A Specification Language

UML: www.rational.com/uml

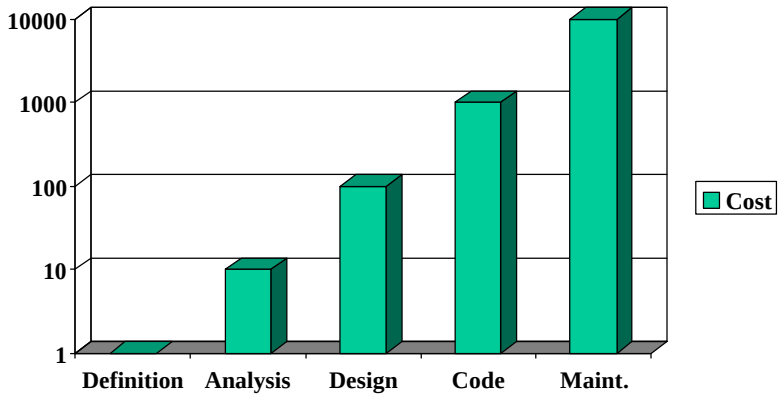
Model architecture

Object Constraint Language

Testing and OCL

- A class model is not enough for a precise and unambiguous specification.
- Useful to express specifications, test requirements and test specifications
- A useful notation to know: part of UML

The relative cost of correcting an error increases over SDLC



How to prevent defects?

- More precise specifications. OCL helps.
- Start testing early, at design level: See homework 2!!!
- Use walkthroughs and inspections

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

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

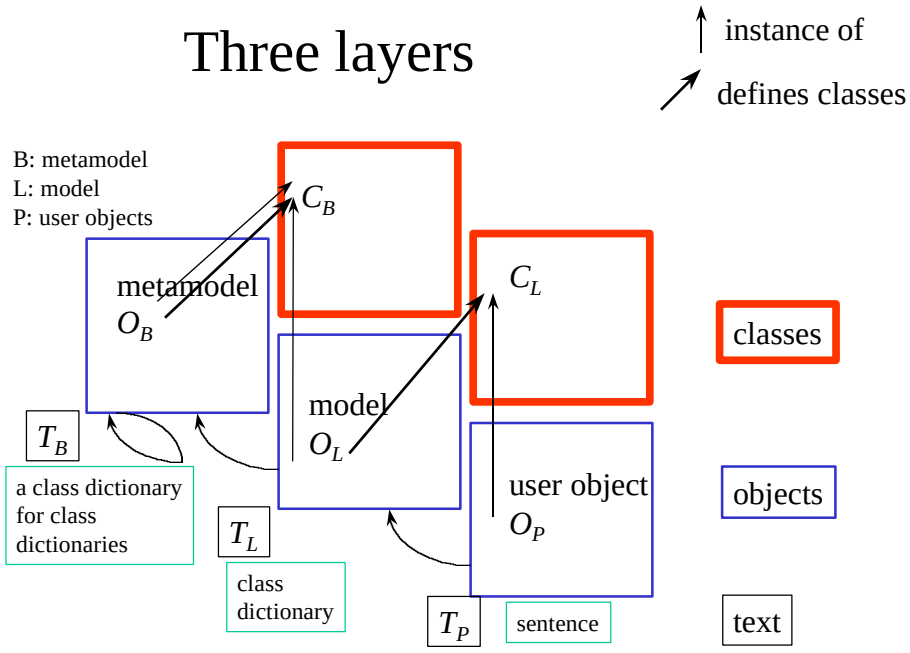
Four layer architecture

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

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

Three layers



UML OCL

- Object Constraint Language
  - allows you to define side effect-free constraints for UML and other models
  - 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)

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

Where to use OCL?

- Specify invariants for classes and types
- Specify pre- and post-conditions for methods
- As a navigation language
- To specify constraints on operations
- Test requirements and specifications

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.

