News from USE

Martin Gogolla
University of Bremen, Germany
Database Systems Group

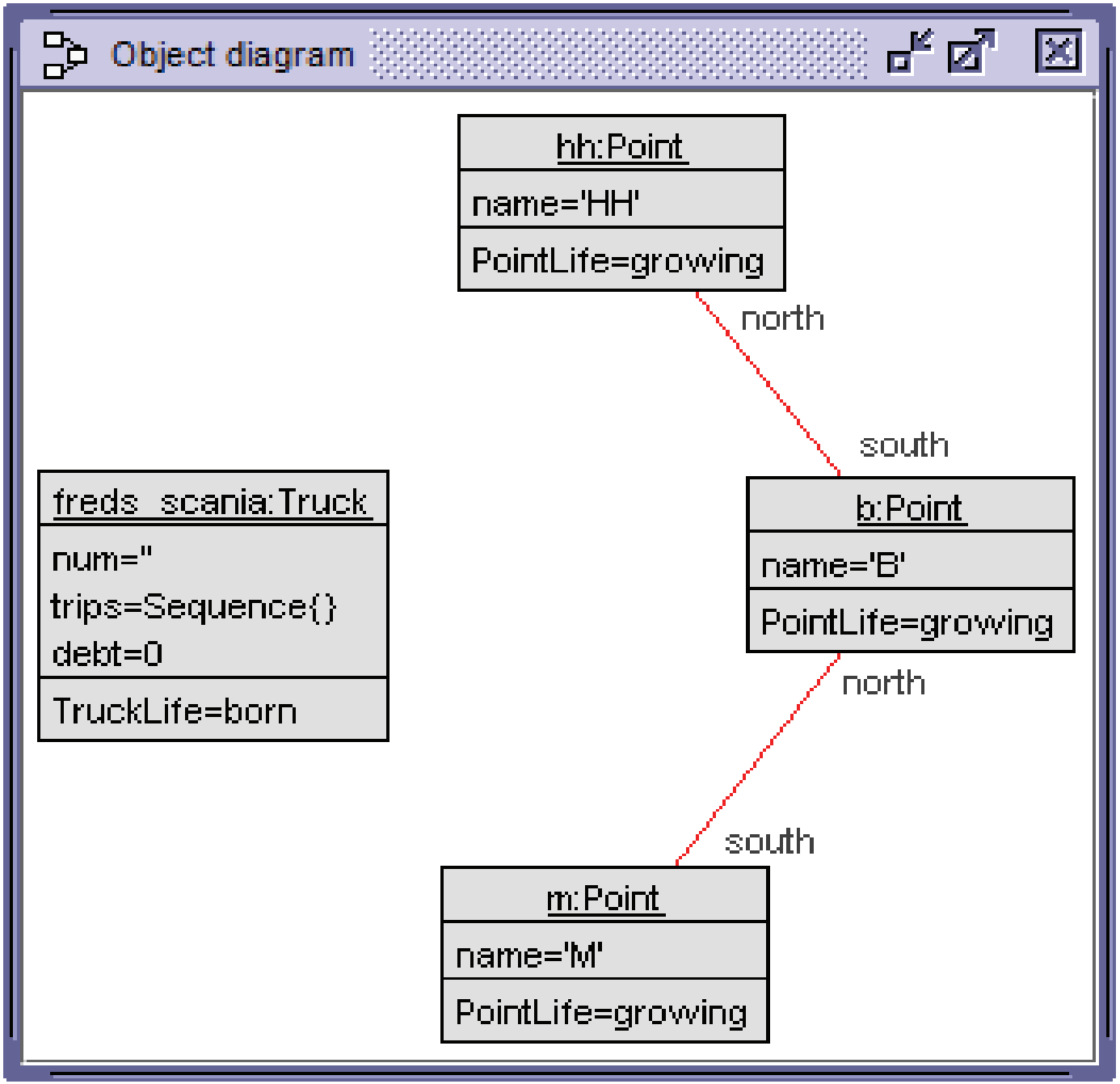
Tool USE (UML-based Specification Environment)

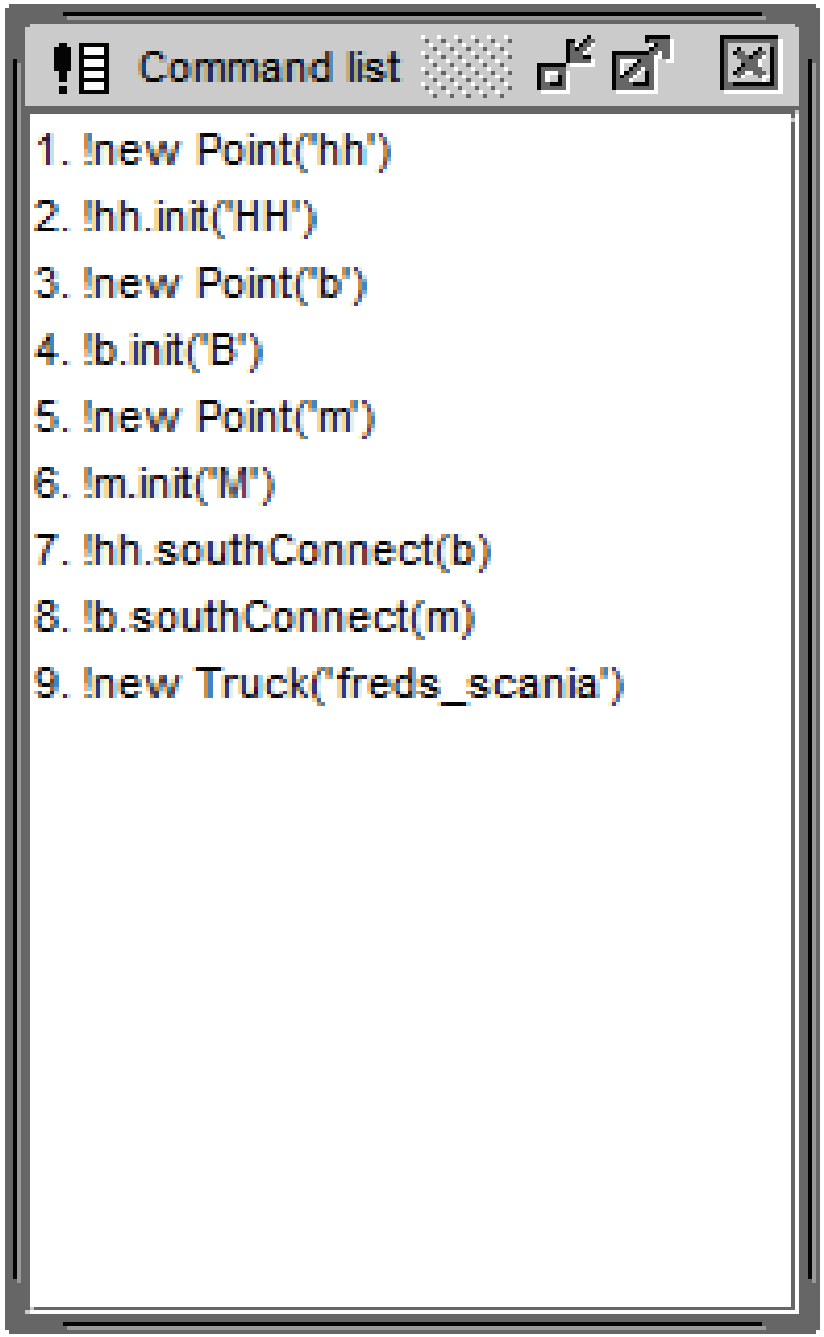
- Validation and verification tool for UML and OCL models
- UML class, statechart, object, sequence, and communication diagrams
- OCL support for
  + class invariants
  + operation pre- and postconditions
  + query operations and ad-hoc queries
  + derivation rules for attributes and associations
  + state invariants, transition guards and transition postconditions
- Imperative action language for implementing non-query operations on the model level: SOIL (Simple Ocl-based Imperative Language)
- Model validation by executing test scenarios
- Automatic generation of object diagrams through a model validator based on a translation of UML and OCL into relational logic (realized in Kodkod/Alloy) starting from a class diagram and invariants
- Verification of model properties like model consistency, model minimality or model state reachability

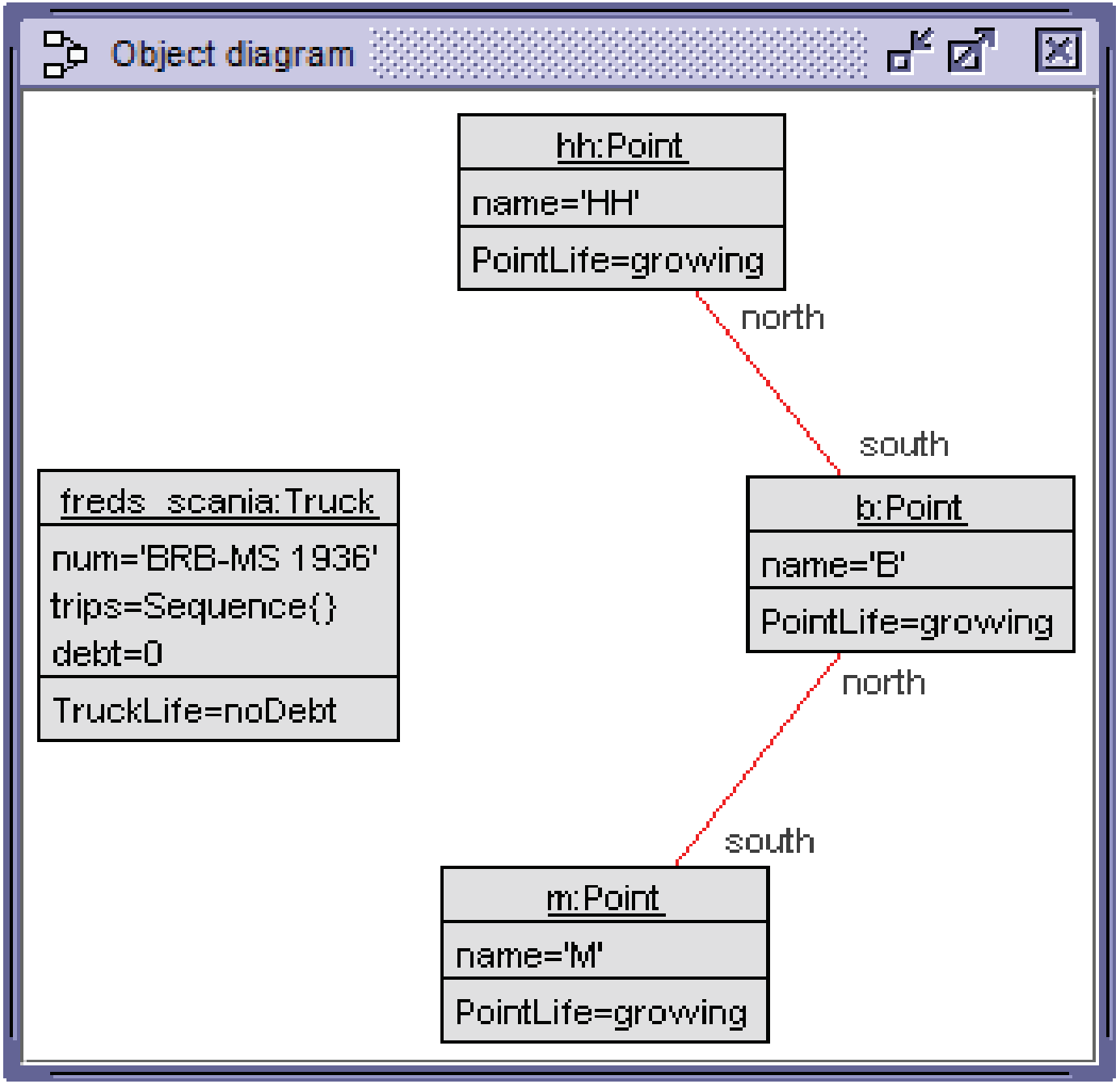
Validation ...

