs in model
- Meta-metamodel
 - data interchange between modeling tools

12/10/98

AOO / UML / OCL/Strategies

5

Three layers of Demeter



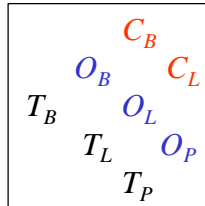
12/10/98

AOO / UML / OCL/Strategies

6

Icon

Demeter Tiling



Use as reminder for
Demeter Tiling.

UML OCL

- Object Constraint Language
 - allows you to define side effect-free constraints for UML and other models (for example adaptive programs)
 - used in UML to defined well-formedness rules of the UML meta model (invariants for meta model classes)

Why OCL

- It is a formal mathematical language
- Tend to be hard to use by average modelers
- OCL is intended for average modelers
- Developed as business modeling language within IBM insurance division (has roots in Syntropy method)
- OCL is a pure expression language (side effect free)

12/10/98

AOO / UML / OCL/Strategies

9

Companies behind OCL

- Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam

12/10/98

AOO / UML / OCL/Strategies

10

Where to use OCL?

- Specify invariants for classes and types
- Specify pre- and post-conditions for methods
- As a navigation language

OCL properties

- LL(1) language
 - finally back to the Pascal days!
 - Grammar provided uses EBNF syntax
- Parser generated by JavaCC

What is OCL?

- Predicate calculus for objects
- Traditional predicate calculus:
 - individuals
 - variables, predicate and function symbols
 - terms (for all, Boolean connectives)
 - axioms and the theories they define (group theory, number theory, etc.)
- In OCL: individuals -> objects

Structured individuals

- some “structural” constraints imposed by UML class diagram; further constraints can be imposed by OCL expressions
- annotated UML class diagram defines textual representation

Connection to model

- Self. Each OCL expression is written in the context of an instance of a specific type.

Company

```
self.numberOfEmployees
```

c : Company

```
c.numberOfEmployees
```

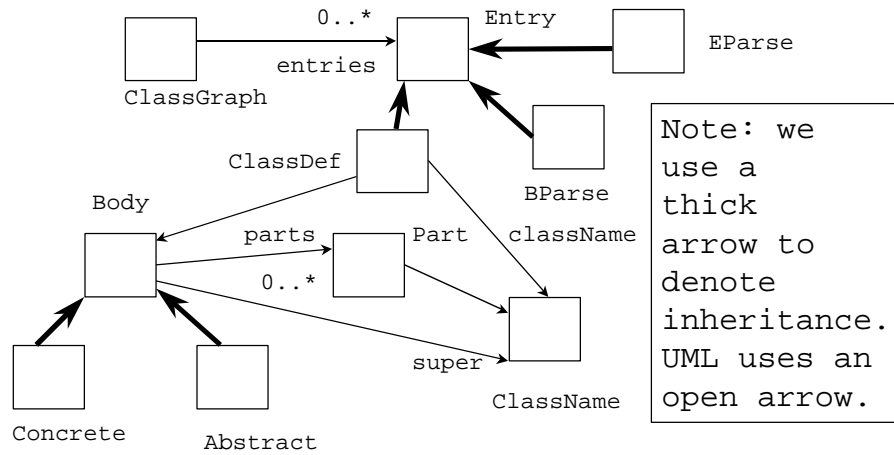
Connection to model

- Invariants of a type. An OCL expression stereotyped with <<invariant>>. An invariant must be true for all instances of the type at any time.

