SWEN 223
Software Engineering Analysis

Object-Constraint Language

Thomas Kühne
Victoria University of Wellington
Thomas.Kuehne@ecs.vuw.ac.nz, Ext. 5443, Room Cotton 233

Why Constraints?

Achieving Well-Formedness

- UML class diagrams are type models for all their possible instance models
  - » concepts, their allowed relationships and multiplicities restrict the set of instance models conforming to a type model
- However, a number of restrictions cannot be expressed by means of the visual notation only
  - » Similar in programming languages: The grammar has to be augmented by static semantics rules

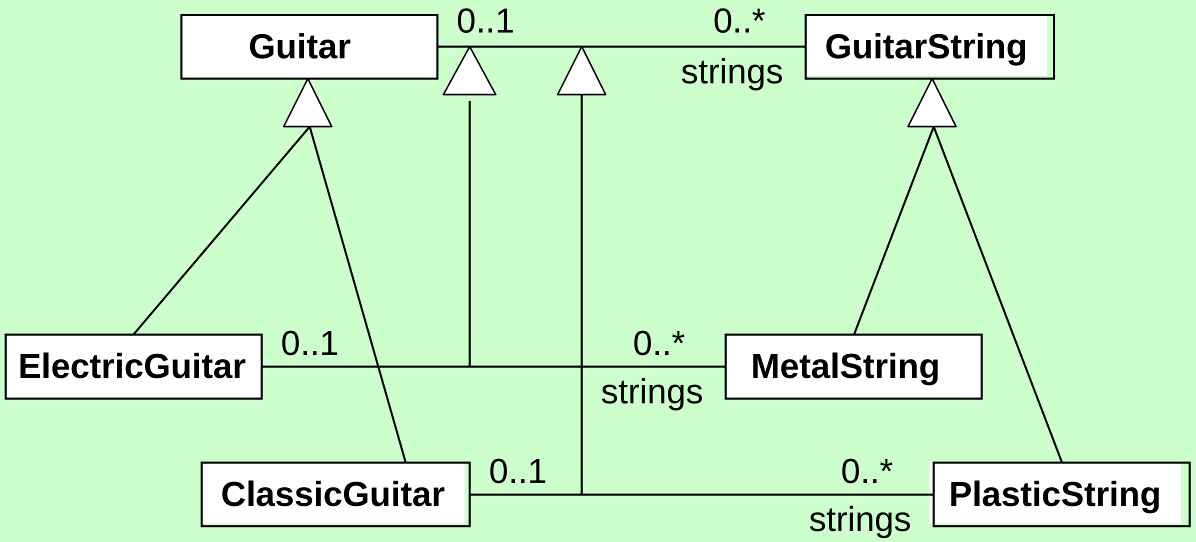
Applications

Constraints are used to express, e.g.,

- Limits
  - constrain values to certain ranges
- Uniqueness
  - constrain instances values to be unique
- Consistency
  - express invariants on data structures
- Contracts
  - pre- and post conditions for operations