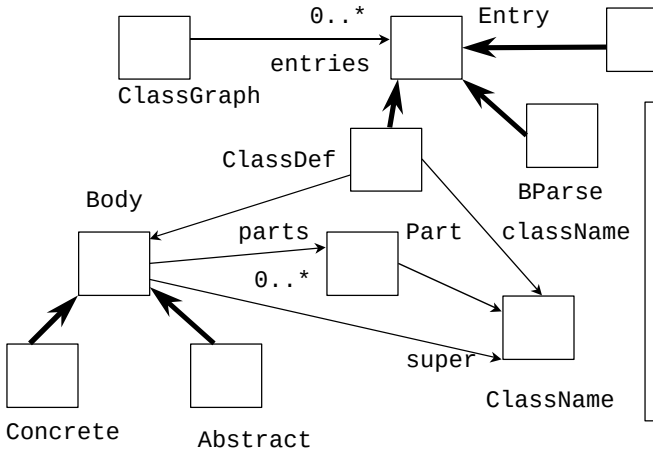
```
Company
   self.numberOfEmployees
c : Company
   c.numberOfEmployees
```


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.

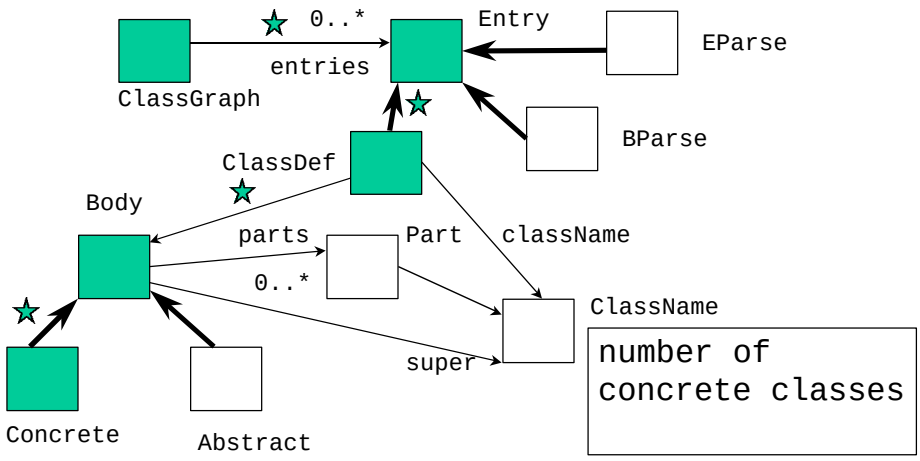
```
Person
  self.age >= 0
```

Example: UML class diagram ClassGraph



Note: we use a thick arrow to denote inheritance. UML uses an open arrow.

UML class diagram ClassGraph



number of concrete classes

Example

```
-> collection op
select:subset
```

- 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 …):
   ReturnType
 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.

May5 1998

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)`

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();
   ...
```


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 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)
```

Collection Operations

- Select and reject operations
  - `collection->select (v:Type|boolean-expr-with-v)`

```
Company self.employee->select
 (p:Person|p.age > 50)
```


Select syntax

- Define a subset
  - `collection->select (v:Type|boolean-expr-with-v)`
  - `collection->select (v|boolean-expr-with-v)`
  - `collection->select (boolean-expr)`

`type redundancy`

`refer to entire object`

`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 colletion by applying expression to elements of old collection

- Creates a bag

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

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`

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

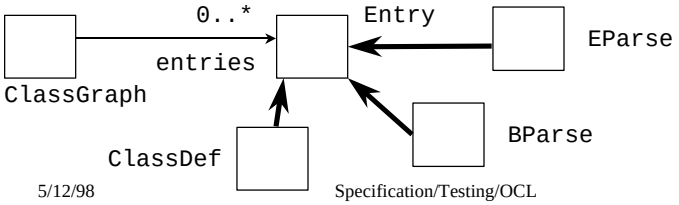
- OclType: type: instance of OclType
  - `type.name : String`
  - `type.attributes:Set(String)`
  - `type.associationEnds:Set(String)`
  - `type.operations:Set(String)`
  - `type.supertypes:Set(OclType)`
  - `type.allSupertypes:Set(OclType)`
  - `type.allInstances:Set(type)`

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`

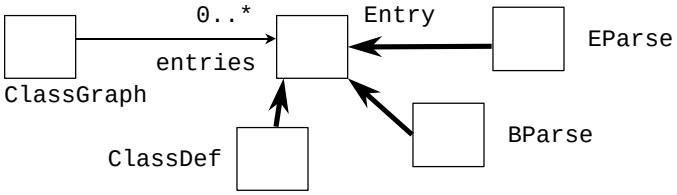
Applications

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



Applications

- Number of class definitions: What about using strategies to define collections?
  - `ClassGraph self.{to ClassDef}`
    `->size`



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*.