Person

```
<<invariant>> self.age >= 0
```

Example: UML class diagram ClassGraph

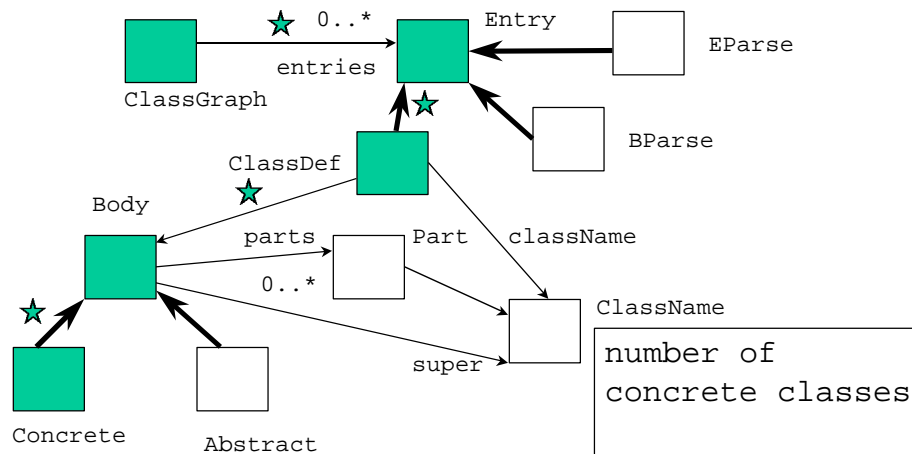


12/10/98

AOO / UML / OCL / Strategies

17

UML class diagram ClassGraph



12/10/98

AOO / UML / OCL / Strategies

18

Example

```
-> collection op  
select:subset  
collect:new set
```

- Number of concrete classes:

```
- ClassGraph self.entries->  
  select(c:Entry|c.  
    oclIsTypeOf(ClassDef))->  
    collect(body)->  
      select (b:Body|b.  
        oclIsTypeOf(Concrete))  
->size
```

Pre- and post-conditions

- constraints stereotyped with
<<precondition>> and <<postcondition>>

for an operation or method. Example:

```
Type::op(param1 : Type1 ...):
```

```
  Return Type
```

```
  <<pre>>: param1 ...
```

```
  <<post>>: result = ...
```

Pre- and post-conditions

- Example: Post condition for insert operation:

```
person.allInstances ->  
forAll(p1, p2 | p1 <> p2  
implies p1.id <> p2.id)
```

Basic values and types

- Boolean true, false
– and or xor not implies if-then-else
- Integer 1 2 3 subtype of Real
– * + - / abs
- Real 3.14
– * + - / floor
- String 'To be or not to be'
– toUpper concat

Basic values and types

- Collection
 - Set subtype of Collection
 - Sequence subtype of Collection
 - Bag subtype of Collection

if element types conform to each other

Types from the UML Model

- Each OCL expression lives in the context of a UML model, a number of types/classes and their features and associations and their generalizations.
- All types/classes from the UML model are types in OCL.

Type Conformance

- OCL is typed
- Type conformance rules for types in the class diagram are simple:
 - each type conforms to its supertype
 - type conformance is transitive

Objects and properties

- The value of a property on an object that is defined in a class diagram is specified by a dot followed by the property name.

Atype

`self.property`

Properties

- an attribute
- an association end
- an operation with *isQuery* true
- a method with *isQuery* true

Properties

- an attribute
`Person self.age >= 0`
`self.employer->size`
- an association end
`Company`
`self.manager --type Person`
`self.employee--type Set(Person)`

Properties

- an operation with *isQuery* true
Person self.income(aDate)
Company self.stockPrice()

Missing role names

- Whenever a role name is missing at one of the ends of an association, the name of the type at the association end, starting with a lowercase character is used as role name. If this results in an ambiguity, the role name is mandatory

Navigation over associations

- Company self.manager
object of type Person or Set(Person)
 - used as Set(Person)
self.manager->size -- result 1
 - used as Person
self.manager.age

OclType and OclAny

- All types in a UML model, or predefined within UML have a type. This type is an instance of the OCL type called OclType.
- OclType: meta type of all types. OclAny supertype of all types. OclType : Class = OclAny : Object (analogy to Java)
- Features of OclType: good for meta programming.

Predefined OCL types

- OclType: type: instance of OclType
 - type.name : String
 - type.attributes:Set (String)
 - type.associationEnds:Set (String)
 - type.operations:Set (String)
 - type.supertypes:Set (OclType)
 - type.allSupertypes:Set (OclType)
 - type.allInstances:Set (type)

12/10/98

AOO / UML / OCL/Strategies

33

Similarity: java.lang.Class

- instances of class Class represent classes and interfaces in a way that can be read (but not modified) by a running Java program

```
public final class Class{
    public String getName();
    public Class getSuperClass();
    public Class[] getInterfaces();
    . . .
```

12/10/98

AOO / UML / OCL/Strategies

34

Predefined OCL types

- OclAny: supertype of all types in the model. object: instance of OclAny
 - object=(object2:OclAny)
 - object<>(object2:OclAny):Boolean
 - object.oclType:OclType
 - object.oclIsKindOf(type:OclType): Boolean