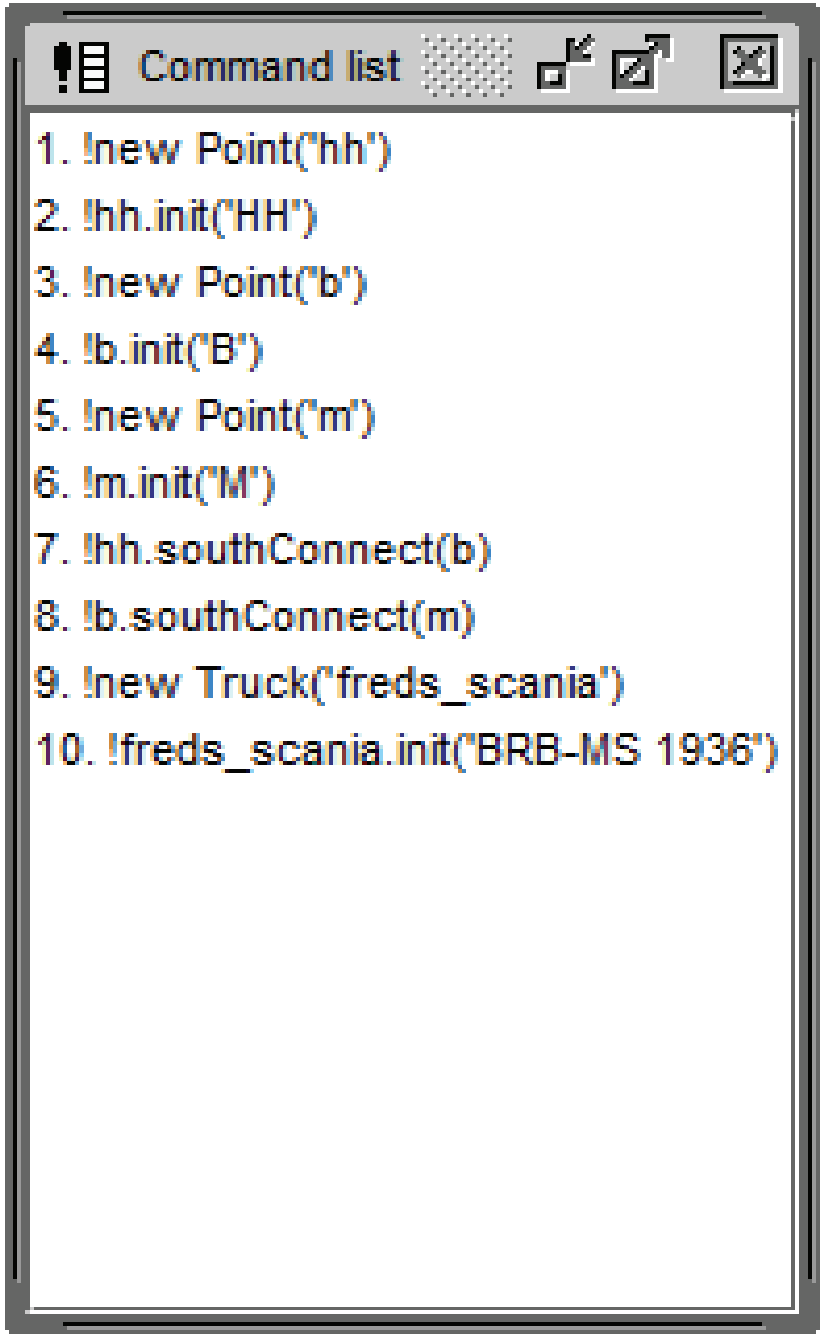
Example model: Toll Collect

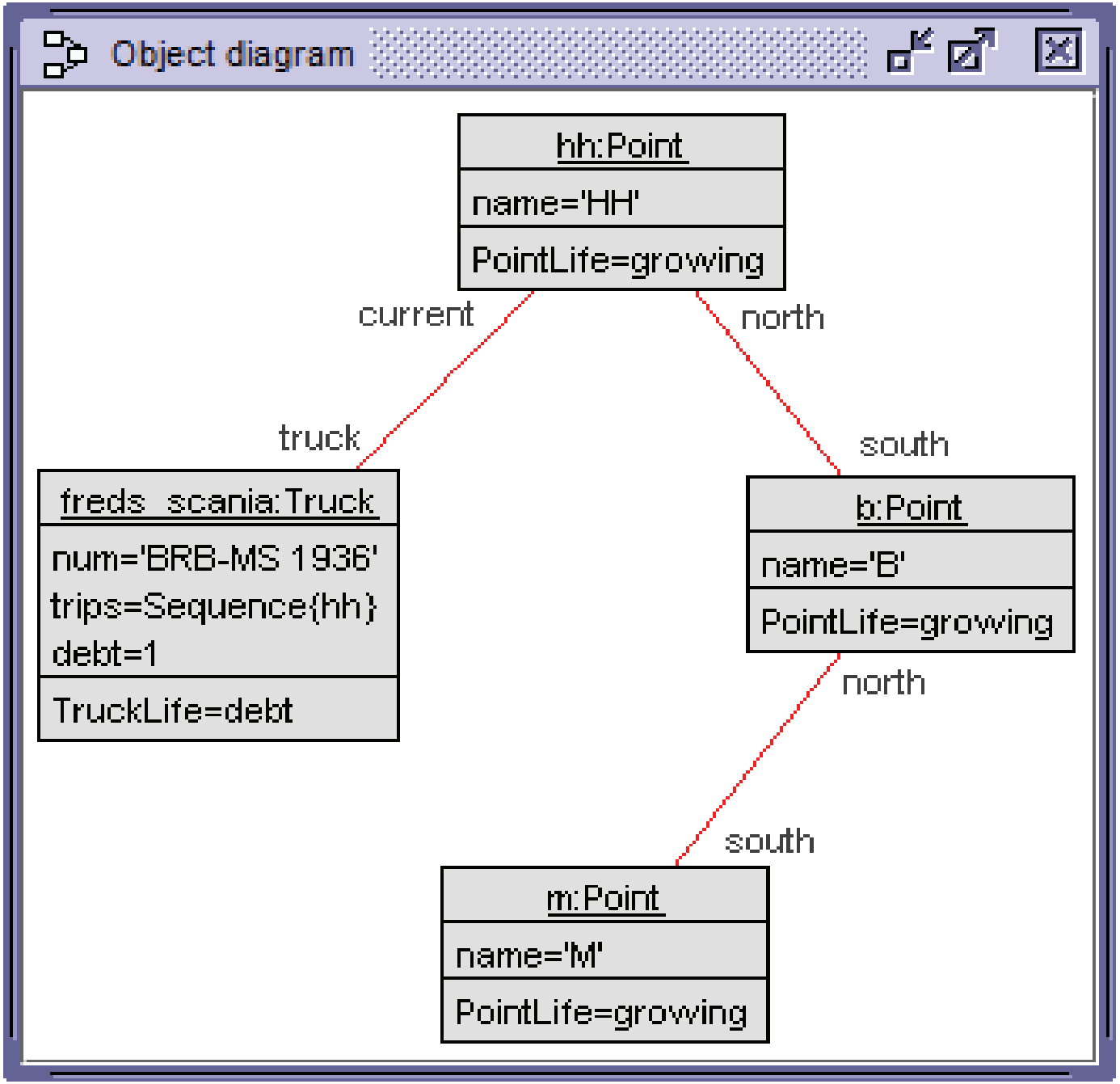
- Toll collection system for trucks on German motorways
- Very abstract version (2 classes, 2 associations)
- Class diagram with attributes and operations including invariants and operation pre- and postconditions
- Protocol state machine for each class with state invariants (transition pre- and postconditions not used)
- SOIL implementation for operations
- Validation: Construct example scenario where all operations are invoked and all invariants and operation pre- and postconditons are satisfied
- Scenario is explained and documented with object, sequence and communication diagrams
- Validation: Informal expectations checked against the formal system properties (diagrams, OCL queries, ...)

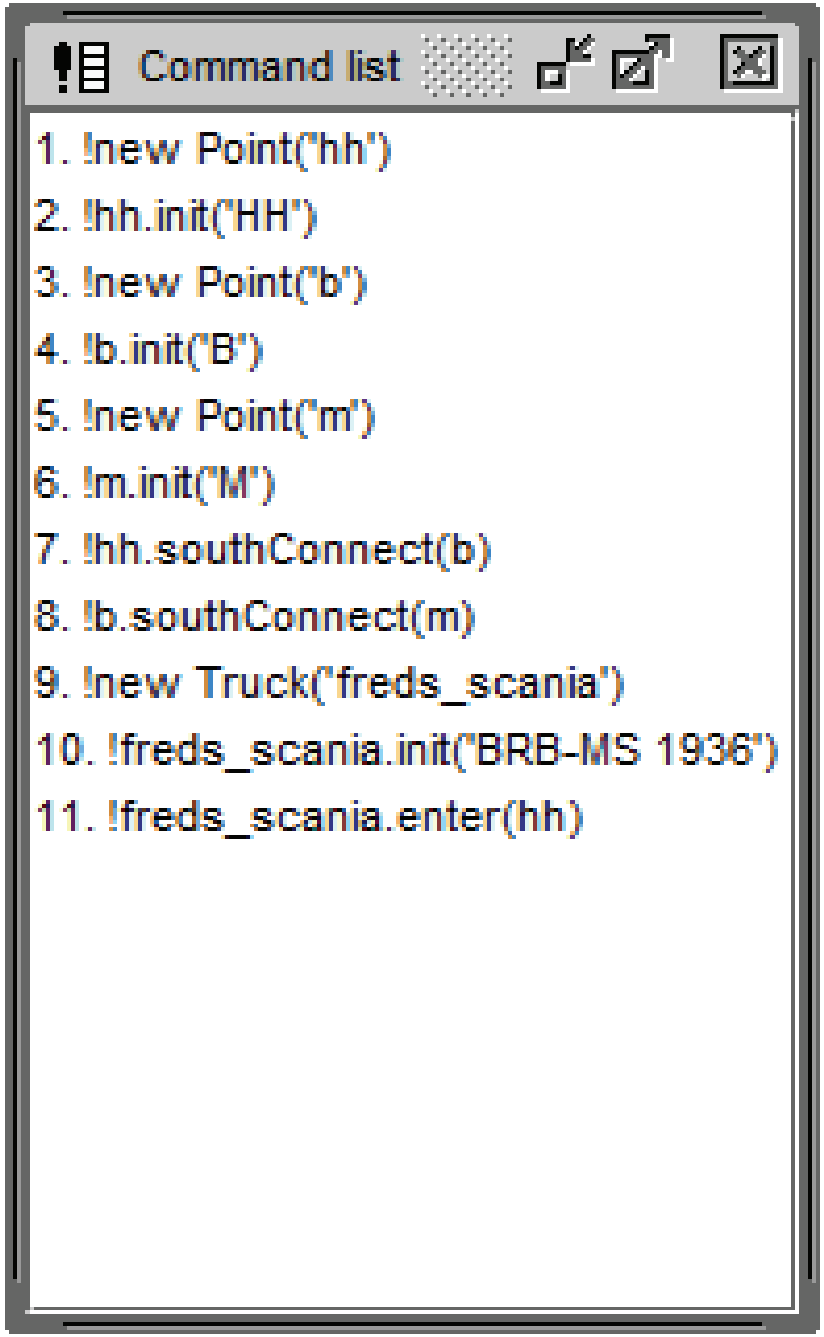

Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num=''
trips=Sequence{}
debt=0
TruckLife=born
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing

