USE OCL 4 MONDEX

Martin Gogolla

University of Bremen

- OCL - Object Constraint Language
- Part of the UML
- OMG Standard
- Developers of (ideas of) OCL:
  J. Warmer (NL consultancy), S. Cook (MS), B. Selic (IBM), A. Wills (Trident, MS), ...
- Use of OCL within the UML
  - CD: invariants, pre/postconditions of operations, operation definitions, ...
  - SC: guards, pre/postconditions of actions, ...
  - UCD: pre/postconditions of use cases, ...
  - ...
- OCL – 'Assembler' Language of the UML

- OCL concepts
  - Descriptive Language for expressions (logical values, objects, object collections)
  - Objects
  - **Navigation**
  - (Finite!) Collections: Set, Bag, Seq
  - Collection operations: forAll, exists, select, ...
  - Formal semantics: naive set-theoretic (PhD thesis Richters, also part of the OMG standard)
- OCL applications
  - UML metamodel (syntax of the UML)
  - Other metamodels (CWM, MOF, ODM, ER/RE)
  - 'Business' applications

- UML tools with OCL support
  - Poseidon, ArgoUML
  - MagicDraw, MaxUML, Together, XMF-Mosaic
- OCL tools
  - Dresden OCL Compiler (OCL 2 Java)
  - Octopus (Warmer/Kleppe; syntax check & code generator)
  - KeY (Karlsruhe; OCL prover embedded in Together); OCL & verification also done in Zürich (Isabelle) and Kiel (PVS?)
  - OCLE (Romania; validation tool)
  - USE: UML Specification Environment (validation and 'certification' tool)
  - ...

- USE does not assist you in doing proofs, but ...

- USE allows to
  - Get confidence in models (formal descriptions) by 'testing' it with scenarios
  - Check consistency of models (invariants); by constructing an Object Diagram
  - Show independency of invs (no invariant follows from the others); by constructing a state violating an invariant INV but satisfying all other invariants (a state satisfying 'negated INV' and ...)
  - Check whether unwanted properties PROP hold; by showing it is not possible to construct a state where all invariants hold and the unwanted PROP holds; drawback of the technique: search space for state construction has to be restricted by (expressed as) an ASSL program

- USE supports class diagrams (transformation resp. MDA extension under development)
- invariants, pre/posts of operations, op definitions
  - Construct object diagram (system state) explicitly (create & destroy objects & links, set attributes)
  - Check invariants; inspect details with the 'Evaluation Browser'

  - Construct operation call sequence
  - Check pre/postconditions & invariants

  - Generate object diagram descriptively (give desired properties of the object diagram)
  - Describe the search space (a set of object diagrams) by enumerating it with an 'ASSL' program

- MONDEX
  - Abstract model – Between model – Concrete model
  - A model: „small, simple, and easy to understand“
  - Must be possible to be described with OCL
  - B and C models: as far as possible
  - Aim: develop test scenarios (system states & operation call sequences)

  - **Problematic:** No notion of refinement in OCL; How to deal with refinement proofs?
  - Idea: Put A/B resp. B/C into one model; formally relate appropriate elements (e.g. by associations); examine absense of contradictions

- MONDEX Reference model: Z model?
- Danger of encoding Z model in our formalism
- Precise, textual reference description?
- What is the reference description? The yellow book? Variations by Jim? Other variations?
- What is the result for each participating group?
  A yellow book (200 pages)?
  A paper (20 pages)?

- No promises: Our results will depend on the student doing this as her/his Diploma thesis

USE: c:/mg/civstat/scp-sub/civstat.use

File Edit State View Help

- CivilStatusWorld
  - Classes
    - Person
  - Associations
    - Marriage
  - Invariants
    - Person::attributesDefined
    - Person::nameCapitalThenSmallLetters
    - Person::nameIsUnique
    - Person::femaleHasNoWife
    - Person::maleHasNoHusband
  - Pre-/Postconditions
    - pre birth::freshUnlinkedPerson
    - post birth::nameAssigned
    - post birth::civstatAssigned
    - post birth::genderAssigned
    - post birth::isAliveAssigned
    - pre marry::aSpouseDefined
    - pre marry::isAlive
    - pre marry::aSpouseAlive
    - pre marry::isUnmarried
    - pre marry::aSpouseUnmarried
    - pre marry::differentGenders
    - post marry::isMarried
    - post marry::femaleHasMarriedHusband
    - post marry::maleHasMarriedWife
    - pre divorce::isMarried
    - pre divorce::isAlive
    - pre divorce::husbandAlive
    - pre divorce::wifeAlive
    - post divorce::isDivorced
    - post divorce::husbandDivorced
    - post divorce::wifeDivorced
    - pre death::isAlive
    - post death::notAlive
    - post death::husbandWidowed
    - post death::wifeWidowed

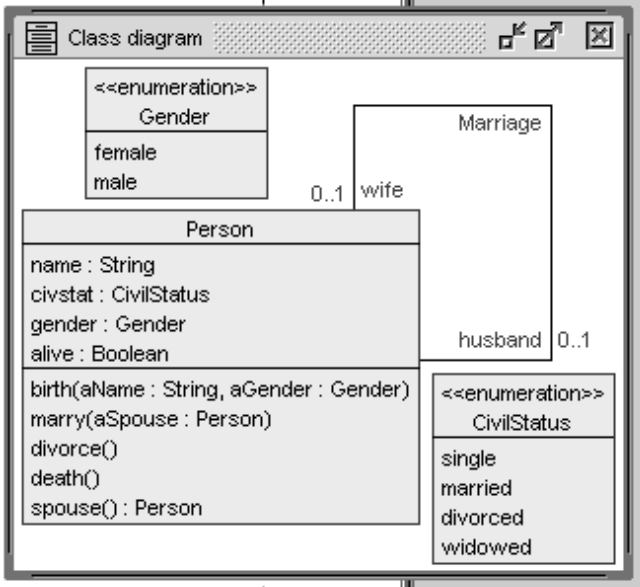
context Person::marry(aSpouse : Person)
pre differentGenders: (self.gender <> aSpouse.gender)