Similarity: java.lang.Object

- All objects, including arrays, implement the methods of this class

```
public class Object {  
    public final Class getClass();  
    public boolean  
        equals(Object obj);  
    ...  
}
```

Predefined features on all objects (OclAny)

- Type of an object

```
oclType : OclType
```

Feature oclType results in type of an object

- Direct type

```
oclIsTypeOf(t:OclType):Boolean
```

- Direct or super type

```
oclIsKindOf(t:OclType):Boolean
```

Examples

- Person

```
self.oclType
```

results in Person

- Person

```
self.oclIsTypeOf(Person)--true
```

```
self.oclIsTypeOf(Company)--false
```

Predefined features on types

- Two kinds of properties
 - on instances of classes
 - on types/classes themselves
- Most important predefined feature on each type: `allInstances`

```
Person.allInstances ->  
forAll(p1, p2 | p1 <> p2  
implies p1.id <> p2.id)
```

Collections

- Navigation will most often result in a collection.
- `Collection` predefined
- Large number of predefined operations
- `Collection(X) :`
 - `Set(X) | Sequence(X) | Bag(X) .`
- Specifiable by a literal

Collection type conformance

- Collection(X) is a subtype of OclAny.
- Rules (only number 3 collection specific)
 - T1 conforms to T2 if T1=T2.
 - T1 conforms to T2 when T1 is a subtype of T2.
 - Collection(T1) conforms to Collection(T2) if T1 conforms to T2
 - conformance is transitive

Previous value in post-conditions

- Pre- and post-conditions on operations and methods
 - the value of a property at the start of the operation or method
 - the value of a property upon completion of the operation or method
 - `Person::birthdayHappens()`
 `post: age = age@pre + 1`

Collection Operations

- Select and reject operations
 - collection->select(boolean-expr)
Company
self.employee->select(age > 50)
 - collection->select
(v|boolean-expr-with-v)
Company self.employee->select
(p|p.age > 50)

12/10/98

AOO / UML / OCL/Strategies

43

Collection Operations

- Select and reject operations
 - collection->select
(v:Type|boolean-expr-with-v)
Company self.employee->select
(p:Person|p.age > 50)

12/10/98

AOO / UML / OCL/Strategies

44

Select syntax

- Define a subset

- collection->select

- (v:Type|boolean-expr-with-v)

type
redundancy

- collection->select

- (v|boolean-expr-with-v)

refer to
entire
object

- collection->select

- (boolean-expr)

refer to
parts

Reject syntax

- Define a subset

- collection->reject

- (v:Type|boolean-expr-with-v)

- collection->reject

- (v|boolean-expr-with-v)

- collection->reject

- (boolean-expr)

- Instead negate expression

Collect syntax

```
- collection->collect  
  (v:Type|expr-with-v)  
- collection->collect  
  (v|expr-with-v)  
- collection->collect  
  (expr)
```

Build new
collection
by applying
expression
to elements
of old
collection

- Creates a bag

```
self.employee->collect(bdate)->asSet
```

12/10/98

AOO / UML / OCL / Strategies

47

Shorthand for Collect

- Because navigation through many objects is very common, there is a shorthand notation for collect that makes OCL expressions more readable. Both are correct:

```
- self.employee ->  
  collect(birthdate.year)  
- self.employee.birthdate.year
```

- Violations of Law of Demeter!

12/10/98

AOO / UML / OCL / Strategies

48

ForAll operation

- All elements satisfy Boolean expression
 - `collection->forAll`
`(v:Type|boolean-expr-with-v)`
 - `collection->forAll`
`(v|boolean-expr-with-v)`
 - `collection->forAll`
`(boolean-expr)`

Exists operation

- At least one element satisfies Boolean expression
 - `collection->exists`
`(v:Type|boolean-expr-with-v)`
 - `collection->exists`
`(v|boolean-expr-with-v)`
 - `collection->exists`
`(boolean-expr)`

Predefined OCL types

- Integer, Real, String, Boolean
- OclExpression, OclType, OclAny
- OclType
 - all types defined in a model have type OclType
 - allows access to the meta-level of the model