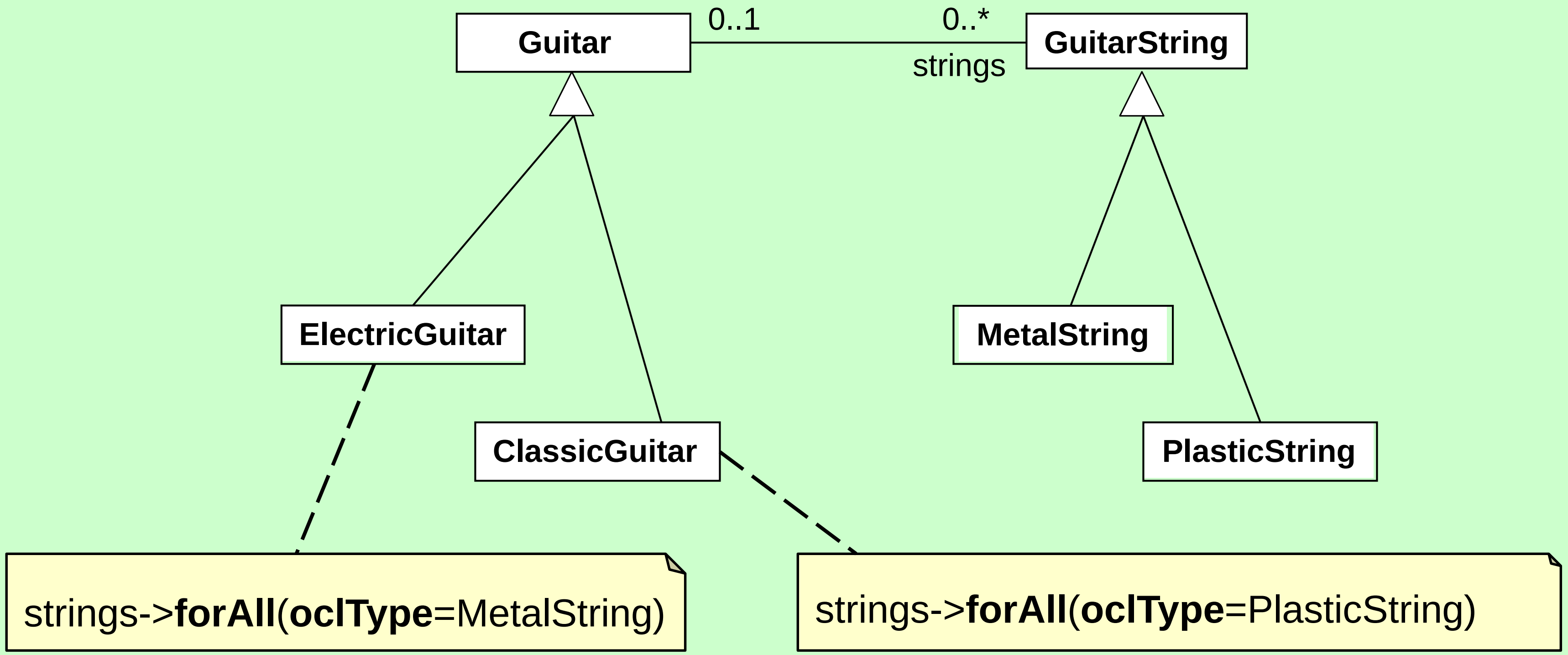
Graphical vs Constraints

Fully graphical



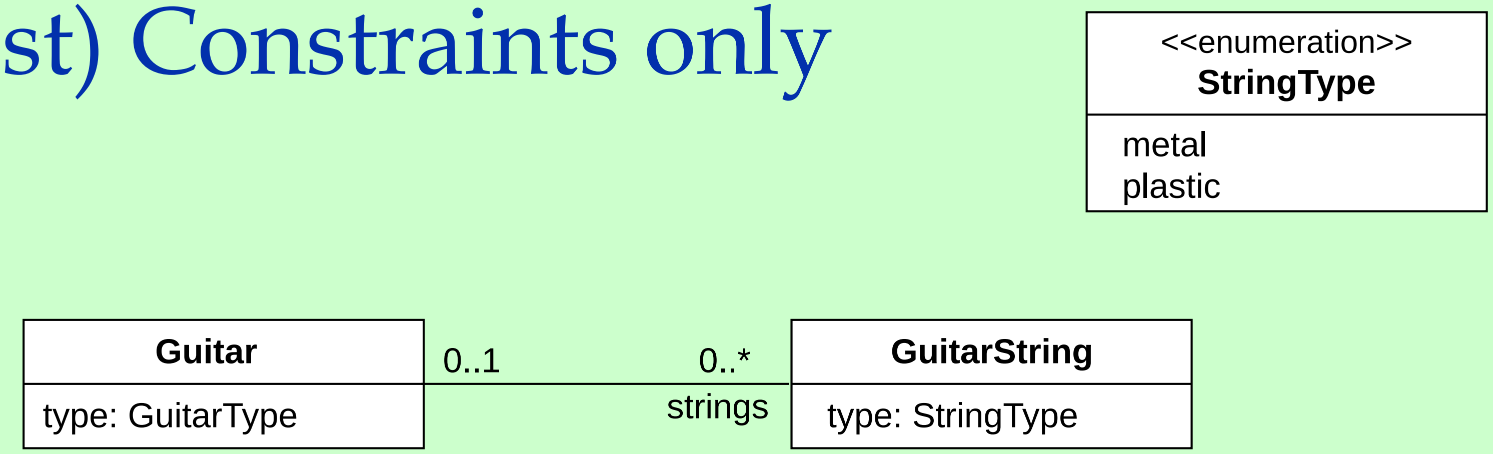
Graphical vs Constraints

Graphical with constraints


Guitar
0..1
0..*
strings
GuitarString
ElectricGuitar
ClassicGuitar
MetalString
PlasticString
strings->**forAll**(**oclType**=MetalString)
strings->**forAll**(**oclType**=PlasticString)