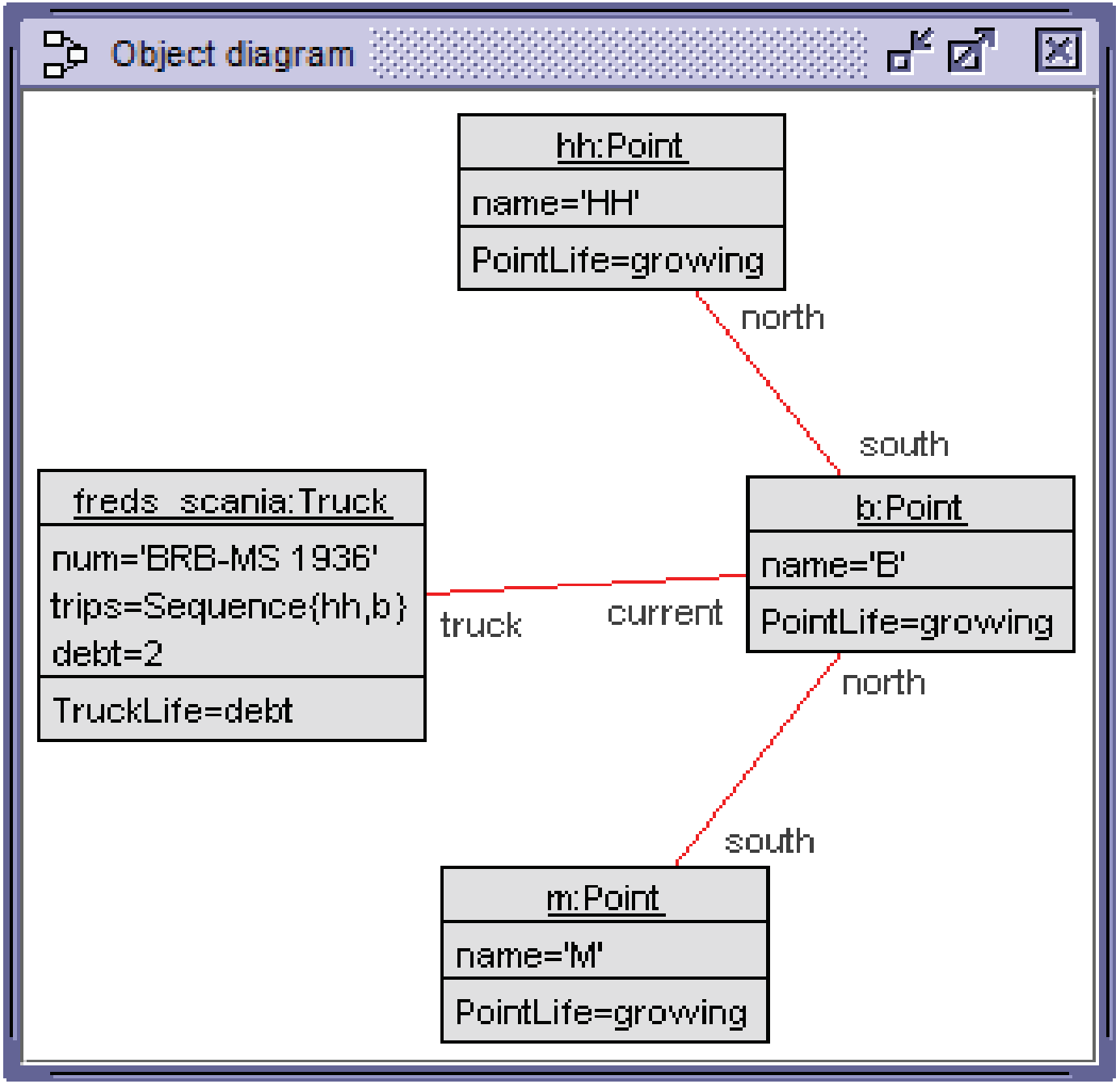

Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')

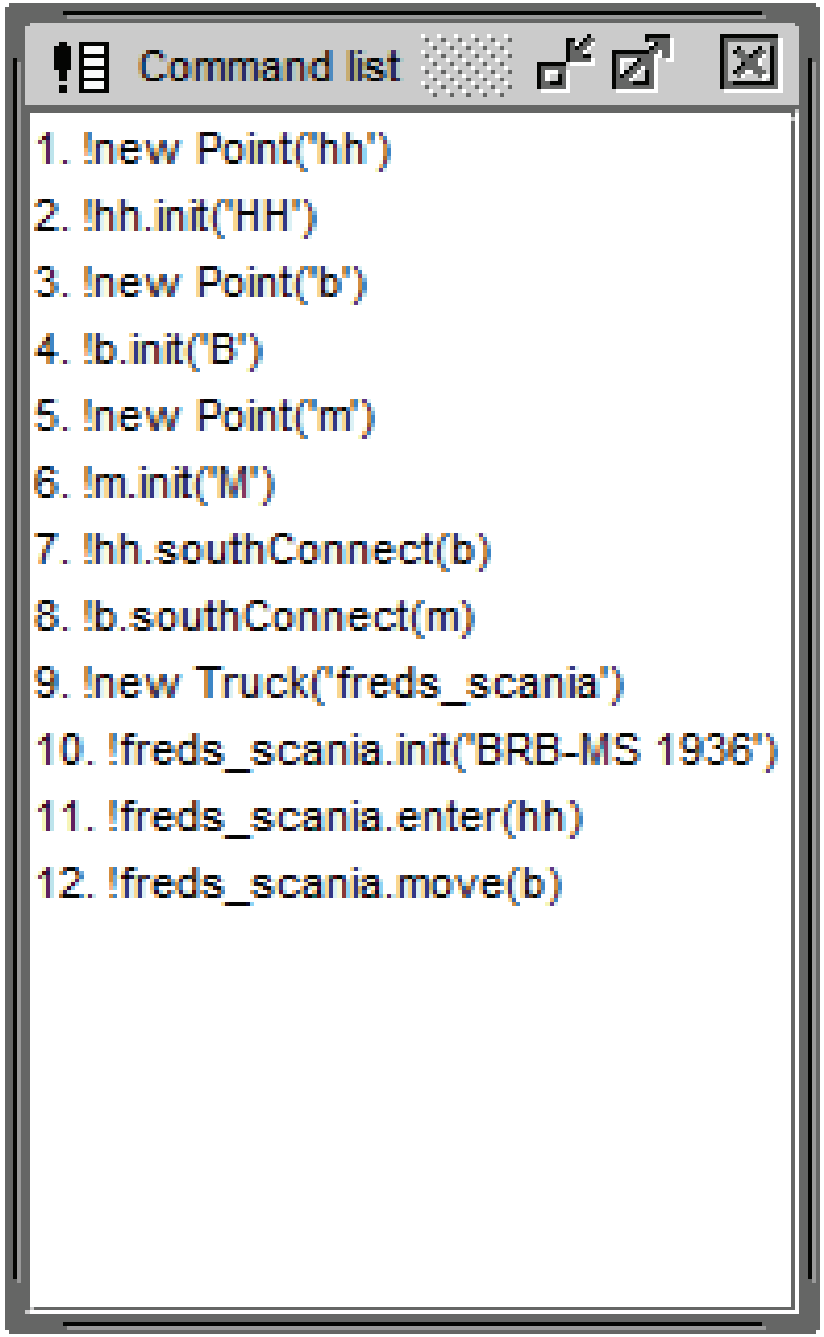

Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{}
debt=0
TruckLife=noDebt
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing

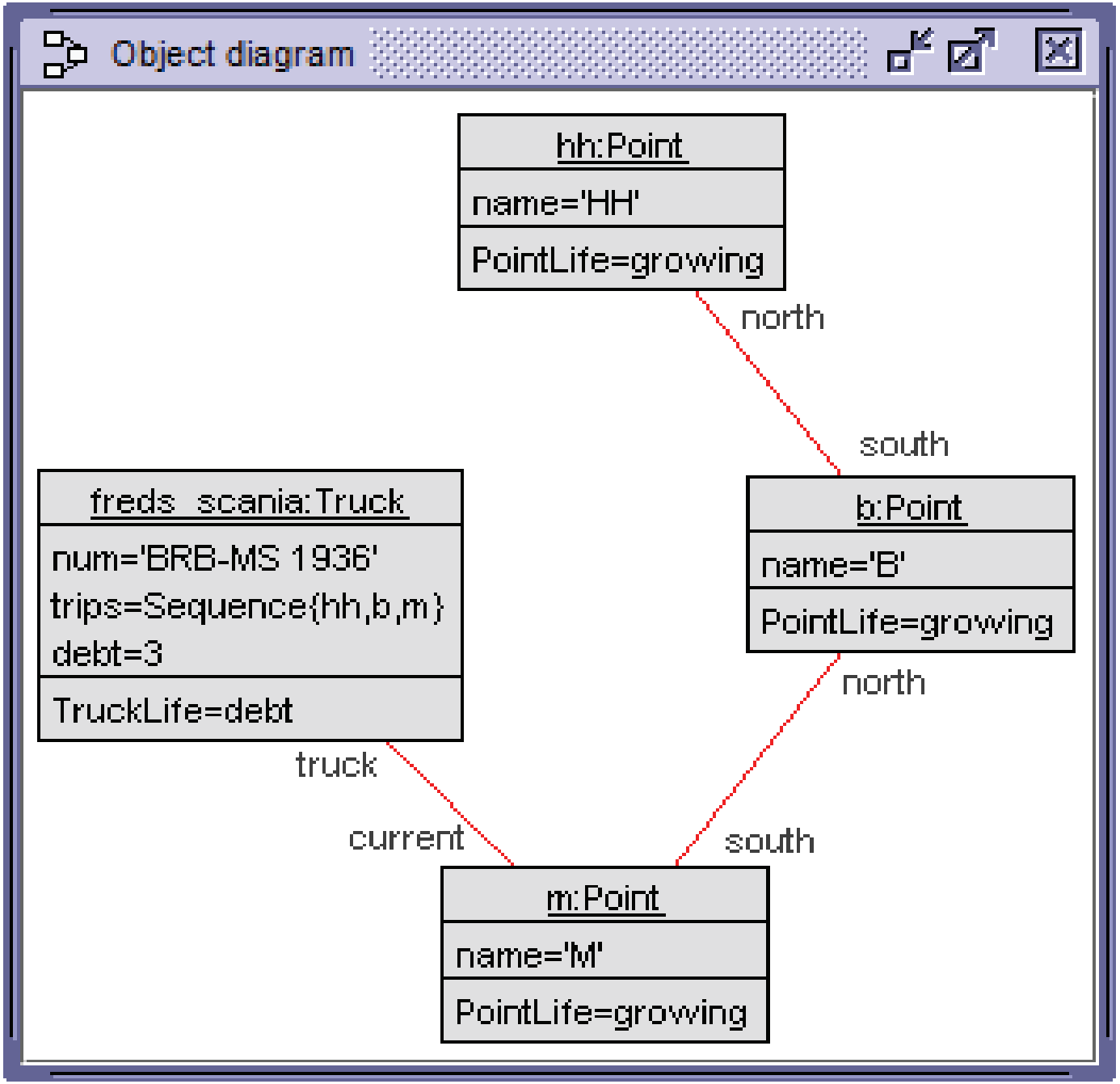

Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')