Predefined OCL types

- OclType: type: instance of OclType
 - type.name : String
 - type.attributes:Set (String)
 - type.associationEnds:Set (String)
 - type.operations:Set (String)
 - type.supertypes:Set (OclType)
 - type.allSupertypes:Set (OclType)
 - type.allInstances:Set (type)

Predefined OCL types

- OclAny: supertype of all types in the model. object: instance of OclAny
 - object=(object2:OclAny)
 - object<>(object2:OclAny):Boolean
 - object.oclType:OclType
 - object.oclIsKindOf(type:OclType): Boolean

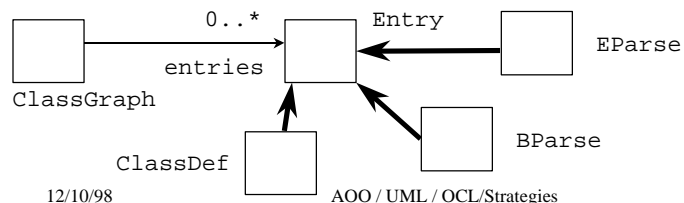
12/10/98

AOO/UML/OCL/Strategies

53

Applications

- Number of class definitions:
 - ClassGraph self.entries->size **wrong**
 - ClassGraph self.entries->
select(c:Entry|c.
oclIsTypeOf(ClassDef))->size



12/10/98

AOO/UML/OCL/Strategies

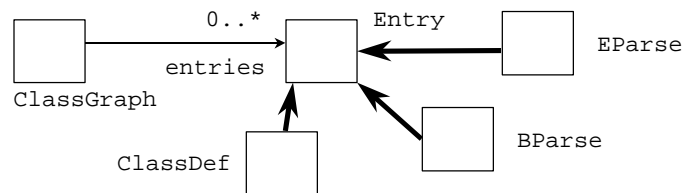
54

Applications



- Number of class definitions: What about using strategies to define collections?

```
- ClassGraph self.{to ClassDef}  
->size
```



12/10/98

AOO / UML / OCL / Strategies

55

Improve OCL: make adaptive

- OCL stresses the importance of collections
- Collections are best specified adaptively
- A strategy $SS = (S, B, s, t)$ with source s and target set t and name map N for class graph G defines a collection of objects contained in a $N(s)$ -object. The collection type CT is the union of $N(t_1)$ for t_1 in t .

12/10/98

AOO / UML / OCL / Strategies

56

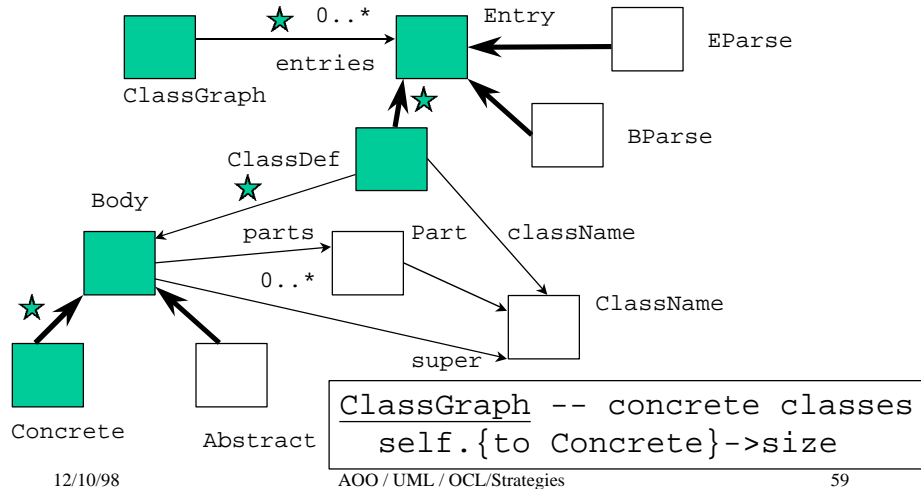
Improve OCL

- The collection consists of *CT*-objects reached during the traversal of the $N(s)$ object following strategy *SS*.