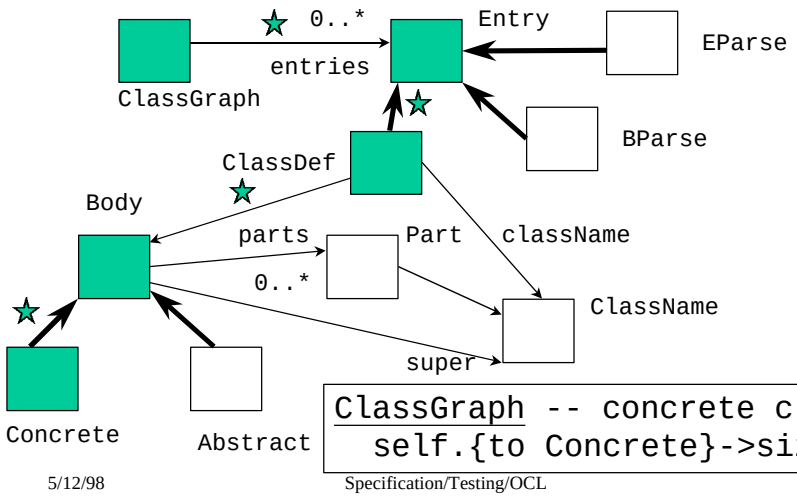
Improve OCL

- 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
```

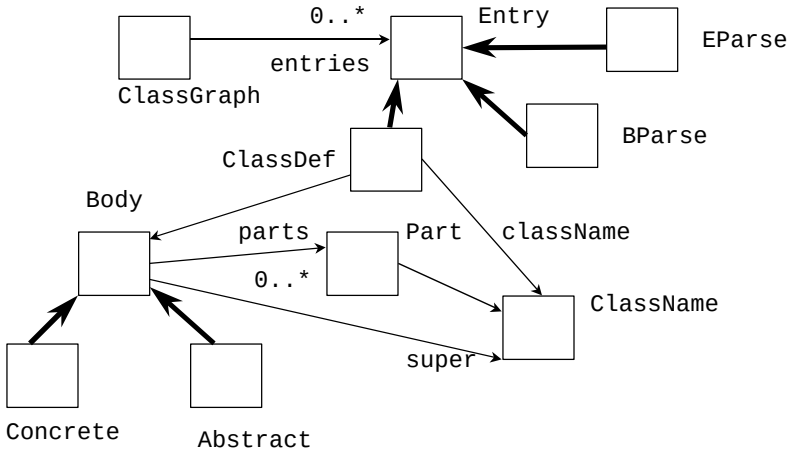