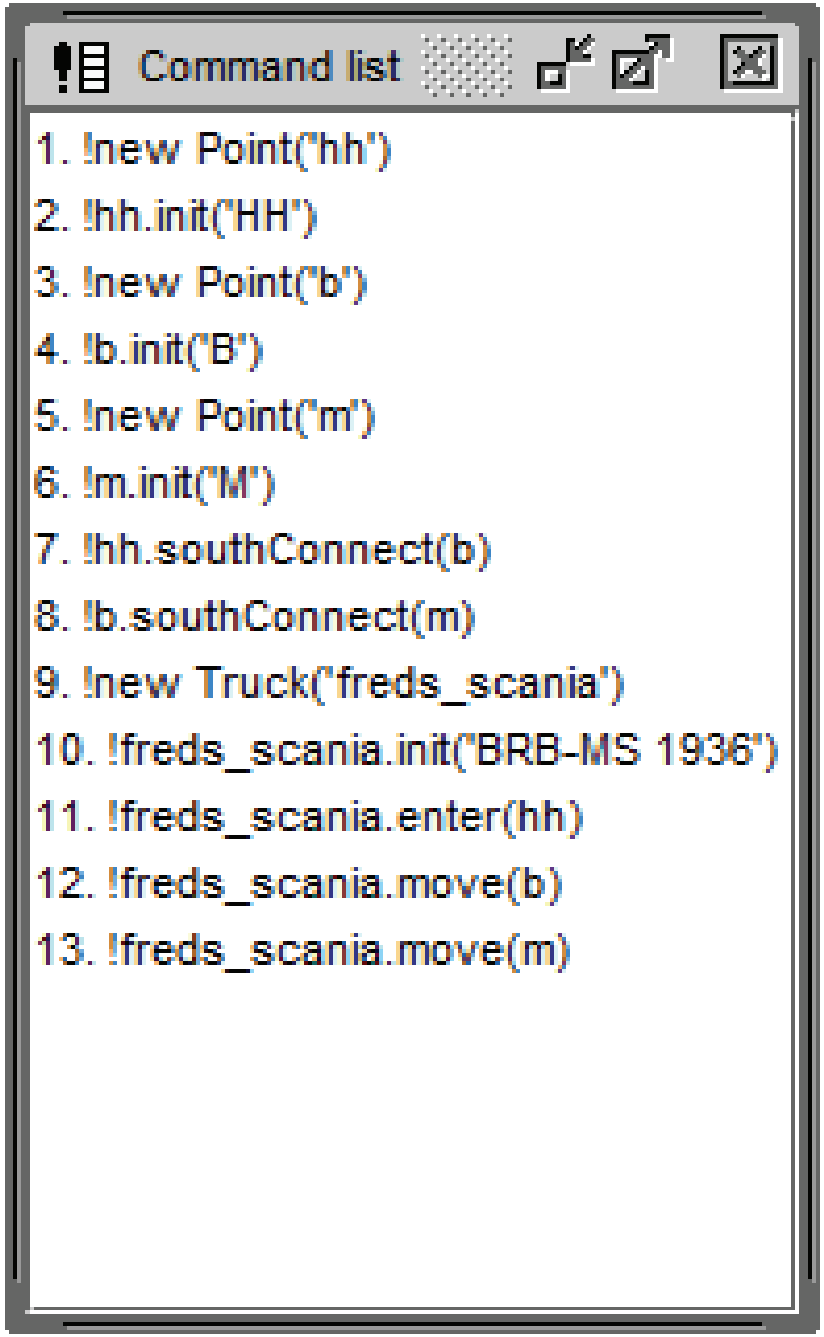

Object diagram
hh:Point
name='HH'
PointLife=growing
current
north
truck
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh}
debt=1
TruckLife=debt
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing

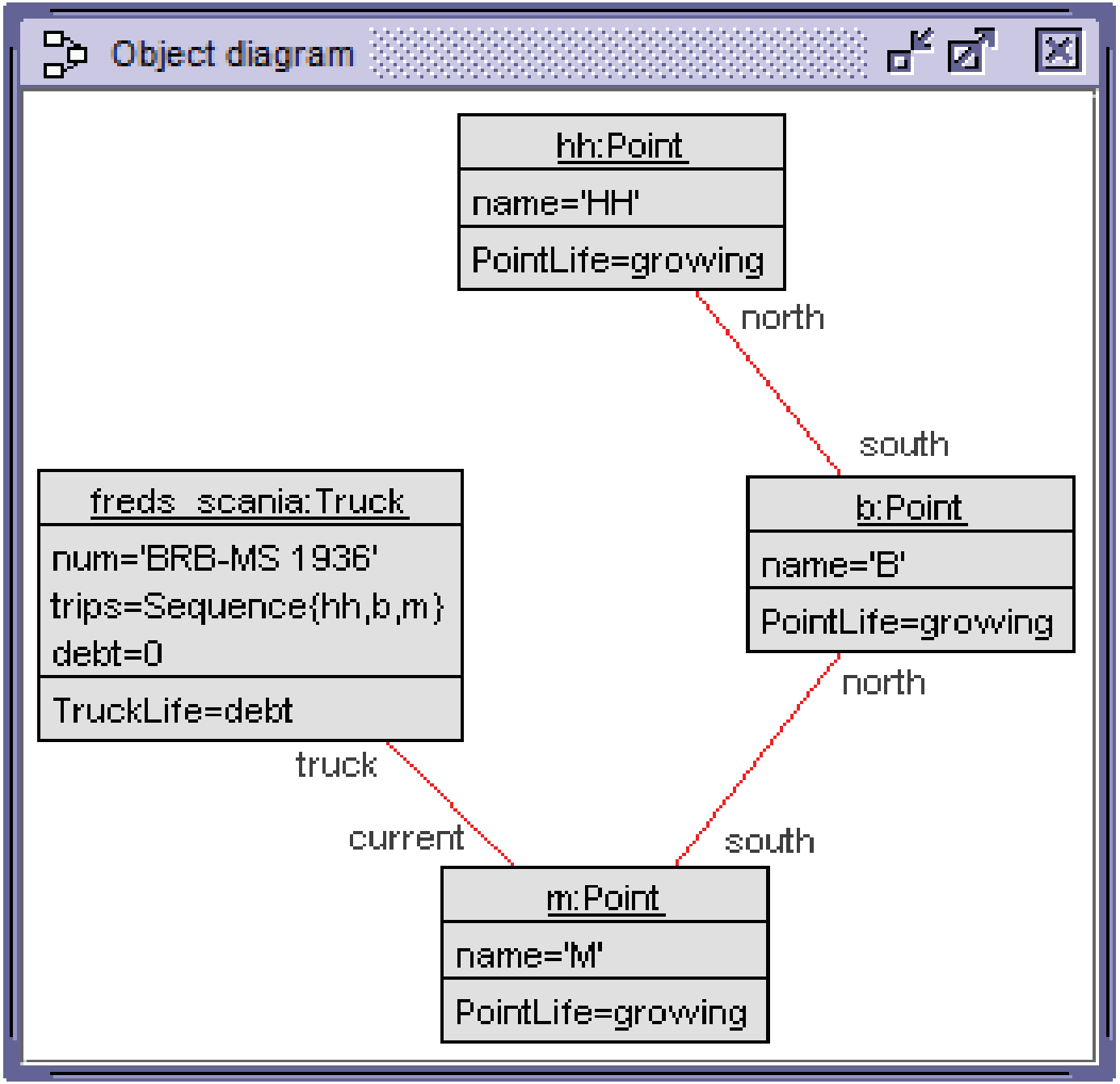

Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')
11. !freds_scania.enter(hh)

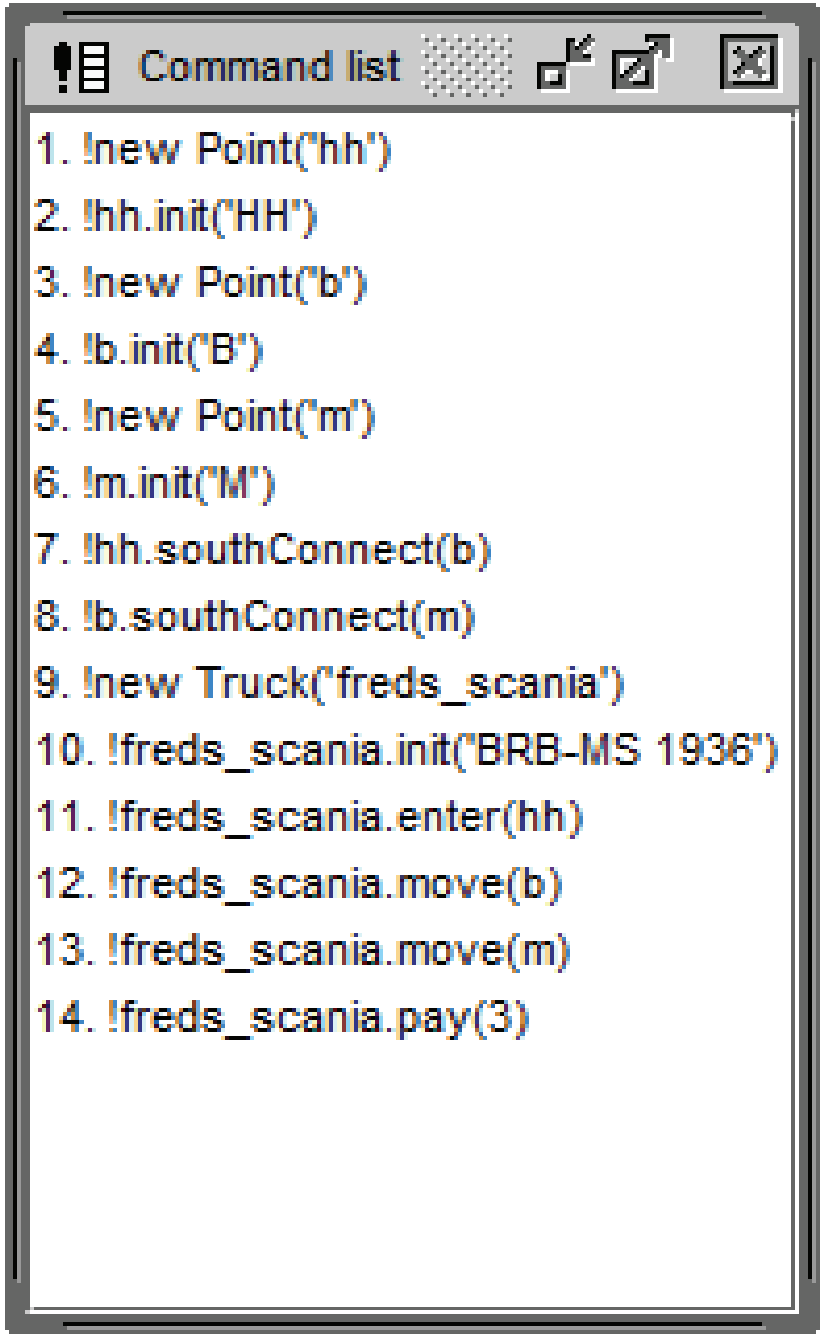

Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b}
debt=2
TruckLife=debt
truck
current
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing

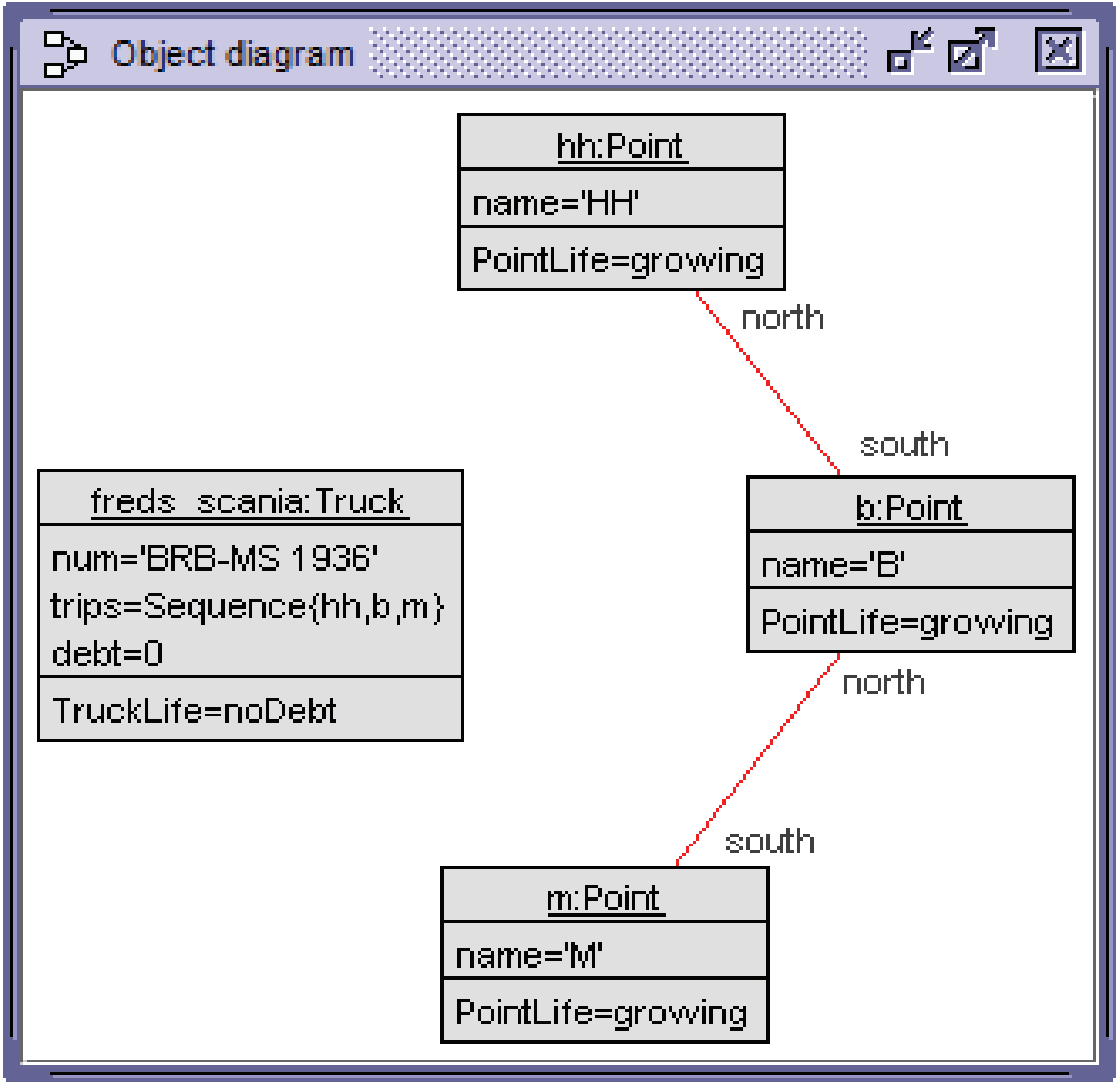

Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')
11. !freds_scania.enter(hh)
12. !freds_scania.move(b)