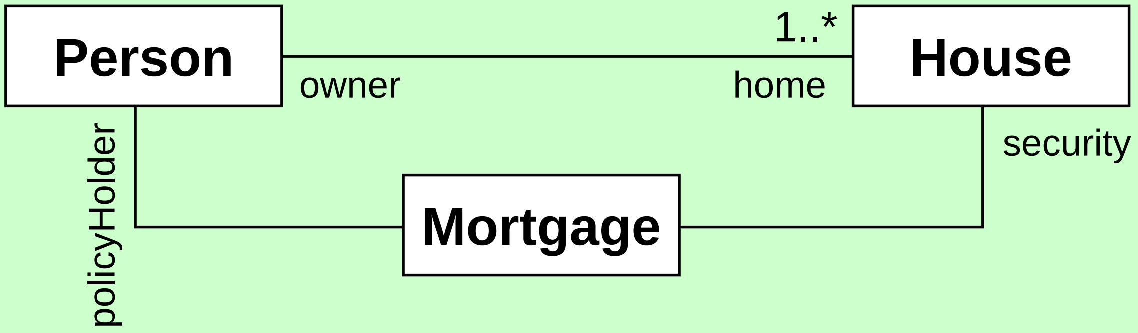
Graphical vs Constraints

(Almost) Constraints only



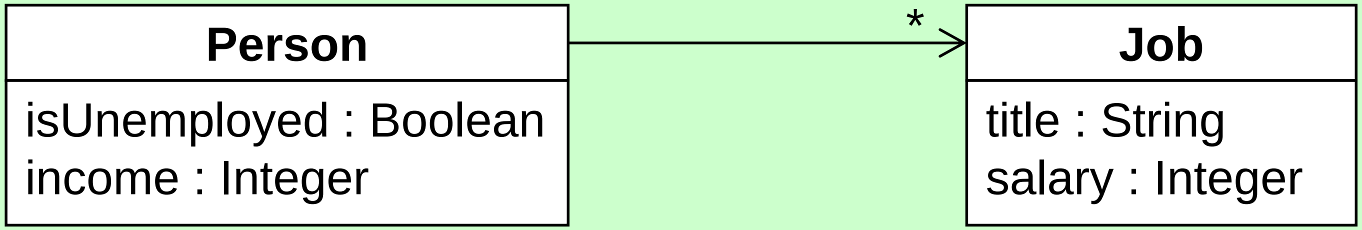
```
context Guitar inv correctStrings:
    (self.type = GuitarType::electric  implies strings->forAll(type = StringType::metal) and
    (self.type = GuitarType::classic   implies strings->forAll(type = StringType::plastic)
```

Fixing Identity



context Mortgage **inv** ownsSecurity:
self.policyHolder **=** **self**.security.owner

Invariants



```
context Person inv appropriateSalary:
let income : Integer = self.job.salary->sum() in
  let hasTitle(t : String) : Boolean = self.job->exists(title = t) in
    if isUnemployed then
      self.income < 300
    else
      self.hasTitle('manager') implies self.income >= 4000
    endif
```

Types of Constraints

- Invariant
  - » constraint that states a condition that must always be met by all instances of the type. Invariants must be true all the time (except during operation execution).
- Precondition
  - » a precondition to an operation is a restriction that must be true before the operation is going to be executed.
- Postcondition
  - » a postcondition to an operation is a restriction that must be true after that the operation has just ended its execution.