Sequence diagram

ada:Person | bob:Person | cyd:Person | dan:Person

birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()

Class diagram



Class extent

Person	name	civstat	gender	alive
ada	'Ada'	#widowed	#female	true
bob	'Bob'	#divorced	#male	true
cyd	'Cyd'	#married	#male	false
dan	'Dan'	#single	#male	true

Evaluate OCL expression

Enter OCL expression:

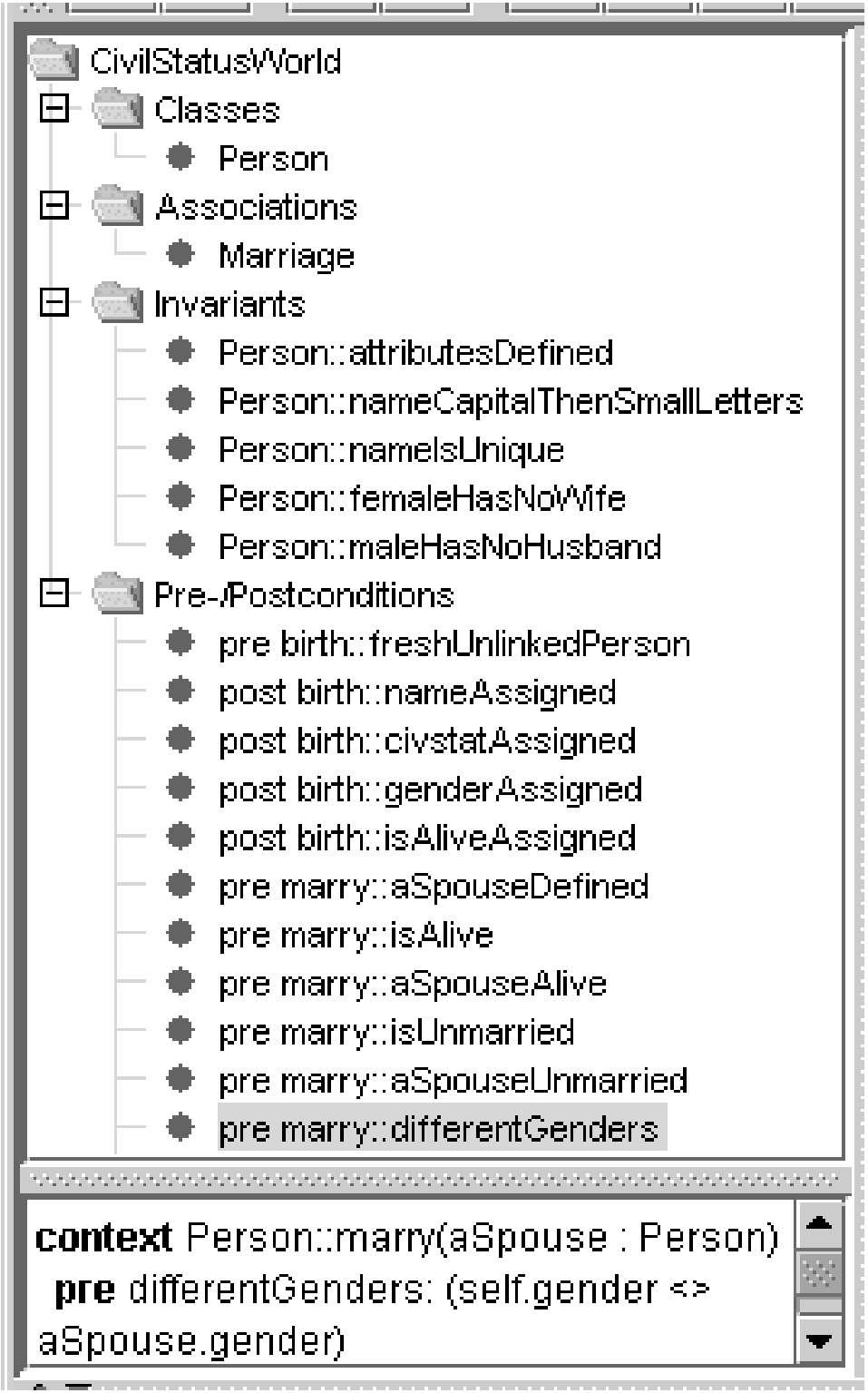
```
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})
```