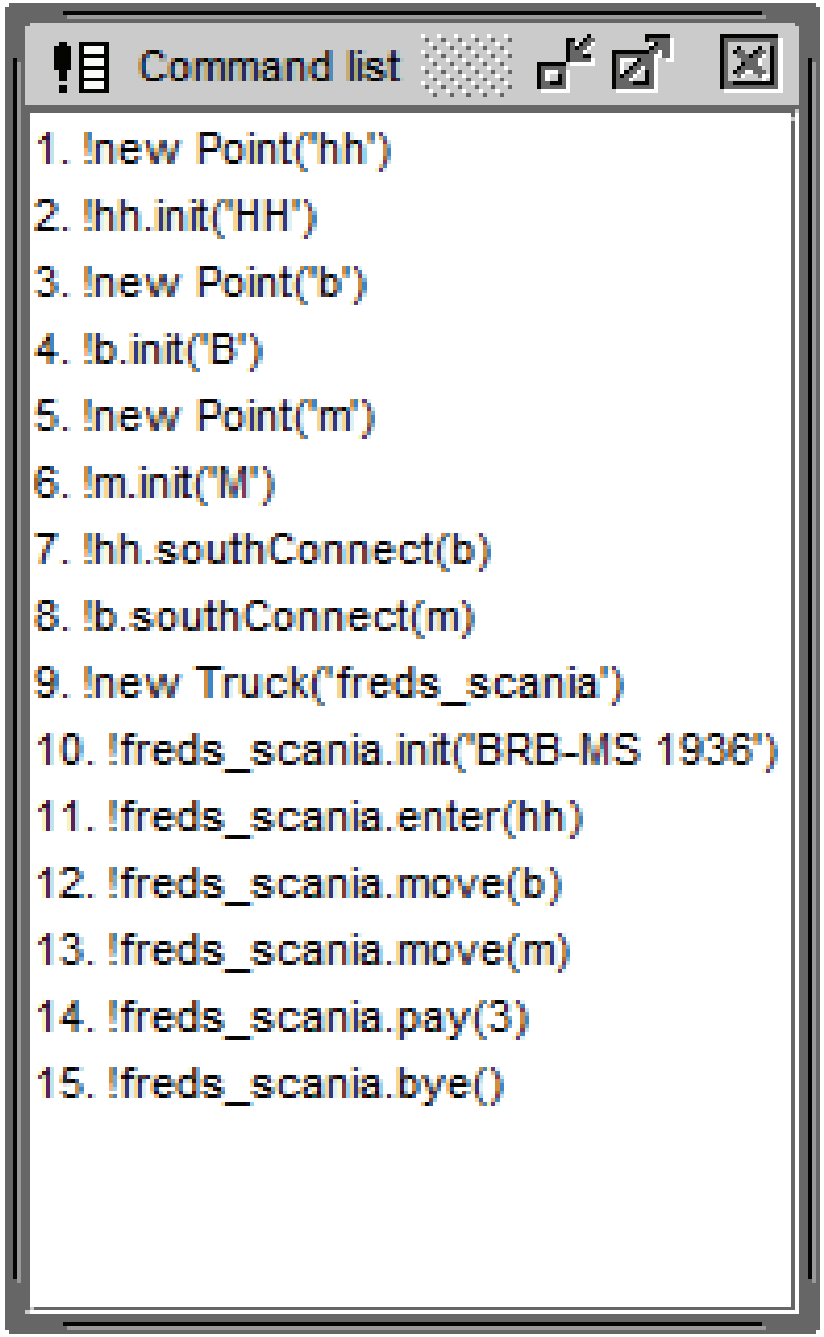

Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b,m}
debt=3
TruckLife=debt
b:Point
name='B'
PointLife=growing
north
truck
current
south
m:Point
name='M'
PointLife=growing

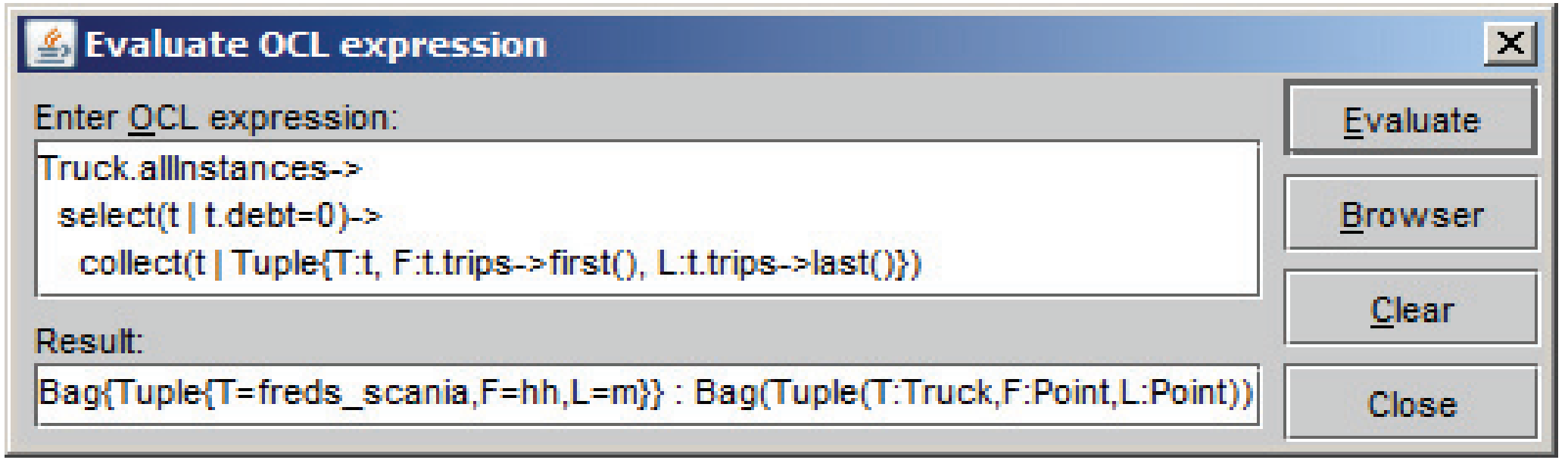

Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')
11. !freds_scania.enter(hh)
12. !freds_scania.move(b)
13. !freds_scania.move(m)

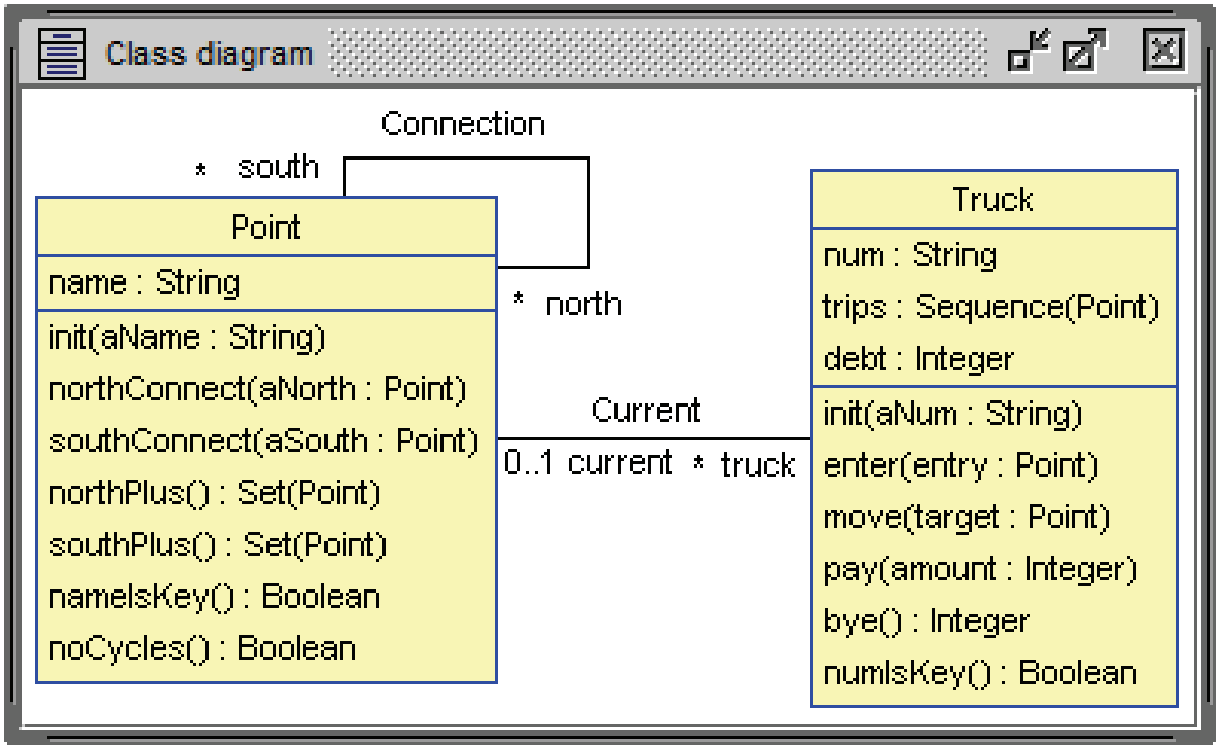

Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b,m}
debt=0
TruckLife=debt
b:Point
name='B'
PointLife=growing
north
truck
current
south
m:Point
name='M'
PointLife=growing

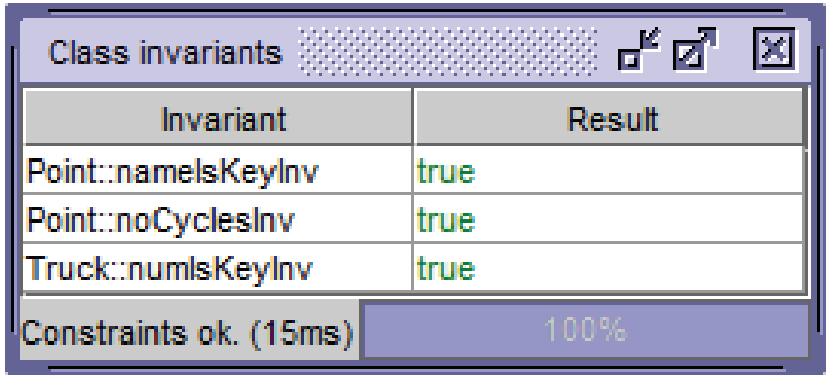

Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')
11. !freds_scania.enter(hh)
12. !freds_scania.move(b)
13. !freds_scania.move(m)
14. !freds_scania.pay(3)


Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b,m}
debt=0
TruckLife=noDebt
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing


Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')
11. !freds_scania.enter(hh)
12. !freds_scania.move(b)
13. !freds_scania.move(m)
14. !freds_scania.pay(3)
15. !freds_scania.bye()