Properties

- In OCL
 - an attribute
 - an association end
 - an operation with *isQuery* true
 - a method with *isQuery* true
- Add for adaptive OCL
 - a strategy { ... } with a single source

UML class diagram ClassGraph



Applications

- Number of concrete classes:

```

- ClassGraph self.entries->
  select (c:Entry|c.
    oclIsTypeOf (ClassDef)) ->
  collect (body)->
  select (b:Body|b.
    oclIsTypeOf (Concrete))
->size

```

```

ClassGraph self.entries->
  select(c:Entry|c.
    oclIsTypeOf(ClassDef))->
    collect(body)->
      select (b:Body|b.
        oclIsTypeOf(Concrete))
->size -- count concrete classes

```

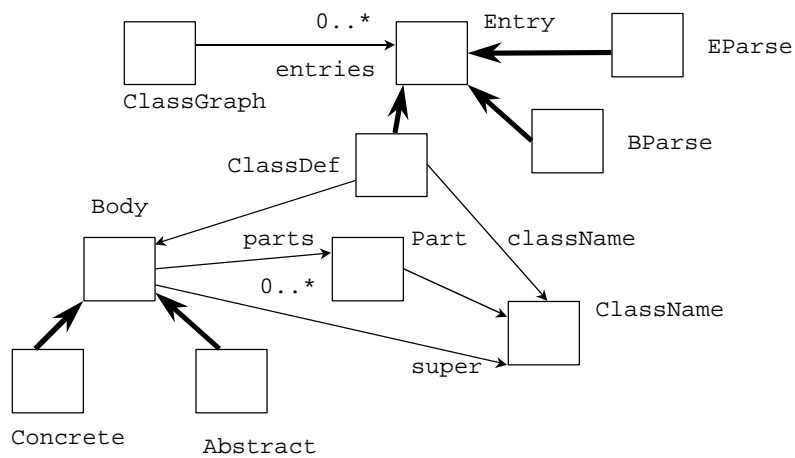
```

ClassGraph -- count concrete classes
self.{to Concrete}->size

```

Which one is easier to write?

UML class diagram ClassGraph



Applications

- Terminal buffer rule

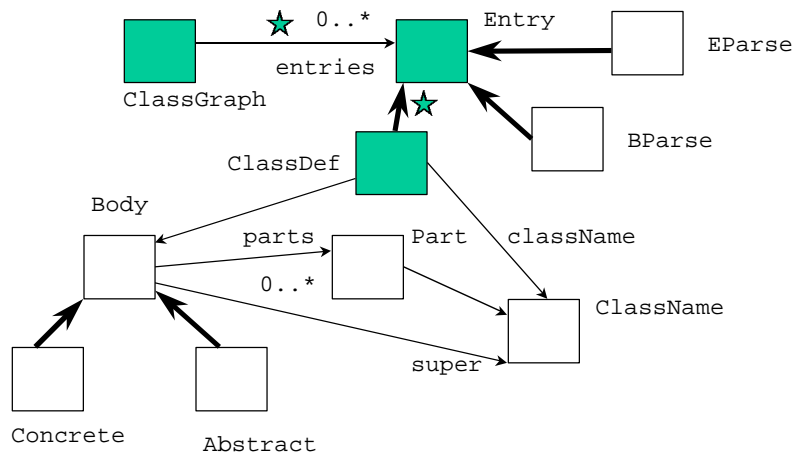
```
ClassGraph self.{to ClassDef}
  ->forall(r|r.termBProp())
ClassDef Boolean termBProp(){
  partCNS=self.{via Part to ClassName};
  result=if (partCNS->size)>1 then
    (partCNS->intersection(predefCNS))
    -> isEmpty
  else true endif}
```