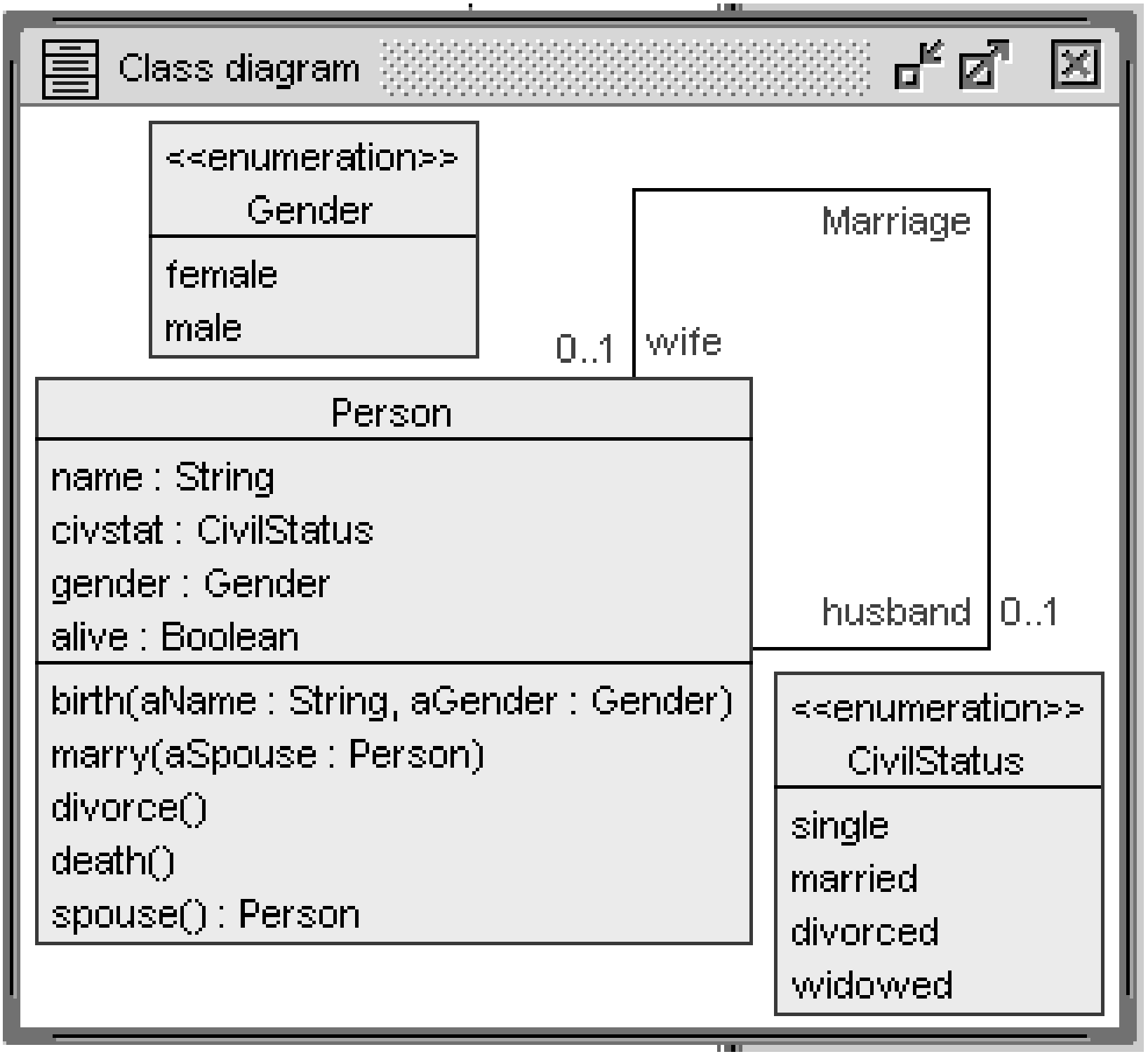
Result:

```
Bag{Sequence{'Ada',#widowed},Sequence{'Bob',#divorced},Sequence{'Dan',#single}} : Bag(Sequence(OclAny))
```

Evaluate | Clear Result | Close

Ready.


CivilStatusWorld
Classes
Person
Associations
Marriage
Invariants
Person::attributesDefined
Person::nameCapitalThenSmallLetters
Person::namelsUnique
Person::femaleHasNoWife
Person::maleHasNoHusband
Pre-/Postconditions
pre birth::freshUnlinkedPerson
post birth::nameAssigned
post birth::civstatAssigned
post birth::genderAssigned
post birth::isAliveAssigned
pre marry::aSpouseDefined
pre marry::isAlive
pre marry::aSpouseAlive
pre marry::isUnmarried
pre marry::aSpouseUnmarried
pre marry::differentGenders
context Person::marry(aSpouse : Person)
pre differentGenders: (self.gender <>
aSpouse.gender)


Class diagram
<<enumeration>>
Gender
female
male
Marriage
0..1
wife
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
husband
0..1
<<enumeration>>
CivilStatus
single
married
divorced
widowed

```
model CivilStatusWorld

enum CivilStatus {single, married, divorced, widowed}
enum Gender {female, male}

class Person

attributes
  name:String
  civstat:CivilStatus
  gender:Gender
  alive:Boolean
end

association Marriage between
  Person [0..1] role wife
  Person [0..1] role husband
end
```

```
birth(aName:String, aGender:Gender)
marry(aSpouse:Person)
divorce()
death()

spouse():Person=
  if gender=#female then husband else wife endif
```

```
marry(aSpouse:Person)
pre  aSpouseDefined: aSpouse.isDefined
pre  isAlive: alive
pre  aSpouseAlive: aSpouse.alive
pre  isUnmarried: civstat<>#married
pre  aSpouseUnmarried: aSpouse.civstat<>#married
pre  differentGenders: gender<>aSpouse.gender
post isMarried: civstat=#married
post femaleHasMarriedHusband: gender=#female implies
     husband=aSpouse and husband.civstat=#married
post maleHasMarriedWife: gender=#male implies
     wife=aSpouse and wife.civstat=#married
```

```
constraints
  inv attributesDefined: name.isDefined and
      civstat.isDefined and
      gender.isDefined and alive.isDefined

  inv nameCapitalThenSmallLetters:
      let small:Set(String)=
        Set{'a','b','c', ... ,'x','y','z'} in
      let capital:Set(String)=
        Set{'A','B','C', ... ,'X','Y','Z'} in
      capital->includes(name.substring(1,1)) and
      Set{2..name.size}->forAll(i |
        small->includes(name.substring(i,i)))

  inv nameIsUnique: Person.allInstances->forAll(self2|
      self<>self2 implies self.name<>self2.name)

  inv femaleHasNoWife:
    gender=#female implies wife.isUndefined
  inv maleHasNoHusband:
    gender=#male implies husband.isUndefined
```

USE: c:/mg/civstat/scp-sub/civstat.use

File Edit State View Help

- CivilStatusWorld
  - Classes
    - Person
  - Associations
    - Marriage
  - Invariants
    - Person::attributesDefined
    - Person::nameCapitalThenSmallLetters
    - Person::nameIsUnique
    - Person::femaleHasNoWife
    - Person::maleHasNoHusband
  - Pre-/Postconditions
    - pre birth::freshUnlinkedPerson
    - post birth::nameAssigned
    - post birth::civstatAssigned
    - post birth::genderAssigned
    - post birth::isAliveAssigned
    - pre marry::aSpouseDefined
    - pre marry::isAlive
    - pre marry::aSpouseAlive
    - pre marry::isUnmarried
    - pre marry::aSpouseUnmarried
    - pre marry::differentGenders
    - post marry::isMarried
    - post marry::femaleHasMarriedHusband
    - post marry::maleHasMarriedWife
    - pre divorce::isMarried
    - pre divorce::isAlive
    - pre divorce::husbandAlive
    - pre divorce::wifeAlive
    - post divorce::isDivorced
    - post divorce::husbandDivorced
    - post divorce::wifeDivorced
    - pre death::isAlive
    - post death::notAlive
    - post death::husbandWidowed
    - post death::wifeWidowed