Beyond Types

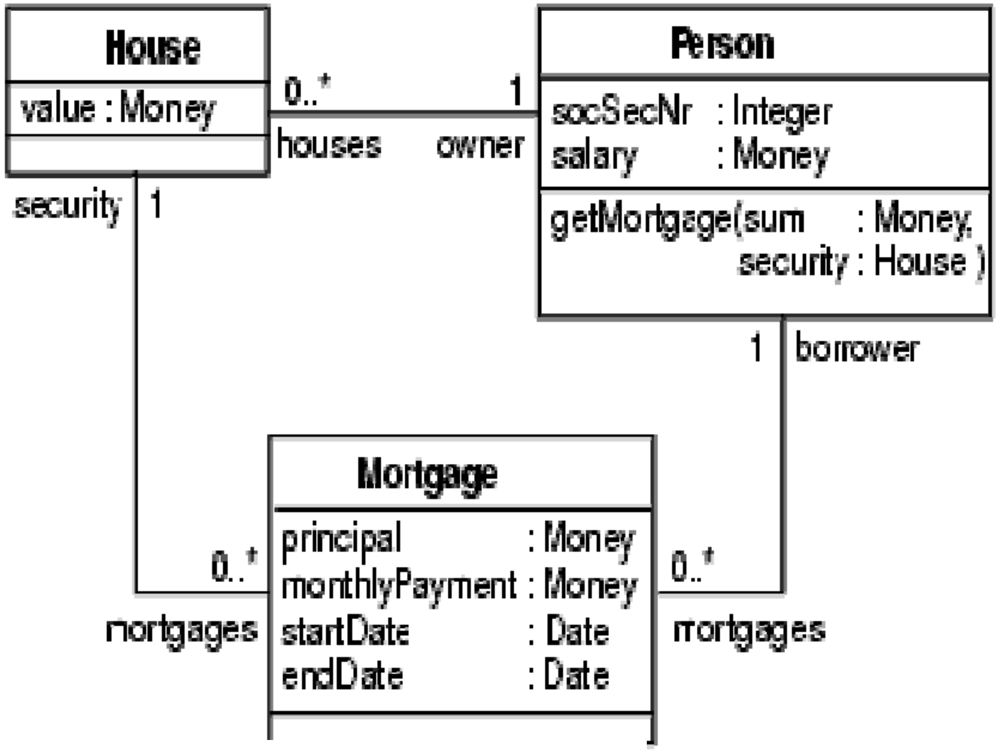
context *Mortgage*
inv*: security.owner = borrower*

context *Mortgage*
inv*: startDate < endDate*

context *Person*
inv*: Person::allInstances()->isUnique(socSecNr)*

context *Person::getMortgage(sum : Money, security : House)*
pre*: self.mortgages.monthlyPayment->sum() <= self.salary * 0.30*

context *Person::getMortgage(sum : Money, security : House)*
pre*: security.value >= security.mortgages.principal->sum()*



Operations on Sets

Operation	***Description***
s->intersection(t)	Returns the intersection of sets s and t
s->union(t)	Computers the union of s and t
s->notEmpty()	True if s contains at least one element
s->size()	Returns the number of elements in set s
s->excludes(o)	True if o is not an element of s
s->isEmpty()	True if s doesn’t contain any elements
s->excludesAll(u)	True if all elements of set u are not in s
s->includesAll(u)	True if all elements of set u are in s
s->includes(o)	True if o is an element of s
s->count(o)	Number of times element o occurs in s

Iterators over Sets

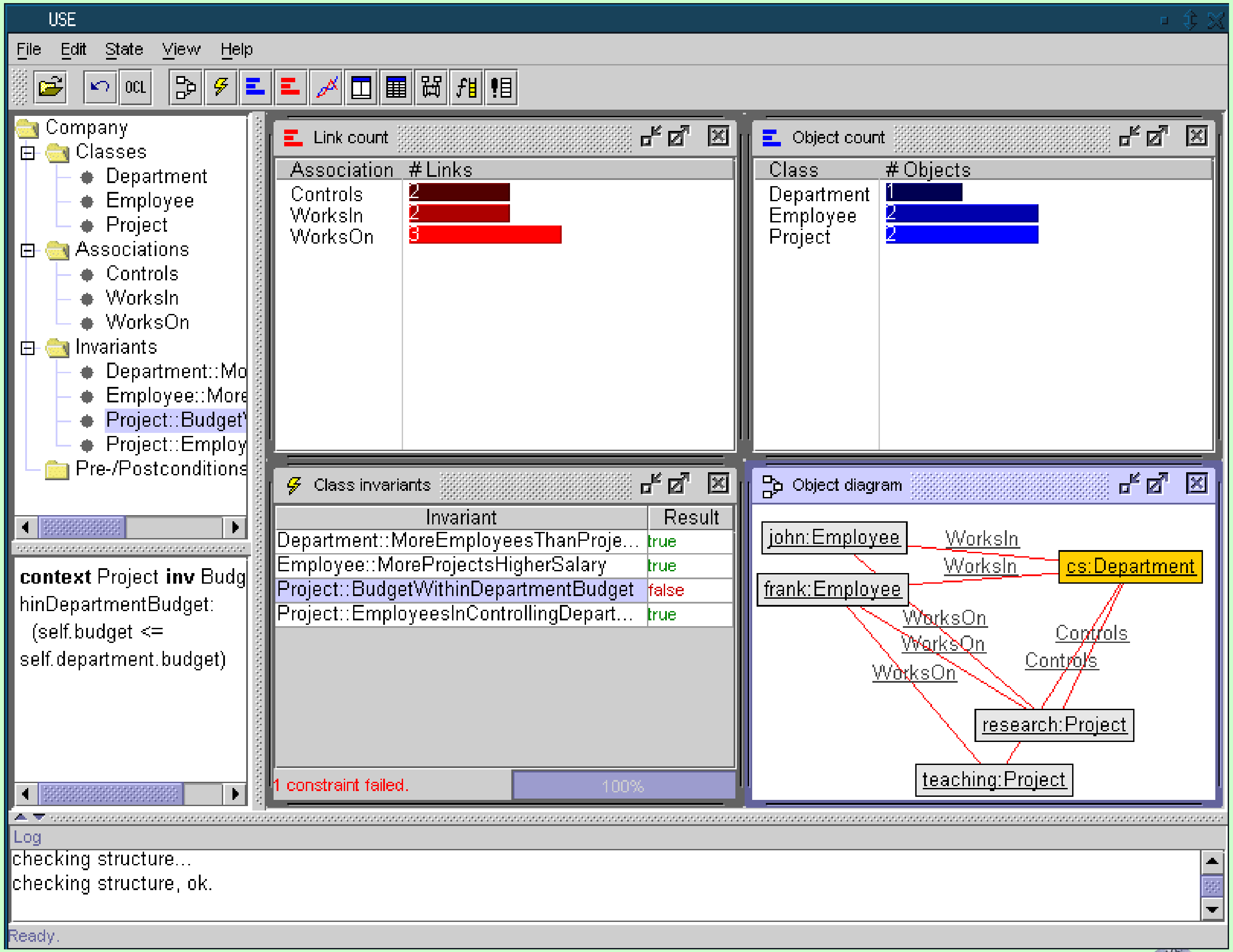
Operation	***Description***
s->reject(expr)	Returns a subset of s containing all elements for which expr is false
s->select(expr)	Returns a subset of s containing all elements for which expr is true
s->forAll(expr)	Returns true if expr is true for all elements in the source collection
s->exists(expr)	Returns true if there is at least one element in the source collection for which expr is true
s->collect(expr)	Returns the set of objects that result from evaluating expr for each element in the source collection
s->any(expr)	Returns a random element for which expr is true