12/10/98

AOO / UML / OCL / Strategies

63

UML class diagram ClassGraph

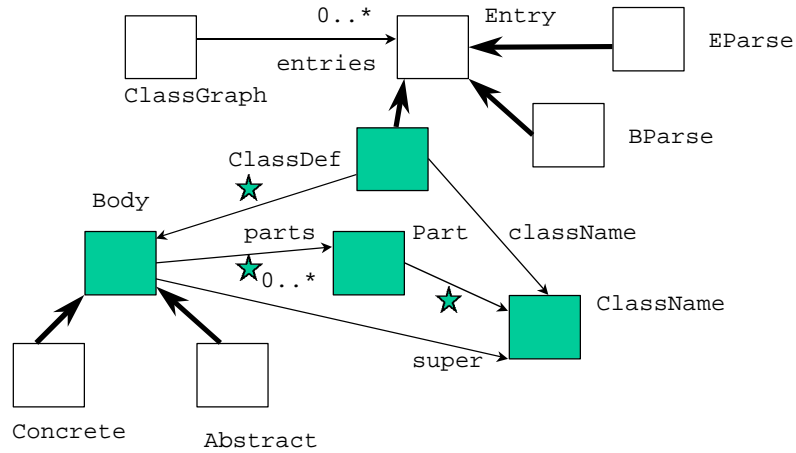


12/10/98

AOO / UML / OCL / Strategies

64

UML class diagram ClassGraph

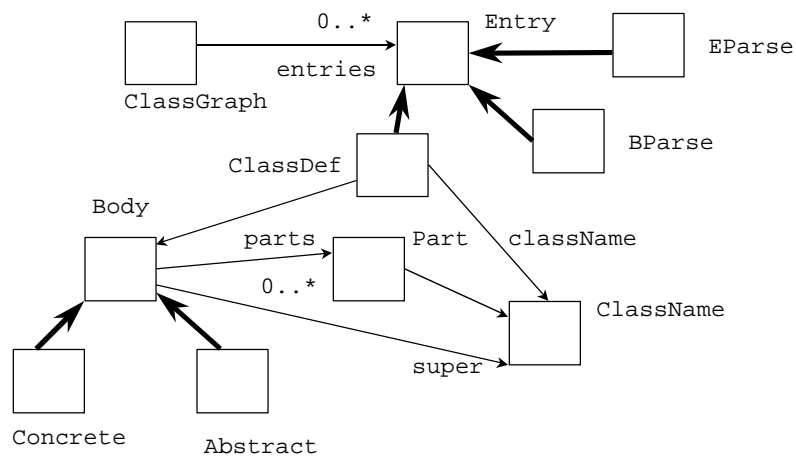


12/10/98

AOO / UML / OCL / Strategies

65

UML class diagram ClassGraph



12/10/98

AOO / UML / OCL / Strategies

66

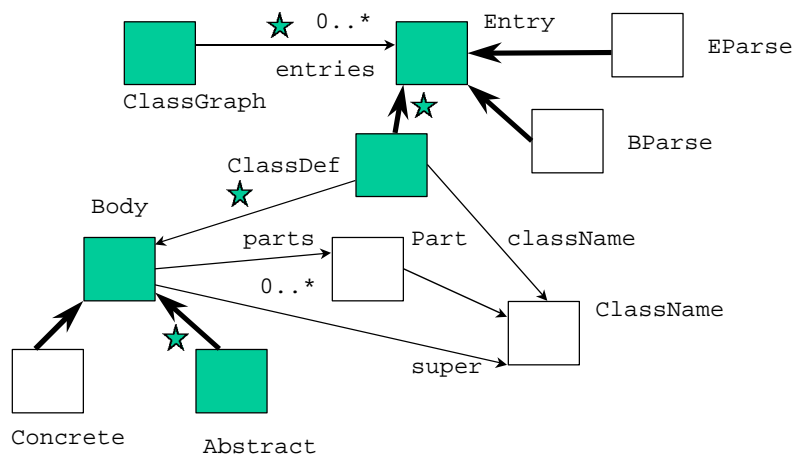
Applications

- Class graph is flat

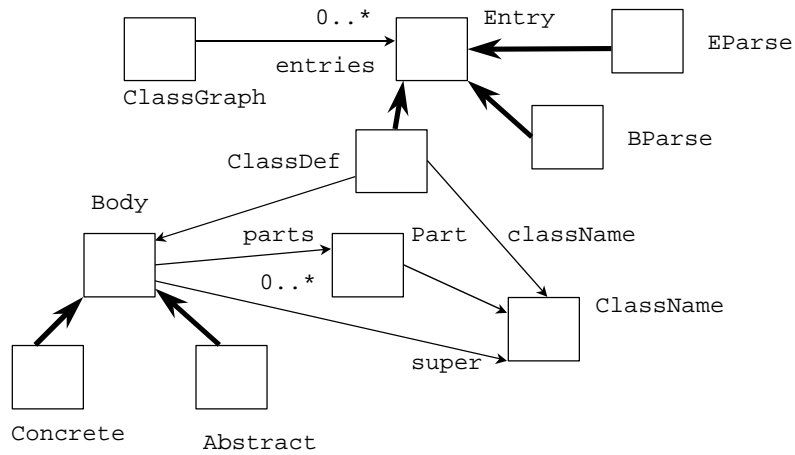
ClassGraph

```
self.{to Abstract}->  
  forAll(a|a.parts->size=0)
```