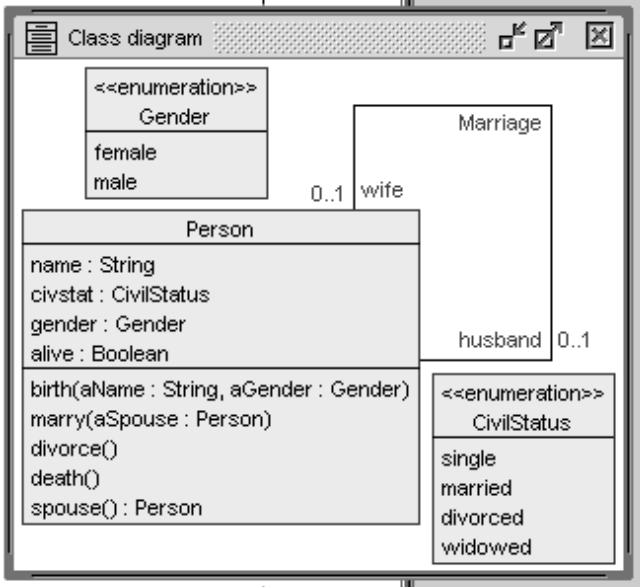
context Person::marry(aSpouse : Person)
pre differentGenders: (self.gender <> aSpouse.gender)

Sequence diagram

ada:Person | bob:Person | cyd:Person | dan:Person

birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()

Class diagram



Class extent

Person	name	civstat	gender	alive
ada	'Ada'	#widowed	#female	true
bob	'Bob'	#divorced	#male	true
cyd	'Cyd'	#married	#male	false
dan	'Dan'	#single	#male	true

Evaluate OCL expression

Enter OCL expression:

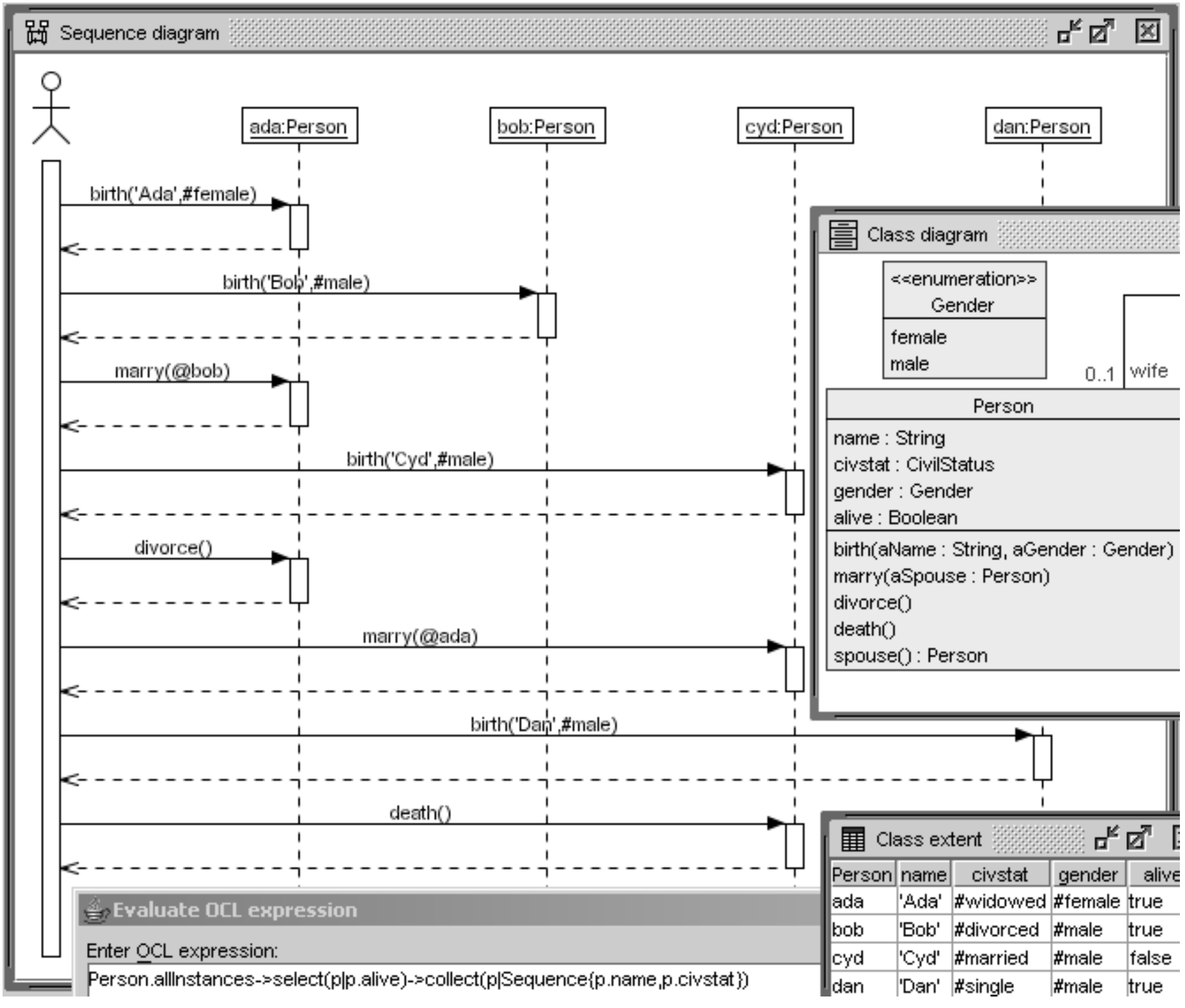
```
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})
```

Result:

```
Bag{Sequence{'Ada',#widowed},Sequence{'Bob',#divorced},Sequence{'Dan',#single}} : Bag(Sequence(OclAny))
```

Evaluate | Clear Result | Close

Ready.