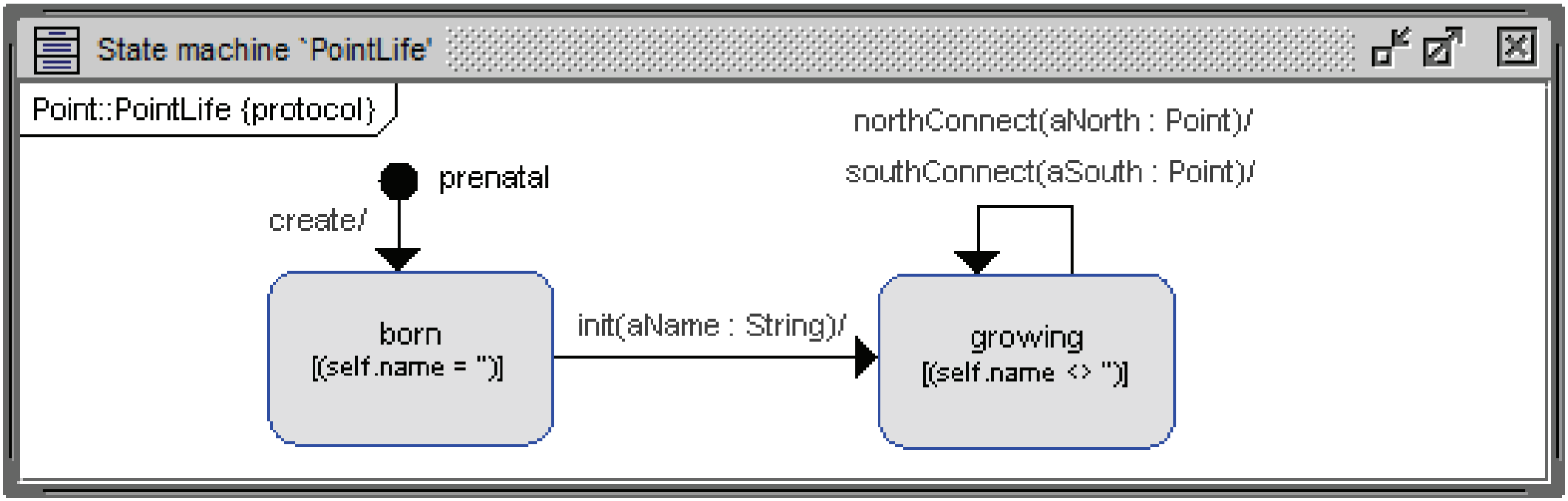

Evaluate OCL expression
Enter OCL expression:
Truck.allInstances->
select(t | t.debt=0)->
collect(t | Tuple{T:t, F:t.trips->first(), L:t.trips->last()})
Result:
Bag{Tuple{T=freds_scania,F=hh,L=m}} : Bag(Tuple(T:Truck,F:Point,L:Point))
Evaluate
Browser
Clear
Close

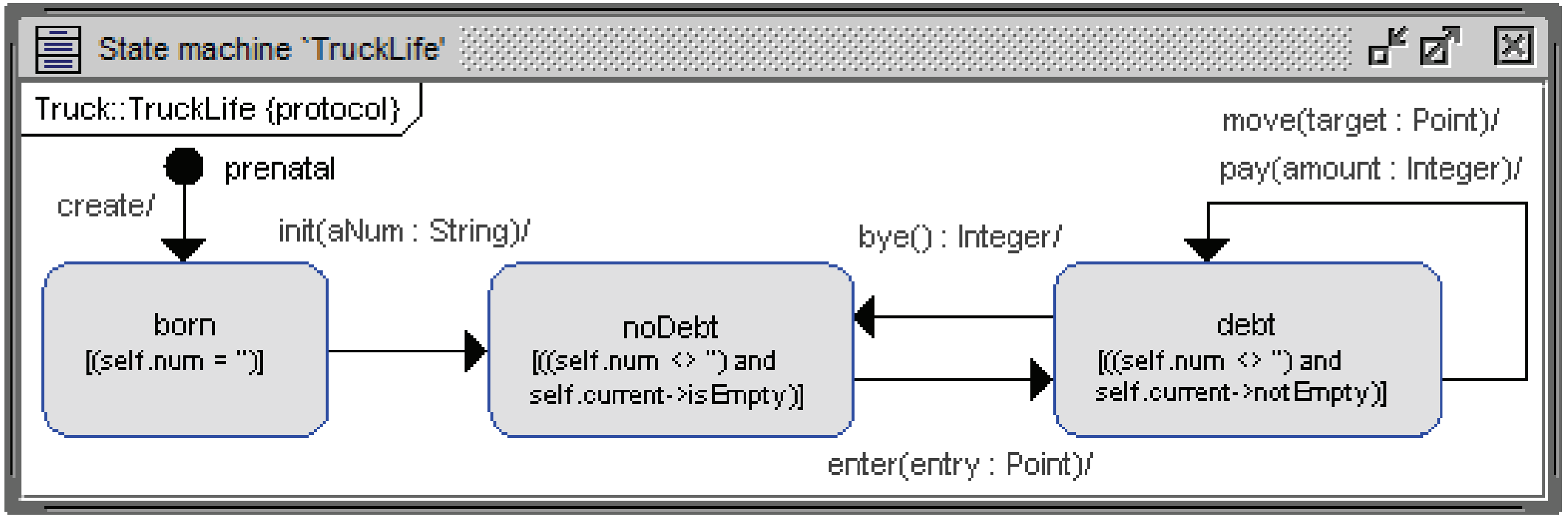


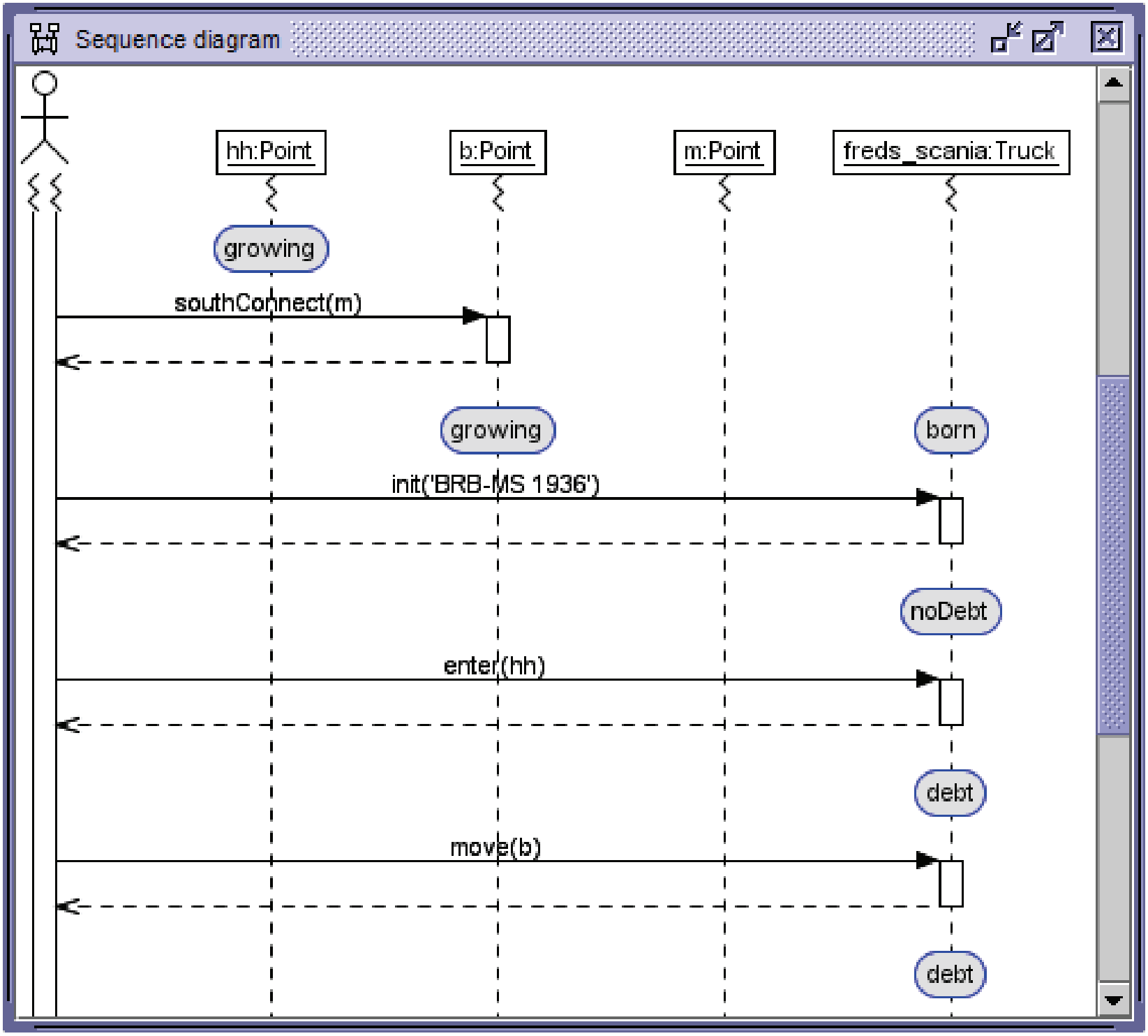


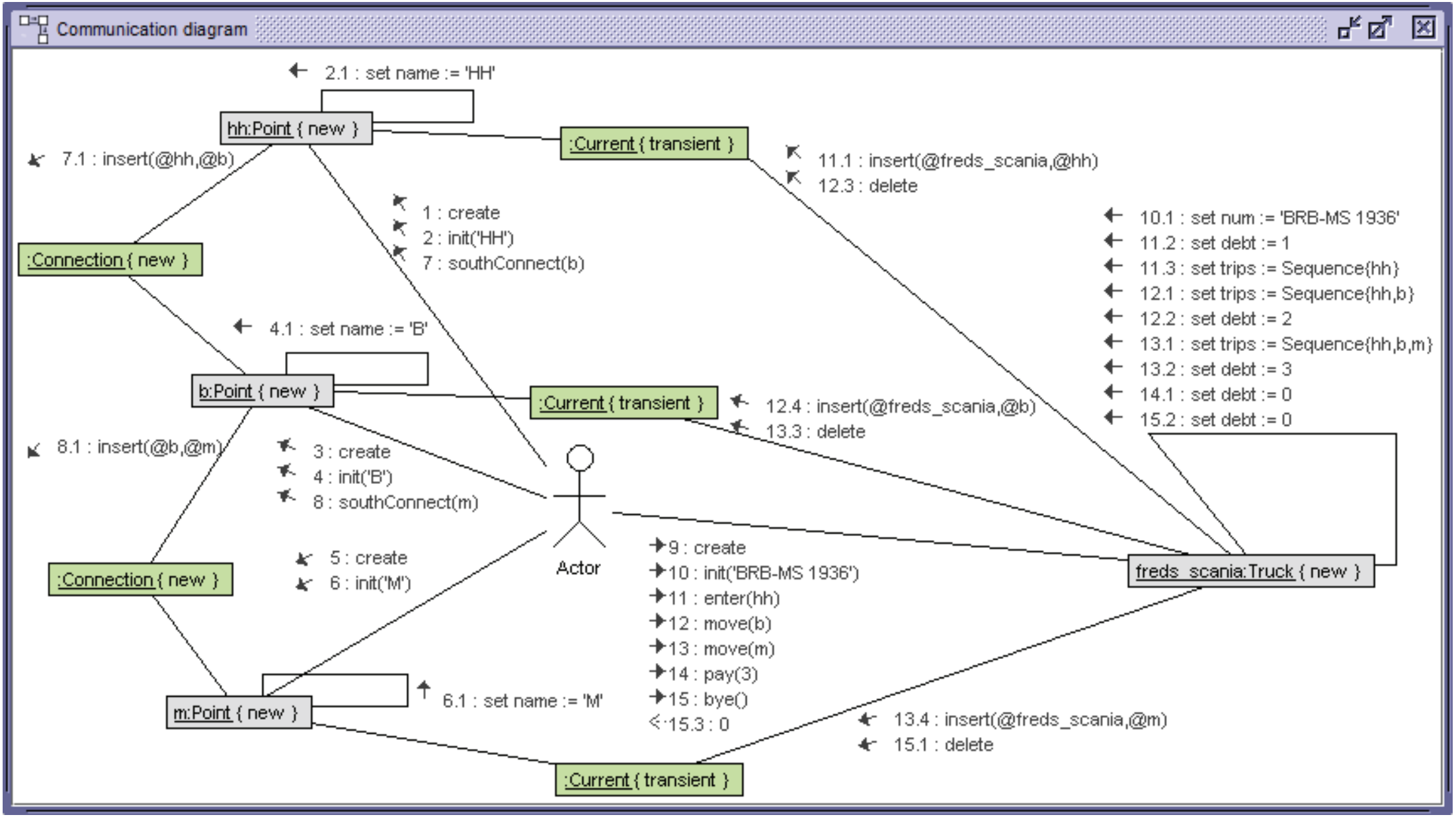

```
context Point inv noCyclesInv:
  noCycles()

Point::noCycles():Boolean =
  Point.allInstances->forAll(p | p.northPlus()->excludes(p))

Point::northPlus():Set(Point) =
  self.north->closure(p|p.north)
```

```
Truck::move(target:Point)

pre currentExists:
  self.current->notEmpty
pre targetReachable:
  self.current.north->union(self.current.south)->includes(target)

post debtIncreased:
  self.debt@pre+1=self.debt
post tripsUpdated:
  self.trips@pre->including(target)=self.trips
post currentAssigned:
  target=self.current
post allTruckInvs:
  numIsKey()

begin                                                    -- SOIL
self.trips:=self.trips->including(target);
self.debt:=self.debt+1;
delete (self,self.current) from Current;
insert (self,target) into Current;
end
```