UML class diagram ClassGraph



UML class diagram ClassGraph



12/10/98

AOO / UML / OCL / Strategies

69

Applications

- Abstract superclass rule

ClassGraph

```

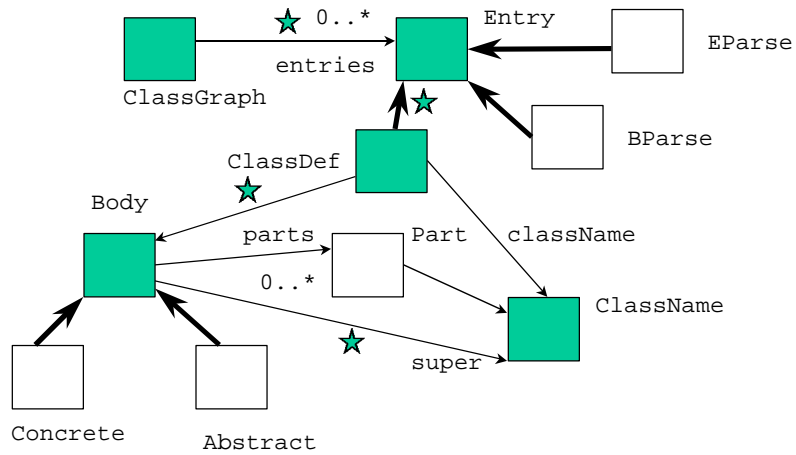
superCls =
  self.{through->*,super,* to ClassName};
self.{to ClassDef}->
  forAll(c|
    if (superCls->includes(c.className))
      then c.{to Abstract}->size=1
    else true
    endif)
  
```

12/10/98

AOO / UML / OCL / Strategies

70

UML class diagram ClassGraph

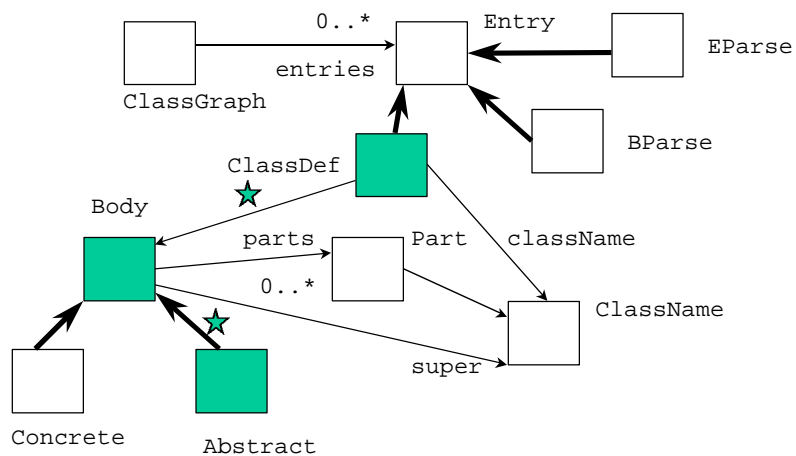


12/10/98

AOO / UML / OCL / Strategies

71

UML class diagram ClassGraph



12/10/98

AOO / UML / OCL / Strategies

72

Conclusions

- OCL is a suitable language for expressing object properties, class invariants and method pre- and post-conditions. (needs capability to define functions and auxiliary variables).
- OCL is NOT a good language for navigation but can be made into one by adding strategies.

Further information

- www.rational.com contains latest information about UML, specifically OCL.
- www.ics.uci.edu/pub/arch/uml

Context switch

Drawbacks of object-oriented software development

- programs are tangled and redundant
 - object-structure tangling, encoded repeatedly
 - synchronization tangling
 - distribution tangling
- programs are hard to maintain and too long
 - because of tangling and redundancy

Eliminating drawbacks with adaptive software

- *Adaptive software* is software which adjusts automatically to changing contexts/aspects.
- Contexts: *object structure*, object marshalling, synchronization, exceptions etc.
- Solution: Split software into cooperating, *loosely* coupled context- or aspect-descriptions. *Strategies* refer to structure.