Sequence diagram
ada:Person
bob:Person
cyd:Person
dan:Person
birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()
Class diagram
<<enumeration>>
Gender
female
male
0..1
wife
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
Class extent
Person | name | civstat | gender | alive
ada | 'Ada' | #widowed | #female | true
bob | 'Bob' | #divorced | #male | true
cyd | 'Cyd' | #married | #male | false
dan | 'Dan' | #single | #male | true
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})

```
procedure Person_marry(self:Person,aSpouse:Person)
begin
[self].civstat:=[#married];
[aSpouse].civstat:=[#married];
if [self.gender=#female] then
  begin Insert(Marriage,[self],[aSpouse]); end
else -- [self.gender=#male]
  begin Insert(Marriage,[aSpouse],[self]); end;
end;
```

```
use> !openter ada marry(bob)
     precondition `aSpouseDefined' is true
     precondition `isAlive' is true
     precondition `aSpouseAlive' is true
     precondition `isUnmarried' is true
     precondition `aSpouseUnmarried' is true
     precondition `differentGenders' is true
use> gen start civstat.assl Person_marry(ada,bob)
use> gen result
     Random number generator was initialized with 8047.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @ada.civstat := #married
     !set @bob.civstat := #married
     !insert (ada,bob) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `isMarried' is true
     postcondition `femaleHasMarriedHusband' is true
     postcondition `maleHasMarriedWife' is true
```

Class extent

Person	name	civstat	gender	alive
ada	'Ada'	#widowed	#female	true
bob	'Bob'	#divorced	#male	true
cyd	'Cyd'	#married	#male	false
dan	'Dan'	#single	#male	true

Evaluate OCL expression

Enter OCL expression:

```
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})
```

ada	'Ada'	#widowed	#female	true
bob	'Bob'	#divorced	#male	true
cyd	'Cyd'	#married	#male	false
dan	'Dan'	#single	#male	true

Evaluate

Clear Result

Close

Result:

```
Bag{Sequence{'Ada',#widowed},Sequence{'Bob',#divorced},Sequence{'Dan',#single}} : Bag(Sequence(OclAny))
```

USE: c:/mg/civstat/scp-sub/civstat.use

File Edit State View Help

- CivilStatusWorld
  - Classes
    - Person
  - Associations
    - Marriage
  - Invariants
    - Person::attributesDefined
    - Person::nameCapitalThenSmallLetters
    - Person::nameIsUnique
    - Person::femaleHasNoWife
    - Person::maleHasNoHusband
  - Pre-/Postconditions
    - pre birth::freshUnlinkedPerson
    - post birth::nameAssigned
    - post birth::civstatAssigned
    - post birth::genderAssigned
    - post birth::isAliveAssigned
    - pre marry::aSpouseDefined
    - pre marry::isAlive
    - pre marry::aSpouseAlive
    - pre marry::isUnmarried
    - pre marry::aSpouseUnmarried
    - pre marry::differentGenders
    - post marry::isMarried
    - post marry::femaleHasMarriedHusband
    - post marry::maleHasMarriedWife
    - pre divorce::isMarried
    - pre divorce::isAlive
    - pre divorce::husbandAlive
    - pre divorce::wifeAlive
    - post divorce::isDivorced
    - post divorce::husbandDivorced
    - post divorce::wifeDivorced
    - pre death::isAlive
    - post death::notAlive
    - post death::husbandWidowed
    - post death::wifeWidowed

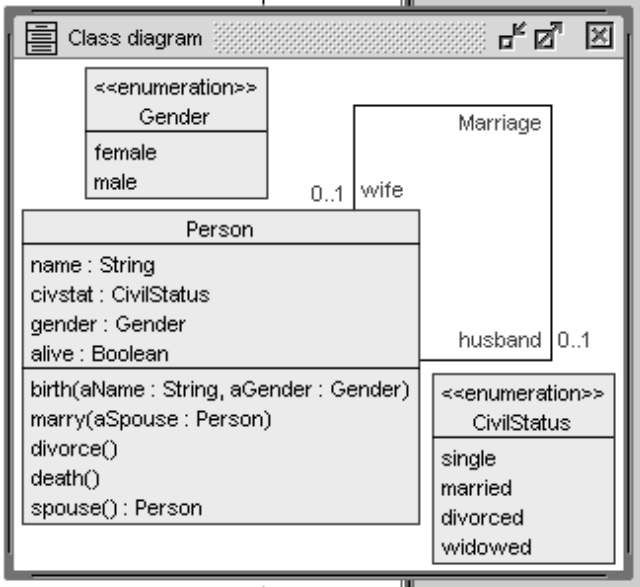
context Person::marry(aSpouse : Person)
pre differentGenders: (self.gender <> aSpouse.gender)

Sequence diagram

ada:Person | bob:Person | cyd:Person | dan:Person

birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()

Class diagram



Class extent

Person	name	civstat	gender	alive
ada	'Ada'	#widowed	#female	true
bob	'Bob'	#divorced	#male	true
cyd	'Cyd'	#married	#male	false
dan	'Dan'	#single	#male	true

Evaluate OCL expression

Enter OCL expression:

```
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})
```

Result:

```
Bag{Sequence{'Ada',#widowed},Sequence{'Bob',#divorced},Sequence{'Dan',#single}} : Bag(Sequence(OclAny))
```

Evaluate | Clear Result | Close

Ready.