State machine `PointLife'
Point::PointLife {protocol}
prenatal
create/
born
[(self.name = '')]
init(aName : String)/
growing
[(self.name <> '')]
northConnect(aNorth : Point)/
southConnect(aSouth : Point)/


State machine `TruckLife'
Truck::TruckLife {protocol}
prenatal
create/
init(aNum : String)/
born
[(self.num = '')]
noDebt
[((self.num <> '') and
self.current->isEmpty)]
bye() : Integer/
debt
[((self.num <> '') and
self.current->notEmpty)]
move(target : Point)/
pay(amount : Integer)/
enter(entry : Point)/


Sequence diagram
hh:Point
b:Point
m:Point
freds_scania:Truck
growing
southConnect(m)
growing
born
init('BRB-MS 1936')
noDebt
enter(hh)
debt
move(b)
debt


Communication diagram
2.1 : set name := 'HH'
hh:Point { new }
:Current { transient }
7.1 : insert(@hh,@b)
11.1 : insert(@freds_scania,@hh)
12.3 : delete
1 : create
2 : init('HH')
7 : southConnect(b)
10.1 : set num := 'BRB-MS 1936'
11.2 : set debt := 1
11.3 : set trips := Sequence{hh}
12.1 : set trips := Sequence{hh,b}
12.2 : set debt := 2
13.1 : set trips := Sequence{hh,b,m}
13.2 : set debt := 3
14.1 : set debt := 0
15.2 : set debt := 0
:Connection { new }
4.1 : set name := 'B'
b:Point { new }
:Current { transient }
12.4 : insert(@freds_scania,@b)
13.3 : delete
8.1 : insert(@b,@m)
3 : create
4 : init('B')
8 : southConnect(m)
Actor
5 : create
6 : init('M')
9 : create
10 : init('BRB-MS 1936')
11 : enter(hh)
12 : move(b)
13 : move(m)
14 : pay(3)
15 : bye()
15.3 : 0
freds_scania:Truck { new }
:Connection { new }
m:Point { new }
6.1 : set name := 'M'
13.4 : insert(@freds_scania,@m)
15.1 : delete
:Current { transient }

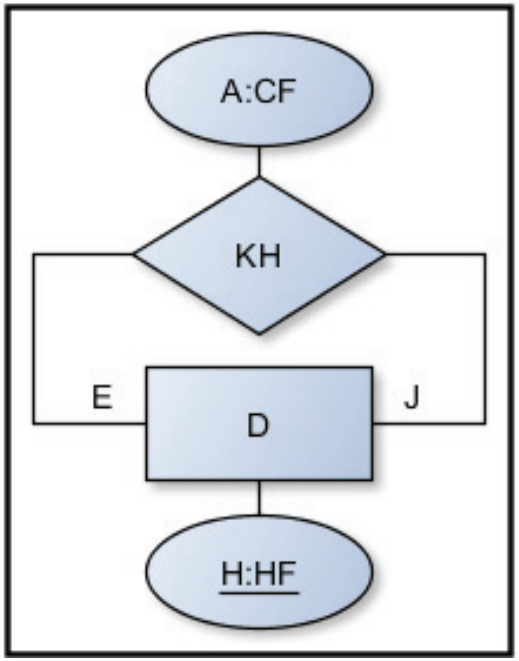
Example model: Toll Collect

- Toll collection system for trucks on German motorways
- Very abstract version (2 classes, 2 associations)
- Class diagram with attributes and operations including invariants and operation pre- and postconditions
- Protocol state machine for each class with state invariants (transition pre- and postconditions not used)
- SOIL implementation for operations
- Validation: Construct example scenario where all operations are invoked and all invariants and operation pre- and postconditons are satisfied
- Scenario is explained and documented with object, sequence and communication diagrams
- Validation: Informal expectations checked against the formal system properties (diagrams, OCL queries, ...)

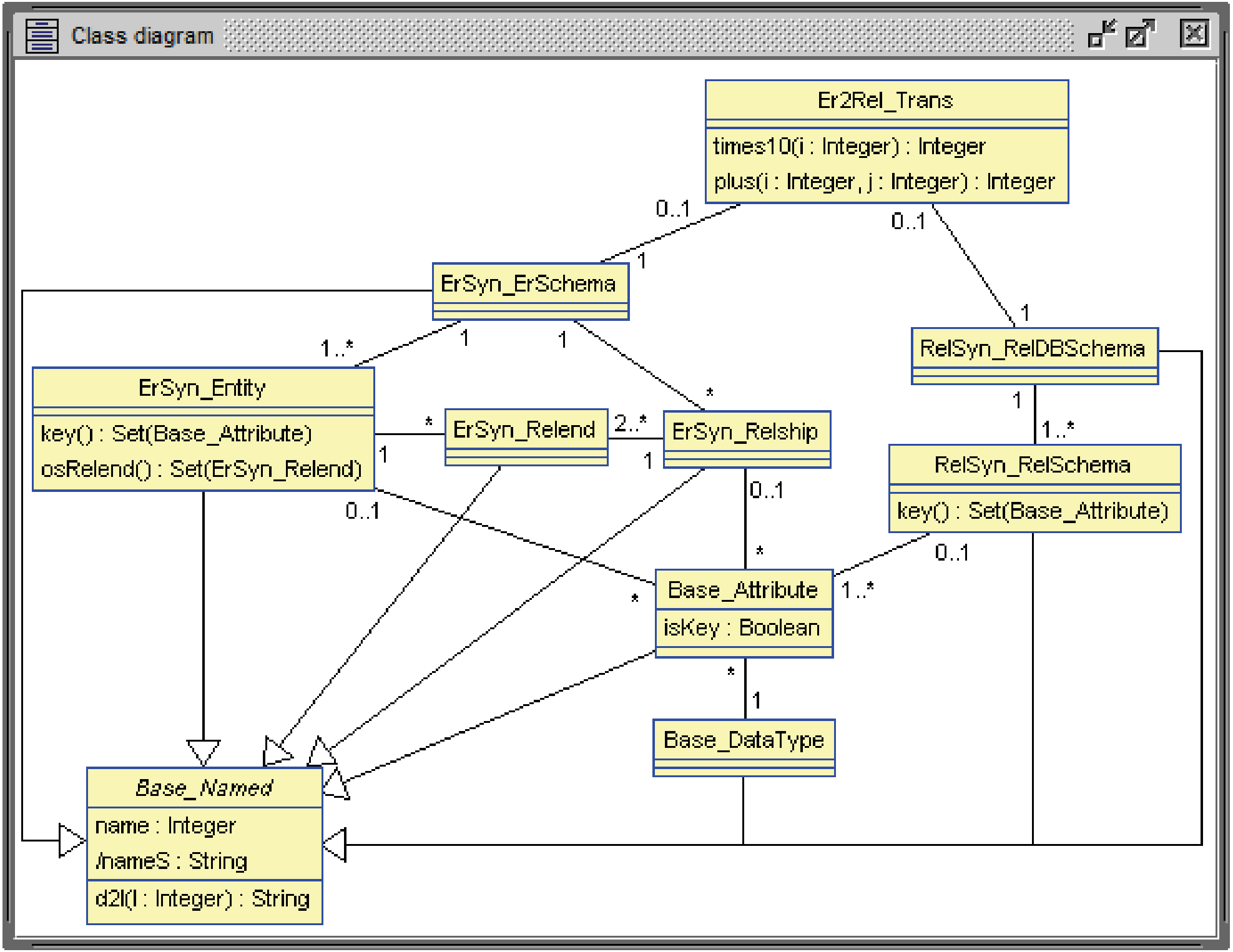
Verification ...

Example model:
Transformation between ER schemas and relational DB schemas

- Metamodel for Entity-Relationship (ER) and for relational datamodel
- Transformation model with single transformation class and invariants
- 23 OCL invariants, partly complex structured
- Verification task: Show transformation model consistency, i.e., automatically construct an example transformation that utilizes all concepts (classes and associations) from the metamodels
- Basis: Model validator that translates UML and OCL models into Kodkod/Alloy and results back in terms of UML and OCL
- Configuration of model validator
  - mandatory upper and optional low. bounds for num. of objects in class
  - optional upper and optional low. bounds for num. of links in assoc.
  - intervals or enumerations for attributes or determination by data types
  - intervals or enumerations for data types




```
create table D(
  H HF primary key)
create table KH(
  EH HF,
  JH HF,
  A CF,
  primary key(EH,JH))
```


Class diagram
Er2Rel_Trans
times10(i : Integer) : Integer
plus(i : Integer, j : Integer) : Integer
ErSyn_ErSchema
RelSyn_RelDBSchema
ErSyn_Entity
key() : Set(Base_Attribute)
osRelend() : Set(ErSyn_Relend)
ErSyn_Relend
ErSyn_Relship
RelSyn_RelSchema
key() : Set(Base_Attribute)
Base_Attribute
isKey : Boolean
Base_DataType
Base_Named
name : Integer
/nameS : String
d2l(l : Integer) : String
0..1
1
1..*
*
2..*

Class invariants

Invariant	Result
Base_Attribute::linkedToOneOfEntityRelshipRelSchema	true
Base_DataType::uniqueDataTypeNames	true
Er2Rel_Trans::forEntityExistsOneRelSchema	true
Er2Rel_Trans::forRelSchemaExistsOneEntityXorRelship	true
Er2Rel_Trans::forRelshipExistsOneRelSchema	true
ErSyn_Entity::differentOsRelendAndAttributeNamesWithinEntity	true
ErSyn_Entity::entityKeyNotEmpty	true
ErSyn_Entity::uniqueAttributeNamesWithinEntity	true
ErSyn_Entity::uniqueOsRelendNamesWithinEntity	true
ErSyn_ErSchema::differentEntityAndRelshipNamesWithinErSchema	true
ErSyn_ErSchema::uniqueEntityNamesWithinErSchema	true
ErSyn_ErSchema::uniqueErSchemaNames	true
ErSyn_ErSchema::uniqueRelshipNamesWithinErSchema	true
ErSyn_Relend::c_Relend_Entity_Relship_ErSchema	true
ErSyn_Relend::relendNameNotZero	true
ErSyn_Relship::differentRelendAndAttributeNamesWithinRelship	true
ErSyn_Relship::relshipKeyEmpty	true
ErSyn_Relship::uniqueAttributeNamesWithinRelship	true
ErSyn_Relship::uniqueRelendNamesWithinRelship	true
RelSyn_RelDBSchema::uniqueRelDBSchemaNames	true
RelSyn_RelDBSchema::uniqueRelSchemaNamesWithinRelDBSchema	true
RelSyn_RelSchema::relSchemaKeyNotEmpty	true
RelSyn_RelSchema::uniqueAttributeNamesWithinRelSchema	true