```
ClassGraph self.entries->
     select(c:Entry|c.
      oclIsTypeOf(ClassDef))->
       collect(body)->
        select (b:Body|b.
         oclIsTypeOf(Concrete))
  ->size -- count concrete classes
```

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

Which one is easier to write?

UML class diagram ClassGraph

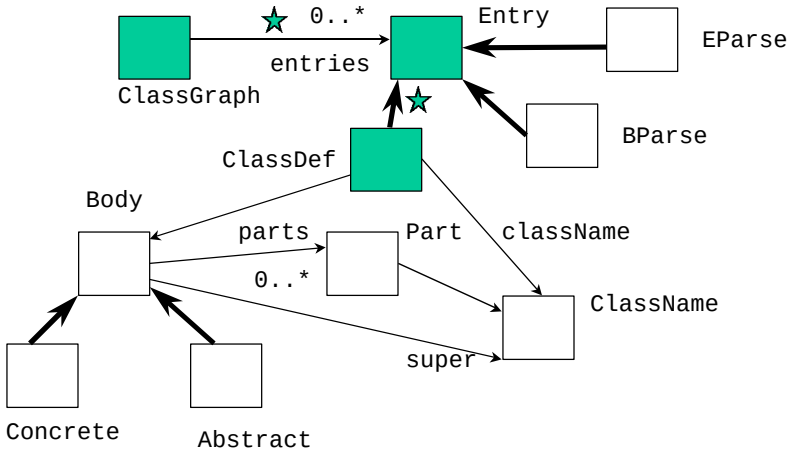


Applications

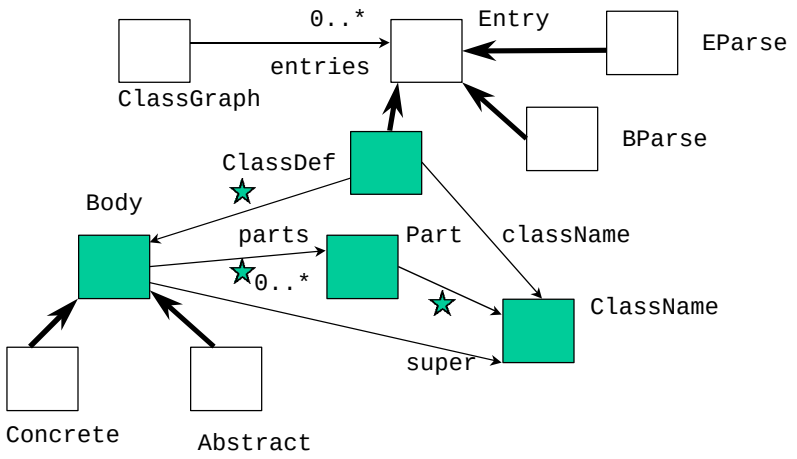
- Terminal buffer rule