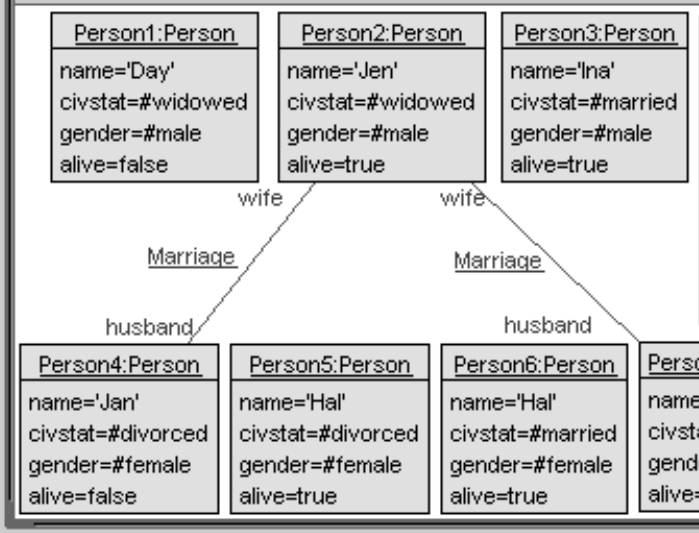
USE: c:/mg/civstat/scp-sub/civstat.use

File Edit State View Help

- CivilStatusWorld
  - Classes
    - Person
  - Associations
    - Marriage
  - Invariants
    - Person::attributesDefined
    - Person::nameCapitalThenSmallLetters
    - Person::nameIsUnique
    - Person::femaleHasNoWife
    - Person::maleHasNoHusband
  - Pre-/Postconditions
    - pre birth::freshUnlinkedPerson
    - post birth::nameAssigned
    - post birth::civstatAssigned
    - post birth::genderAssigned
    - post birth::isAliveAssigned
    - pre marry::aSpouseDefined
    - pre marry::isAlive
    - pre marry::aSpouseAlive
    - pre marry::isUnmarried
    - pre marry::aSpouseUnmarried
    - pre marry::differentGenders
    - post marry::isMarried
    - post marry::femaleHasMarriedHusband
    - post marry::maleHasMarriedWife
    - pre divorce::isMarried
    - pre divorce::isAlive
    - pre divorce::husbandAlive
    - pre divorce::wifeAlive
    - post divorce::isDivorced

```
context Person inv maleHasNoHusband:
  ((self.gender = #male) implies
self.husband.isUndefined)
```

Object diagram



Class invariants

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	false
Person::maleHasNoHusband	n/a
Person::nameCapitalThenSmallLetters	true
Person::nameIsUnique	false

2 constraints failed. 100%

Evaluation browser

- Person.allInstances->forAll(self : Person | ((self.gender = #female) implies self.wife.isUndefined)) = false
  - Person.allInstances = Set{@Person1,@Person2,@Person3,@Person4,@Person5,@Person6,@Person7}
  - ((self.gender = #female) implies self.wife.isUndefined) = true
  - ((self.gender = #female) implies self.wife.isUndefined) = true
  - ((self.gender = #female) implies self.wife.isUndefined) = true
  - ((self.gender = #female) implies self.wife.isUndefined) = false
    - (self.gender = #female) = true
      - self.gender = #female
        - self = @Person4
      - #female = #female
    - self.wife.isUndefined = false
      - self.wife = @Person2
        - self = @Person4

Close

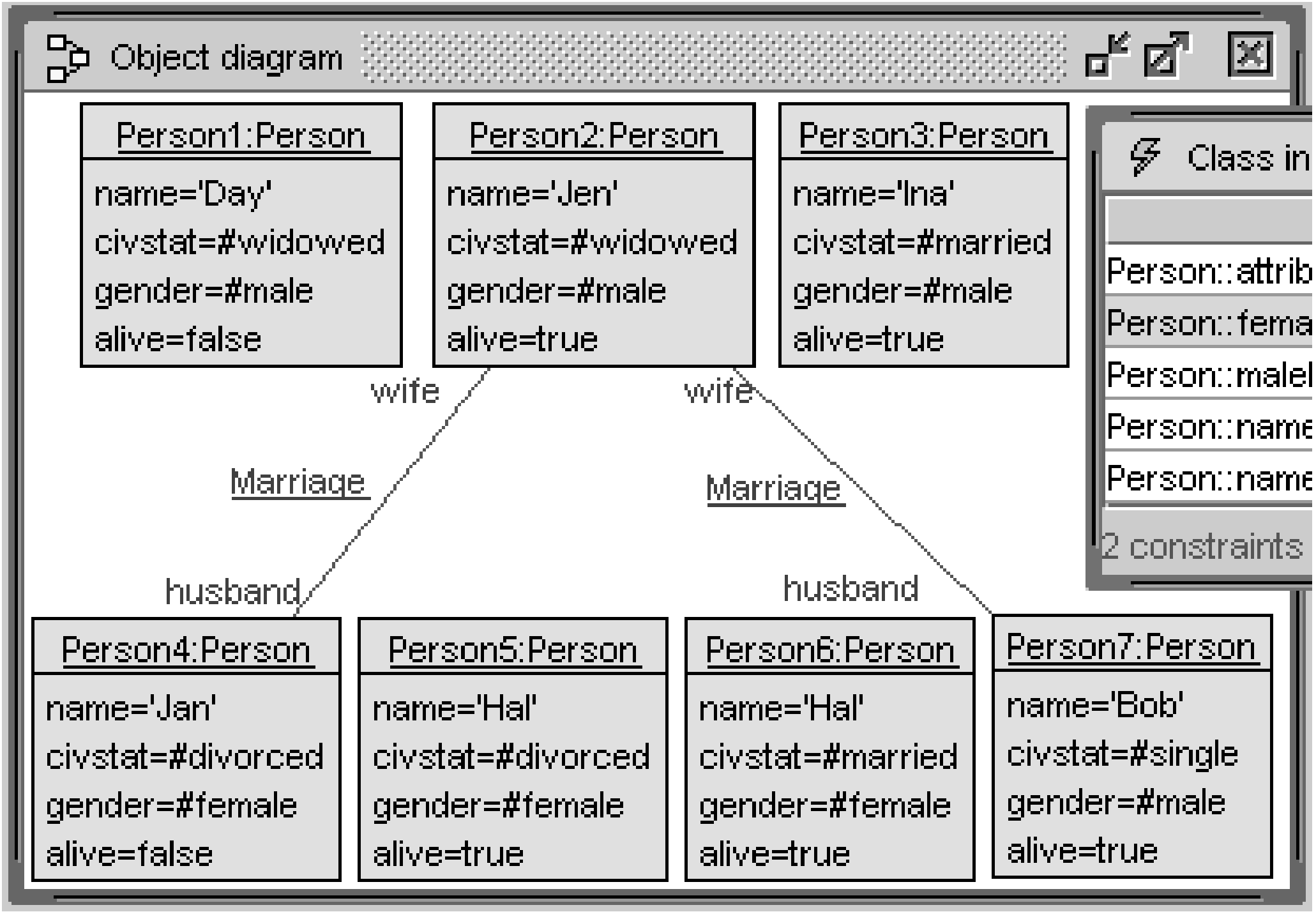
Log

```
checking structure...
Multiplicity constraint violation in association `Marriage':
 Object `Person2' of class `Person' is connected to 2 objects of class `Person'
 but the multiplicity is specified as `0..1'.
checking structure, found errors.
```