What is Demeter

- A high-level interface to object-oriented programming and specification systems
- Demeter System/OPL =
Demeter Method + Demeter Tools/OPL
- So far: OPL = {Java, C++, Perl, Borland Pascal, Flavors}
- Demeter Tools/OPL = Demeter/OPL

Also useful for specification

- UML OCL = Unified Modeling Language Object Constraint Language (an OMG (= Object Management Group) standard)

Risks of adaptive OO

- Advantages: Simpler programs for next project (compensates for learning curve). Programs are easier to evolve and maintain.
- Disadvantages: Additional training costs. New concepts, debugging techniques and tools to learn.

Experience regarding training costs

- GTE project which took approximately four man months by non-adaptive techniques, took only 7 days to complete with adaptive techniques (using Demeter/Java).
- Our experience with Demeter/C++ is that the first project also has a shorter development time and maintenance is much simpler.

Scenarios

- Best: Use Demeter/Java for future projects. Build library of adaptive components. Reduce software development and maintenance costs significantly.
- Worst: Use Demeter/Java only to generate Java code, but then you maintain Java code manually. You still win since a lot of useful Java code is produced.

History

- Hades (HARdware DEScription language
1982 by Niklaus Wirth at ETH Zurich)
- Zeus (a brother of Hades, a silicon
82-85 compilation language at Princeton
University/MIT, implemented at GTE Labs;
predecessor of VHDL)
- Demeter (a sister of Zeus, used to
1985- implement Zeus, started at GTE Labs)

12/10/98

AOO / UML / OCL/Strategies

83

Data bases

- AP and object-oriented data bases
 - design rules for class dictionaries
 - persistence
 - query languages
 - P3

12/10/98

AOO / UML / OCL/Strategies

84

Normalization

- Functional dependency
 - A part V1 of some class C is *functionally dependent* on some part V2 if for all instances of class C each value of V2 has no more than one value of V1 associated with it. V2 functionally determines V1.
 - Extend to sets of parts

Normalization

- Key
 - A *key* for a class C is collection of parts that (1) functionally determines all parts of C, and (2) no proper subset has this property.
 - The concept of the key of a class is not a property of the class definition, but rather about the intended use of the class.

Normalization

- A class is *normalized* if, whenever a part is functionally dependent on a set S of parts, S contains a key (an adaptation of Boyce - Codd normal form to classes).

Example

```
Employee =  
  <employeeNumber> Integer //key  
  <employeeName> String  
  <salary> Integer  
  <projectNumber> Integer  
  <completionDate> String.
```

Source of problems:
project number
determines completion
date, but project
number is not part
of key.

Three Problems:

Before employees are recruited for a project, the completion date can be stored only in a strange way.

If all employees leave project, completion date info. lost.

If completion date of a project is changed, have to search through all instances of class Employee.

Example/Solution

```
Employee =  
  <employeeNumber> Integer //key  
  <employeeName> String  
  <salary> Integer  
  <projectNumber> Integer  
Project =  
  <projectNumber> Integer //key  
  <completionDate> String.
```

Three Problems solved:



Before employees are recruited for a project, the completion date can be stored only in a strange way.



If all employees leave project, completion date info. lost.



If completion date of a project is changed, have to search through all instances of class Employee.

12/10/98

AOO / UML / OCL/Strategies

89

Persistence

- Which parts of an object need to be saved in the OO data base?
 - Define a visitor: `DBCopyVisitor` (analogous to `CopyVisitor`).
 - Define a strategy which says which parts need to be saved to the data base.
 - `o.t(DBCopyVisitor);`

12/10/98

AOO / UML / OCL/Strategies

90

Query Languages

- Problem: Users need to know too much about the data base schema (= UML class diagram) when they write queries.
- Solution: Use strategies with some extensions to write queries.

from `a:A` via `b:B` to `C`
at the `C`-object `a` and `b` are available.

Commercial aspects

- Many companies selling OODBs, e.g. Object Design in Burlington, MA.
- OQL: a standard query language
 - AQL: an adaptive version of OQL (not implemented)

Case study

- Learning cd from examples.