```
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}
```

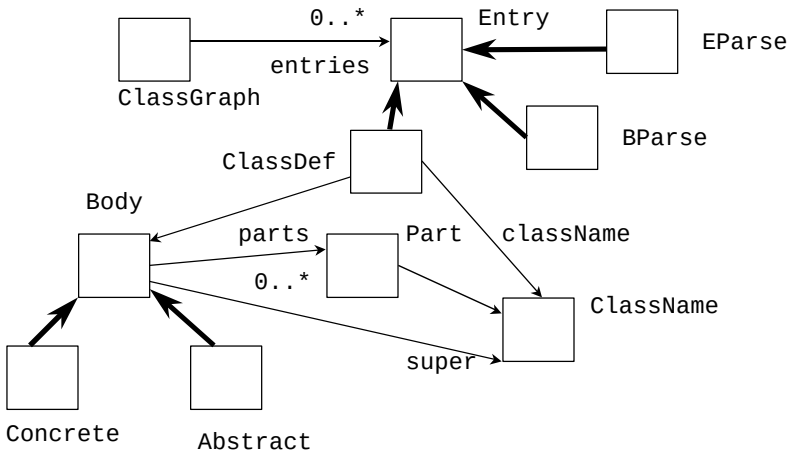
UML class diagram ClassGraph



UML class diagram ClassGraph



UML class diagram ClassGraph

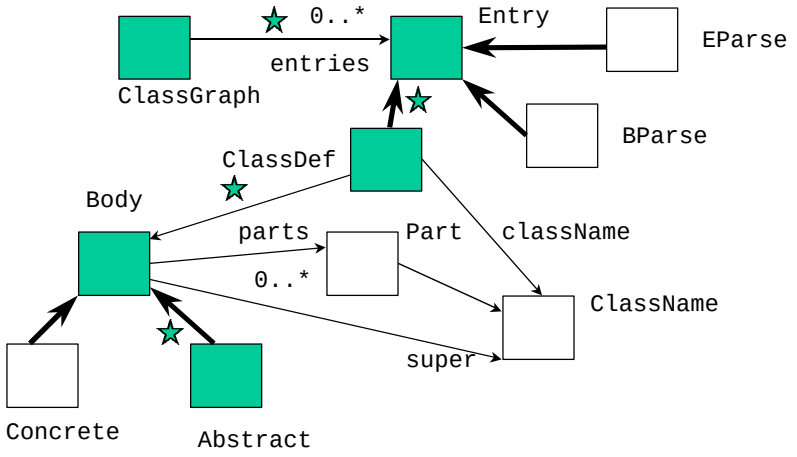


Applications

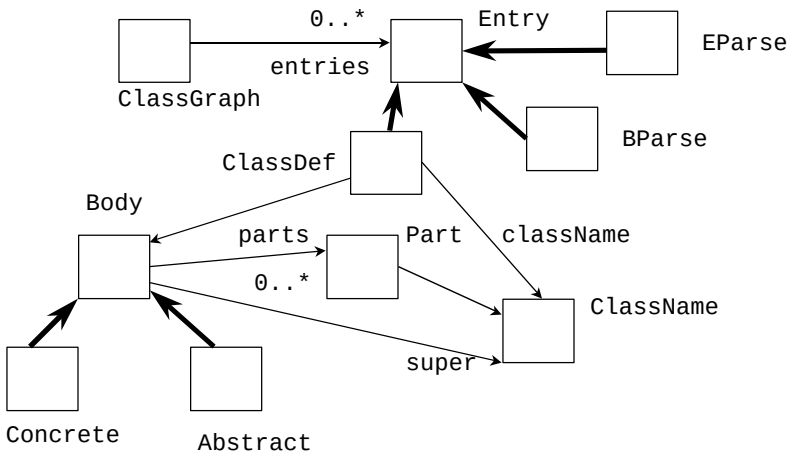
- Class graph is flat

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

UML class diagram ClassGraph



UML class diagram ClassGraph

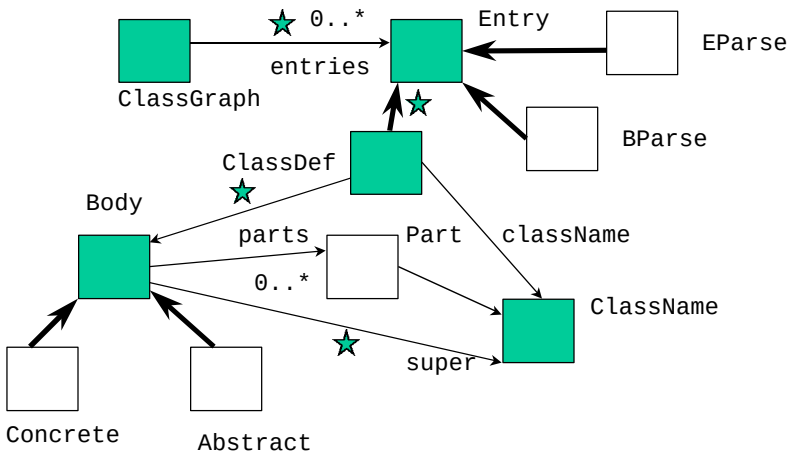


Applications

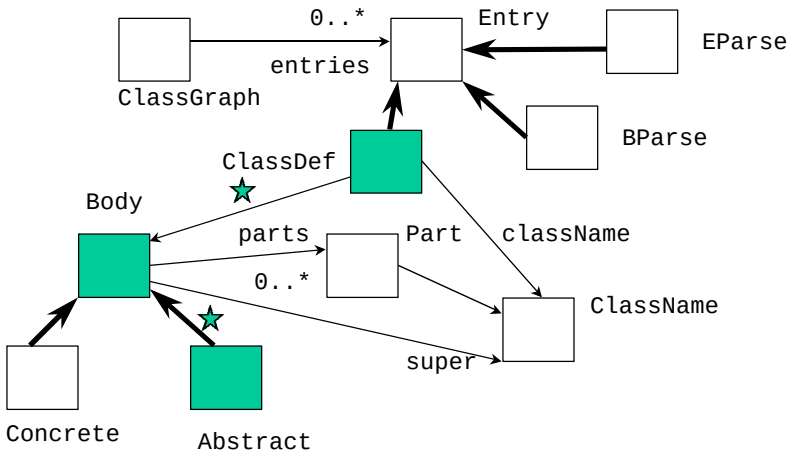
- Abstract superclass rule

```
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)
```


UML class diagram ClassGraph



UML class diagram ClassGraph



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