Ready.

```
procedure crowd(numFem:Integer, numMale:Integer, numMarr:Integer)
var theFemales: Sequence(Person), theMales: Sequence(Person),
    f: Person, m: Person;
begin
  theFemales:=CreateN(Person,[numFem]);
  theMales:=CreateN(Person,[numMale]);
  for i:Integer in [Sequence{1..numFem}] begin
    [theFemales->at(i)].name:=Any([Sequence{'Ada','Bel','Cam',
      'Day','Eva','Flo','Gen','Hao','Ina','Jen'}]);
    [theFemales->at(i)].civstat:=
      Any([Sequence{#single,#married,#divorced,#widowed}]);
    [theFemales->at(i)].gender:=Any([Sequence{#female,#male}]);
    [theFemales->at(i)].alive:=Any([Sequence{false,true}]);
    end;
  for i:Integer in [Sequence{1..numMale}] begin
    ... end;
  for i:Integer in [Sequence{1..numMarr}] begin
    f:=Any([theFemales]); m:=Any([theMales]);
    Insert(Marriage,[f],[m]);
    end;
end;
```

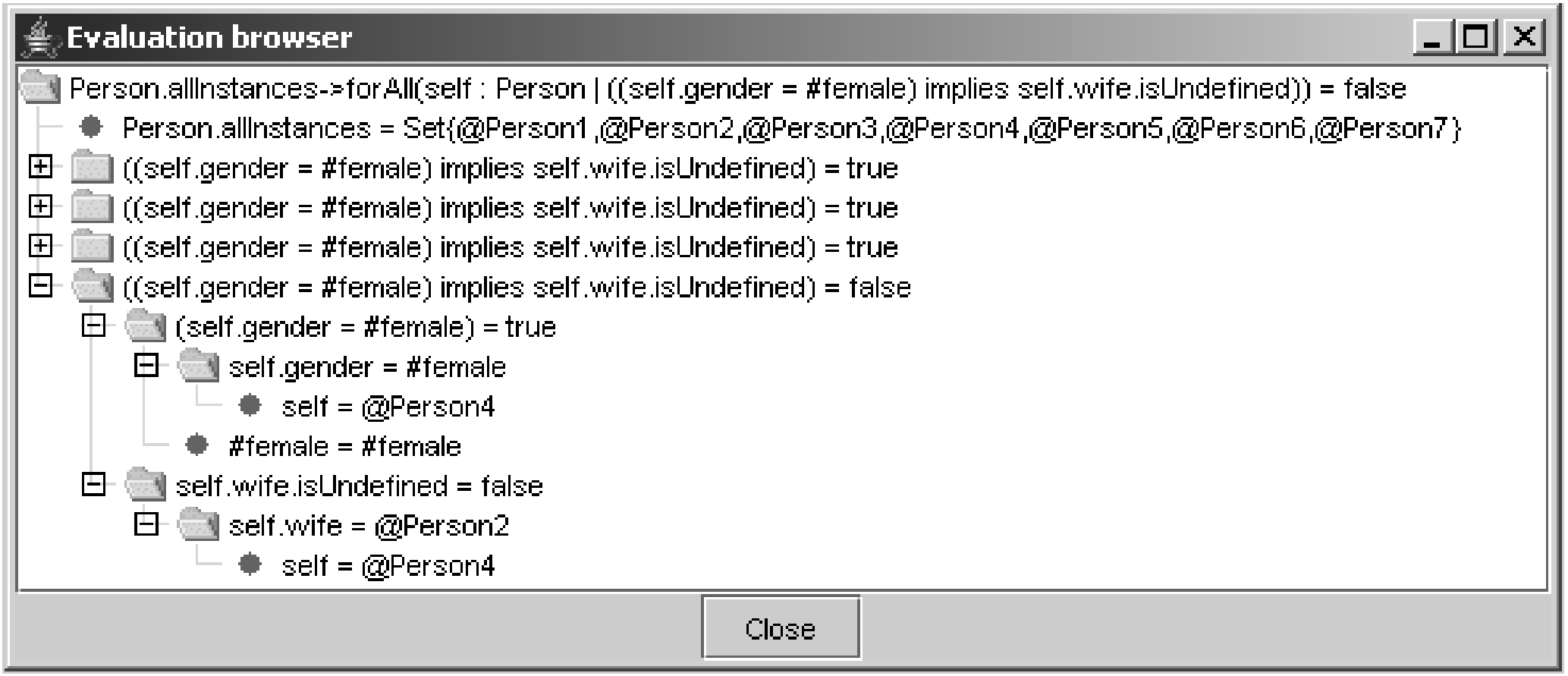
crowd(3,4,2)


Object diagram
Person1:Person
name='Day'
civstat=#widowed
gender=#male
alive=false
Person2:Person
name='Jen'
civstat=#widowed
gender=#male
alive=true
Person3:Person
name='Ina'
civstat=#married
gender=#male
alive=true
wife
wife
Marriage
Marriage
husband
husband
Person4:Person
name='Jan'
civstat=#divorced
gender=#female
alive=false
Person5:Person
name='Hal'
civstat=#divorced
gender=#female
alive=true
Person6:Person
name='Hal'
civstat=#married
gender=#female
alive=true
Person7:Person
name='Bob'
civstat=#single
gender=#male
alive=true
Class in
Person::attrib
Person::fema
Person::malel
Person::name
Person::name
2 constraints

