Relationships between
Class Diagrams, Object Diagrams and OCL Invariants

Martin Gogolla
University of Bremen
Outline

• Running example

• Basic concepts in class diagrams
  • Class, attribute, association

• Basic concepts in object diagrams
  • Object, value assignment, link

• Invariants for restricting object diagrams
  • Invariant context
  • Invariant fulfilment
Basic class diagram concepts

- Class with class name
- Attribute with attribute name and datatype
- Datatype with datatype name
  - refers to a set of datatype values and
  - to datatype operations
- Association with association name
  - Role with role name
  - Role multiplicity with lower and upper bound
datatypes = { Integer, String }

classes = { Person, Company }

associations =
{ WorksFor( 1..* : employee : Person, 0..1 : employer : Company),
  WorkerBoss( 0..1 : boss : Person, 0..* : worker : Person) }

attributes =
{ Person :: name : String,
  Person :: age : Integer,
  Person :: salary : Integer,
  Company :: name : String }

operations =
{ Company :: sumSalary() : Integer }
objects [ Person ] = { ada, bob, cyd }

objects [ Company ] = { ibm, sun }

values [ Person::name ] = 
  { ada → 'Ada Alewife',
    bob → 'Bob Baker',
    cyd → 'Cyd Cook' }

values [ Person::age ] = ...

values [ Person::salary ] = ...

values [ Company::name ] = ...
Basic object diagram concepts (for classes, attributes)

- Class diagram interpreted by the set of all possible object diagrams
- Each single class interpreted in an object diagram by a finite set of objects
- Each object has an object identity that is unique in the object diagram
- Each class attribute interpreted by assigning a datatype value to the attribute for all relevant objects
objects [ Person ] = { ada, bob, cyd }

objects [ Company ] = { ibm, sun }

links [ WorksFor ] =
{ (employee:ada, employer:ibm),
  (employee:bob, employer:ibm),
  (employee:cyd, employer:sun) }

links [ WorkerBoss ] =
{ (worker:bob, boss:ada) }

Exchanging the WorkerBoss link roles gives a different object diagram
Basic object diagram concepts (for associations)

• Each single association interpreted in an object diagram by a (finite) set of links (finiteness implied by finite object set)
• A link connects two (or more) objects
• A link can be considered as a tuple of object identities together with roles
• A link uses (association) roles to describe the character that an object plays in the link

Summary

• Class diagram → Set of all object diagrams
• In an object diagram
  • Class cs → Finite set of objects for cs
  • For object ob, attribute at → Value assignment for at in ob
  • Association as → Finite set of links for as
Exchanging any WorkerBoss link role gives a different object diagram.
There are object diagrams that correspond to non-meaningful real-world situations with regard to attribute values and links.

Needed: Mechanism to ban 'bad' object diagrams.
Required conditions in example

class context Company inv employeeAtLeast16

class context Company inv nameUnique

class context Person inv acyclicBossWorker

class context Person inv ageReasonable

class context Person inv bossSameCompany

class context Person inv nameUnique

class context Person inv salaryPositive

All conditions violated in example
Required conditions in example

context Company inv employeeAtLeast16
context Company inv nameUnique
context Person inv acyclicBossWorker
context Person inv ageReasonable
context Person inv bossSameCompany
context Person inv nameUnique
context Person inv salaryPositive

All conditions violated in example
Required conditions in example

class context Company inv employeeAtLeast16

class context Company inv nameUnique

class context Person inv acyclicBossWorker

class context Person inv ageReasonable

class context Person inv bossSameCompany

class context Person inv nameUnique

class context Person inv salaryPositive

All conditions violated in example
Required conditions in example
context Company inv employeeAtLeast16
context Company inv nameUnique
context Person inv acyclicBossWorker
context Person inv ageReasonable
context Person inv bossSameCompany
context Person inv nameUnique
context Person inv salaryPositive

All conditions violated in example
Required conditions in example
context Company inv employeeAtLeast16
context Company inv nameUnique
context Person inv acyclicBossWorker
context Person inv ageReasonable
context Person inv bossSameCompany
context Person inv nameUnique
context Person inv salaryPositive

All conditions violated in example
Required conditions in example

context Company inv employeeAtLeast16
context Company inv nameUnique
context Person inv acyclicBossWorker
context Person inv ageReasonable
context Person inv[nameUnique, bossSameCompany]
context Person inv salaryPositive