OCL vs Alloy

- OCL
  - integrated into the UML
    - each UML class/interface is automatically an OCL type
    - navigation along associations
  - fully supports primitive types
  - allows recursive definitions
- Alloy
  - supports several styles of specification
    - OO, relational, first order logic
  - comes with a solver

USE Tool

- System states can be created & manipulated
- For each snapshot the constraints are auto-matically checked



Company
- Classes
  - Department
  - Employee
  - Project
- Associations
  - Controls
  - WorksIn
  - WorksOn
- Invariants
  - Department::Mo
  - Employee::More
  - Project::Budget
  - Project::Employ
- Pre-/Postconditions

```
context Project inv Budg
hinDepartmentBudget:
  (self.budget <=
self.department.budget)
```

Association	# Links
Controls	2
WorksIn	2
WorksOn	3

Class	# Objects
Department	1
Employee	2
Project	2

Invariant	Result
Department::MoreEmployeesThanProje...	true
Employee::MoreProjectsHigherSalary	true
Project::BudgetWithinDepartmentBudget	false
Project::EmployeesInControllingDepart...	true

1 constraint failed.

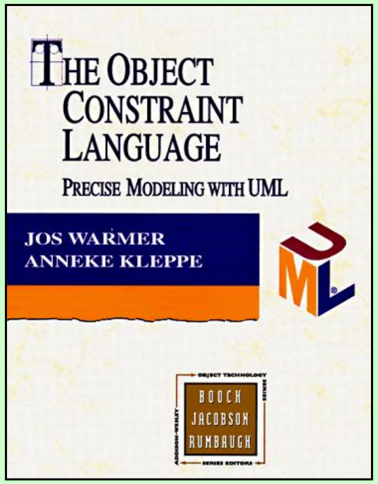
Log
```
checking structure...
checking structure, ok.
```
Ready.

Language Features

- Textual Notation
  - supposedly easier to read than standard logic notations
- Declarative
  - expressions have no side effects
  - “loose semantics” allows admissible solutions but does not prescribe specific solutions
- Statically Typed
  - type errors may be caught before evaluation

Further Resources for OCL

- The Object Constraint Language
  - ISBN 0-201-37940-6 (old)
  - ISBN 0-321-17936-6 (newer, UML 2.0 + MDA)
- OCL home page
  - www.klasse.nl/ocl/index.htm
  - http://www.klasse.nl/books/ocl-intro.html


The Object Constraint Language
Precise Modeling with UML
Jos Warmer
Anneke Kleppe