Class invariants

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	false
Person::maleHasNoHusband	n/a
Person::nameCapitalThenSmallLetters	true
Person::namelsUnique	false

2 constraints failed. 100%


Evaluation browser
Person.allInstances->forAll(self : Person | ((self.gender = #female) implies self.wife.isUndefined)) = false
Person.allInstances = Set{@Person1,@Person2,@Person3,@Person4,@Person5,@Person6,@Person7}
((self.gender = #female) implies self.wife.isUndefined) = true
((self.gender = #female) implies self.wife.isUndefined) = true
((self.gender = #female) implies self.wife.isUndefined) = true
((self.gender = #female) implies self.wife.isUndefined) = false
(self.gender = #female) = true
self.gender = #female
self = @Person4
#female = #female
self.wife.isUndefined = false
self.wife = @Person2
self = @Person4
Close

Log

```
checking structure...
Multiplicity constraint violation in association `Marriage':
  Object `Person2' of class `Person' is connected to 2 objects of class `Person'
  but the multiplicity is specified as `0..1'.
checking structure, found errors.
```

Ready.

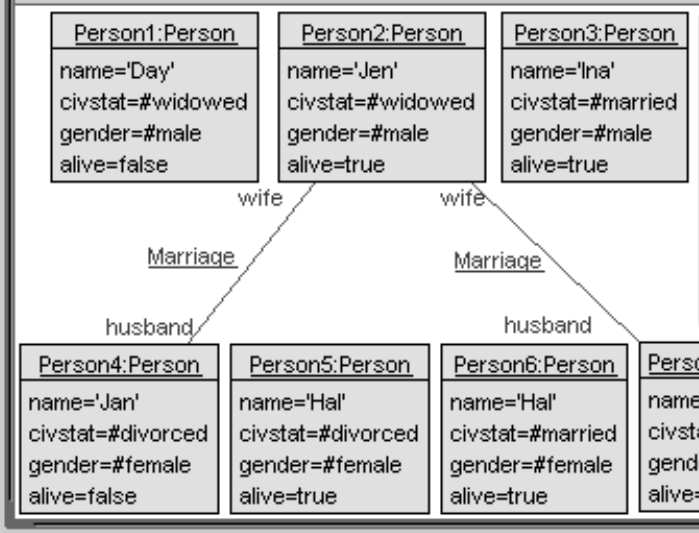
USE: c:/mg/civstat/scp-sub/civstat.use

File Edit State View Help

- CivilStatusWorld
  - Classes
    - Person
  - Associations
    - Marriage
  - Invariants
    - Person::attributesDefined
    - Person::nameCapitalThenSmallLetters
    - Person::nameIsUnique
    - Person::femaleHasNoWife
    - Person::maleHasNoHusband
  - Pre-/Postconditions
    - pre birth::freshUnlinkedPerson
    - post birth::nameAssigned
    - post birth::civstatAssigned
    - post birth::genderAssigned
    - post birth::isAliveAssigned
    - pre marry::aSpouseDefined
    - pre marry::isAlive
    - pre marry::aSpouseAlive
    - pre marry::isUnmarried
    - pre marry::aSpouseUnmarried
    - pre marry::differentGenders
    - post marry::isMarried
    - post marry::femaleHasMarriedHusband
    - post marry::maleHasMarriedWife
    - pre divorce::isMarried
    - pre divorce::isAlive
    - pre divorce::husbandAlive
    - pre divorce::wifeAlive
    - post divorce::isDivorced

```
context Person inv maleHasNoHusband:
  ((self.gender = #male) implies
self.husband.isUndefined)
```

Object diagram



Class invariants

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	false
Person::maleHasNoHusband	n/a
Person::nameCapitalThenSmallLetters	true
Person::nameIsUnique	false

2 constraints failed. 100%

Evaluation browser

- Person.allInstances->forAll(self : Person | ((self.gender = #female) implies self.wife.isUndefined)) = false
  - Person.allInstances = Set{@Person1,@Person2,@Person3,@Person4,@Person5,@Person6,@Person7}
  - ((self.gender = #female) implies self.wife.isUndefined) = true
  - ((self.gender = #female) implies self.wife.isUndefined) = true
  - ((self.gender = #female) implies self.wife.isUndefined) = true
  - ((self.gender = #female) implies self.wife.isUndefined) = false
    - (self.gender = #female) = true
      - self.gender = #female
        - self = @Person4
      - #female = #female
    - self.wife.isUndefined = false
      - self.wife = @Person2
        - self = @Person4

Close

Log

```
checking structure...
Multiplicity constraint violation in association `Marriage':
 Object `Person2' of class `Person' is connected to 2 objects of class `Person'
 but the multiplicity is specified as `0..1'.
checking structure, found errors.
```

Ready.

```
use> gen load bigamy.invs
     Added invariants: Person::bigamy
use> gen start civstat.assl attemptBigamy()
use> gen result
     Random number generator was initialized with 5649.
     Checked 663552 snapshots. Result: No valid state found.
```

```
context Person inv bigamy: wife.isDefined and husband.isDefined
```

```
procedure attemptBigamy()
var p: Person, w: Person, h:Person, thePersons: Sequence(Person);
begin
  thePersons:=CreateN(Person,[3]);
  for i:Integer in [Sequence{1..3}] begin
    [thePersons->at(i)].name:=Try([Sequence{'A','B','C'}]);
    [thePersons->at(i)].civstat:=
      Try([Sequence{#single,#married,#divorced,#widowed}]);
    [thePersons->at(i)].gender:=Try([Sequence{#female,#male}]);
    [thePersons->at(i)].alive:=Try([Sequence{false,true}]);
    end;
  p:=Try([thePersons]); w:=Try([thePersons->excluding(p)]);
  h:=Try([thePersons->excluding(p)->excluding(w)]);
  Insert(Marriage,[w],[p]); Insert(Marriage,[p],[h]);
end;
```

663552 = 3 * 3 * 4 * 2 * 2 * 3 * 2 * 1

p w h

Thanks for your attention!