All conditions violated in example
Required conditions in example

class context Company inv employeeAtLeast16
context Company inv nameUnique
context Person inv acyclicBossWorker
context Person inv ageReasonable
context Person inv bossSameCompany
context Person inv nameUnique
context Person inv salaryPositive

All conditions violated in example
Required conditions in example
context Company inv employeeAtLeast16
context Company inv nameUnique
context Person inv acyclicBossWorker
context Person inv ageReasonable
context Person inv bossSameCompany
context Person inv nameUnique
context Person inv salaryPositive

All conditions violated in example
To exclude non-meaningful object diagrams

- Additional OCL constraints, called invariants, are introduced
- OCL constraints are formulas that are expressed from the viewpoint of a particular class, the so-called context class
  → Invariant context
- Evaluation in an object diagram is done for all objects of the context class and may yield False or True
- Only when the evaluation of all invariants yields True, the object diagram is considered to be meaningful
  → Invariant fulfillment
context Person inv ageReasonable:
  0<=age and age<=110

ccontext Person inv salaryPositive:
  1<=salary

context p:Person inv bossSameCompany:
  (p.employer->size()==1 and p.boss->size()==1) implies p.employer=p.boss.employer

context p:Person inv acyclicBossWorker:
  p.worker->closure(worker)->excludes(p) -- closure expression 
  p.worker U p.worker.worker U p.worker.worker.worker U ...

context c:Company inv employeeAtLeast16:
  c.employee->forAll(p | p.age>=16)

context p1,p2:Person inv nameUnique:
  p1<>p2 implies p1.name<>p2.name

context c:Company inv nameUnique:
  not Company.allInstances->exists(d | d<>c and d.name=c.name)
p1.worker->closure(worker) = Set{ p1, p2, p3 }

p2.worker->closure(worker) = Set{ p1, p2, p3 }
ada.worker->closure(worker) = Set{ bob }

bob.worker->closure(worker) = Set{ }
Summary

• Class diagram → Set of all object diagrams

• In an object diagram
  • Class cs → Finite set of objects for cs
  • For object ob attribute at → Value assignment of at for ob
  • Association as → Finite set of links for as

• Class diagram with invariants → Set of all object diagrams in that all invariants are true for all objects
Thanks for your attention!
How is the operation Company::sumSalary() implemented?
Company::sumSalary() =
    self.employee->collect(p | p.salary)->sum()
ibm.sumSalary() = 5000 in the 'good' object diagram

How does 'closure' work?
p.worker->closure( worker ) : Set(Person) =
    " p.worker ->union( p.worker.worker )
        ->union( p.worker.worker.worker )
        ->union( p.worker.worker.worker.worker )
        ->union ( ... ) ...
    "
until no more new workers appear; only a finite set of workers (persons) possible

What are 'inherent constraints'?
inherent constraints = model inherent constraints
constraints that are already formulated in the UML class diagram
for example, the multiplicity restrictions
context p:Person inv employer_0_1: p.employer->size()<=1
context c:Company inv employee_1_*: c.employee->size()>=1
context p:Person inv boss_0_1: p.boss->size()<=1
Can you give an example for a WorkerBoss hierarchy with 3 levels?
Can closure be used only in context of reflexive associations (one class used twice)?
Are there other collection operations apart from size(), closure(...), excludes(...), forAll(...), exists(...)?

context p:Person inv bossSameCompany:
    (p.employer->size()=1 and p.boss->size=1) implies p.employer=p.boss.employer

class C:

context p:Person inv acyclicBossWorker:
    p.worker->closure(worker)->excludes(p)

class C:

context c:Company inv employeeAtLeast16:
    c.employee->forAll(p | p.age>=16)

context c:Company inv nameUnique:
    not Company.allInstances->exists(d | d<>c and d.name=c.name)

context c:Company inv nameUnique:
    Company.allInstances->select(d | d<>c and d.name=c.name)->isEmpty() -- allowed: ...->notEmpty()

context c:Company inv nameUnique:
    not Company.allInstances->select(d | d<>c)->collect(c | c.name)->includes(c.name)

Important collection operations (even more operations reject(...), one(...), any(...), iterate(...), ...):
- size() : size of collection = number of collection elements
- isEmpty(), notEmpty() : collection has no elements, collection has at least one element
- forAll( cond ), exists( cond ) : condition holds for all elements, condition holds for at least one element
- select( cond ), collect( term ) : sub-collection with elements satisfying condition, collection with elements mapped by term
- includes( elem ), excludes( elem ) : collection contains element, collection does not contain element
- closure( term ) : collection obtained by continuation of term computation = reflexive, transitive closure