Constraints ok. (31ms) 100%

```
context self:Er2Rel_Trans inv forRelSchemaExistsOneEntityXorRelship:
  self.relDBSchema.relSchema->forAll(rl |
    self.erSchema.entity->one(e |
      rl.name=e.name and
      rl.attribute->forAll(ra |
        e.attribute->one(ea |
          ra.name=ea.name and ea.dataType=ra.dataType and
          ra.isKey=ea.isKey)))
    xor
    self.erSchema.relship->one(rs |
      rl.name=rs.name and
      rl.attribute->forAll(ra |
        rs.relend->one(re |
          re.entity.key()->one(rek |
            ra.name=re.name.concat('_').concat(rek.name) and
            ra.dataType=rek.dataType and ra.isKey))
        xor
        rs.attribute->one(rsa |
          ra.name=rsa.name and ra.dataType=rsa.dataType and
          ra.isKey=false))))
```

Example model:
Transformation between ER schemas and relational DB schemas

- Metamodel for Entity-Relationship (ER) and for relational datamodel
- Transformation model with single transformation class and invariants
- 23 OCL invariants, partly complex structured
- Verification task: Show transformation model consistency, i.e., automatically construct an example transformation that utilizes all concepts (classes and associations) from the metamodels
- Basis: Model validator that translates UML and OCL models into Kodkod/Alloy and results back in terms of UML and OCL
- Configuration of model validator
  - mandatory upper and optional low. bounds for num. of objects in class
  - optional upper and optional low. bounds for num. of links in assoc.
  - intervals or enumerations for attributes or determination by data types
  - intervals or enumerations for data types

```
Er2Rel_Trans : 1..1

Er2Rel_OwnershipTransErSchema    : 1..1
Er2Rel_OwnershipTransRelDBSchema : 1..1
```

```
ErSyn_ErSchema : 1..1
ErSyn_Entity   : 1..9
ErSyn_Relship  : 1..9
ErSyn_Relend   : 1..9

ErSyn_OwnershipErSchemaEntity   : 1..9
ErSyn_OwnershipErSchemaRelship  : 1..9
ErSyn_OwnershipEntityAttribute  : 1..9
ErSyn_OwnershipRelshipAttribute : 1..9
ErSyn_OwnershipRelshipRelend    : 1..9
ErSyn_RelendTyping              : 1..9
```

```
RelSyn_RelDBSchema : 1..1
RelSyn_RelSchema   : 1..9

RelSyn_OwnershipRelDBSchemaRelSchema : 1..9
RelSyn_OwnershipRelSchemaAttribute   : 1..9
```

```
Base_Attribute       : 1..9
Base_DataType        : 1..9
Base_Named_name      : Set{0,1,2,3,4,5,6,7,8,9,10,11,..., 97,98,99}
Base_Attribute_isKey : Set{false,true}

Base_AttributeTyping : 1..9

Real      : 0..0
Real_step : 1
String    : 0..0
Integer   : 0..127
```

```
Class       black-on-white
Association black-on-light-grey
```

Object diagram

er2rel_trans1:Er2Rel_Trans

ersyn_erschema1:ErSyn_ErSchema
name=90
/nameS='KA'

relsyn_reldbschema1:RelSyn_RelDBSchema
name=98
/nameS='KJ'

relsyn_relschema1:RelSyn_RelSchema
name=3
/nameS='D'

ersyn_relend9:ErSyn_Relend
name=4
/nameS='E'

ersyn_entity1:ErSyn_Entity
name=3
/nameS='D'

ersyn_relship1:ErSyn_Relship
name=97
/nameS='KH'

relsyn_relschema8:RelSyn_RelSchema
name=97
/nameS='KH'

ersyn_relend1:ErSyn_Relend
name=8
/nameS='J'

base_attribute6:Base_Attribute
name=0
/nameS='A'
isKey=false

base_attribute2:Base_Attribute
name=0
/nameS='A'
isKey=false

base_attribute4:Base_Attribute
name=47
/nameS='EH'
isKey=true

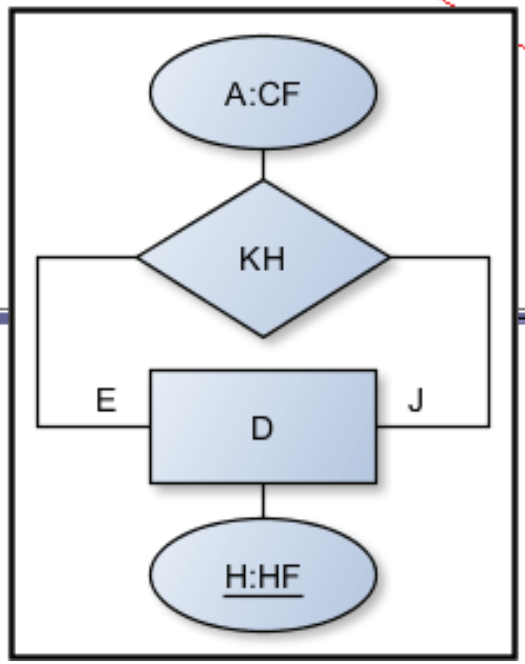
base_attribute3:Base_Attribute
name=87
/nameS='JH'
isKey=true

base_attribute1:Base_Attribute
name=7
/nameS='H'
isKey=true

base_attribute5:Base_Attribute
name=7
/nameS='H'
isKey=true

base_datatype6:Base_DataType
name=75
/nameS='HF'

base_datatype5:Base_DataType
name=25
/nameS='CF'



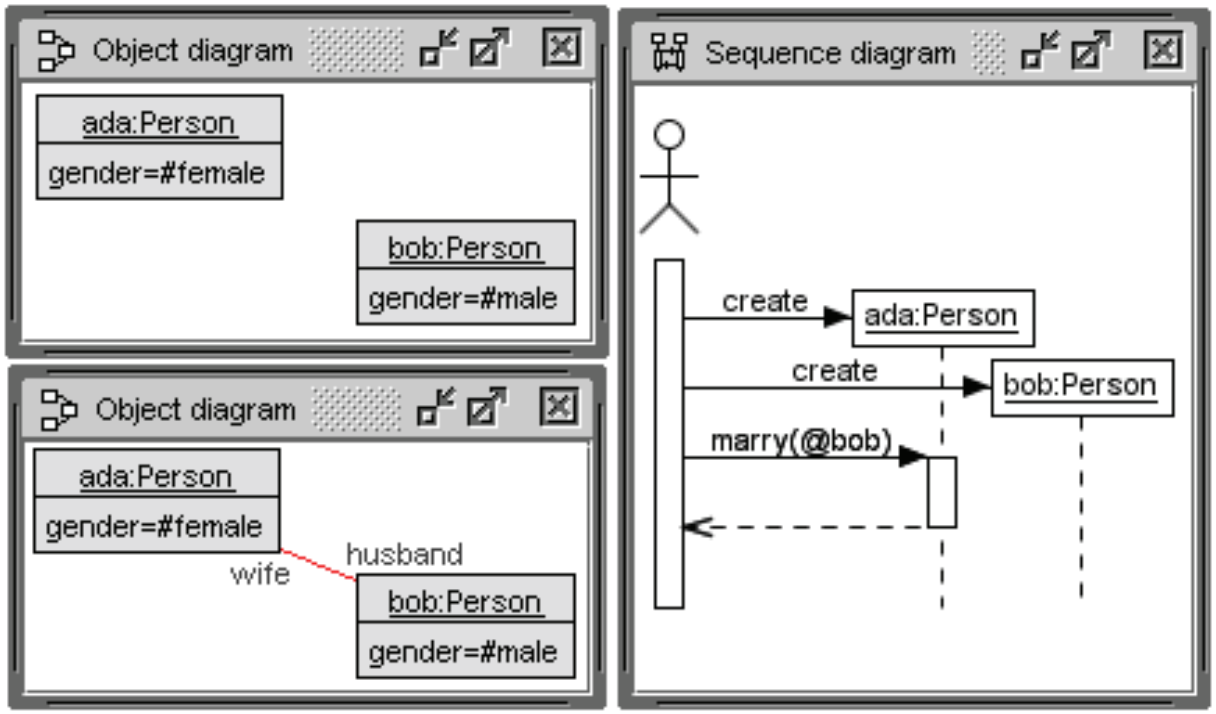

```
create table D(
  H HF primary key)
create table KH(
  EH HF,
  JH HF,
  A CF,
  primary key(EH,JH))
```

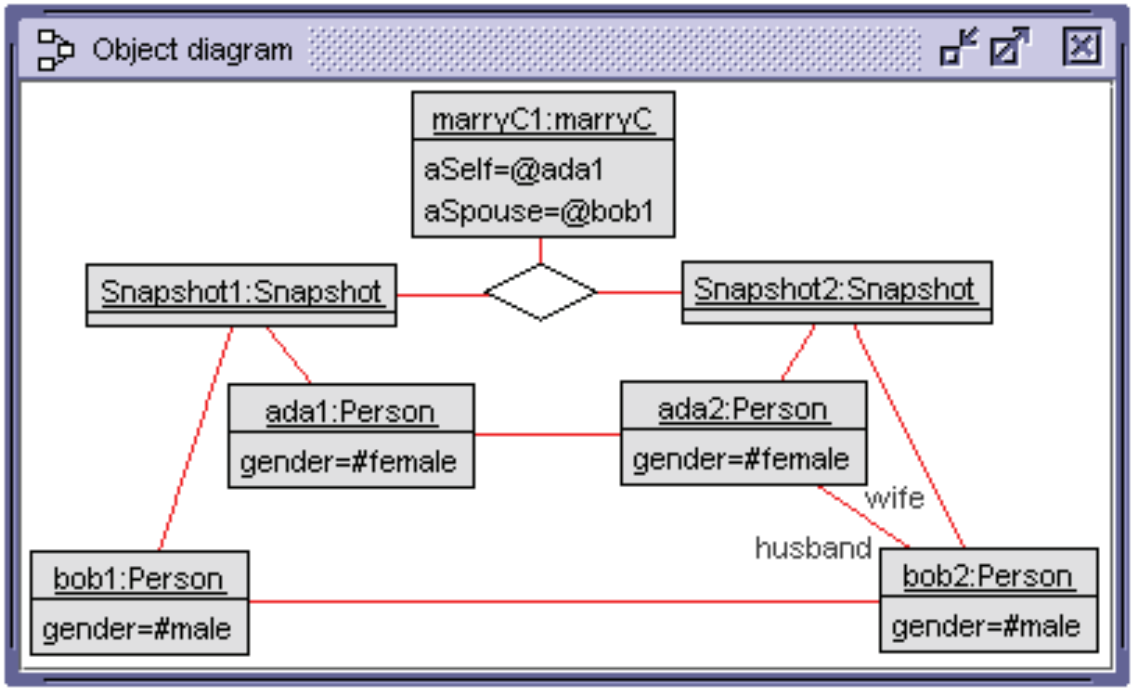
Further options for verification of transformations

- Other forms of transformation model consistency
  - here: class and association instanciability
  - possible: class instanciability
  - possible: transformation class instanciability
- Completion of partial source model by model validator in order to compute the transformation result
- Checks for result uniqueness
- Indepence of invariants (no dependeny among invariants, kind of minimality property)
- Add negated consequences and show insatisfiability (in finite search space) or detect counterexample

Thanks for your attention!

Filmstripping: Transforming pre- and postconditions into invariants





USE Monitor

- Plug-In to support runtime verification
- Verify assumptions of a monitored system
  - single state (multiplicities, invariants,…)
  - execution (pre-/postconditions, transitions, …)
- Extensible by adapters (Java, .NET, RDBMS, …)
- USE model enriched with platform specific information
- Visualization of traces (seq./comm. diagrams)