Behavior Modeling with Interaction Diagrams in a UML and OCL Tool

Martin Gogolla, Lars Hamann, Frank Hilken,
Matthias Sedlmeier, Quang Dung Nguyen

University of Bremen, Germany

Motivation and context

- system modeling with UML behavior diagrams

- statecharts and both kinds of interaction diagrams:
  sequence and communication diagrams

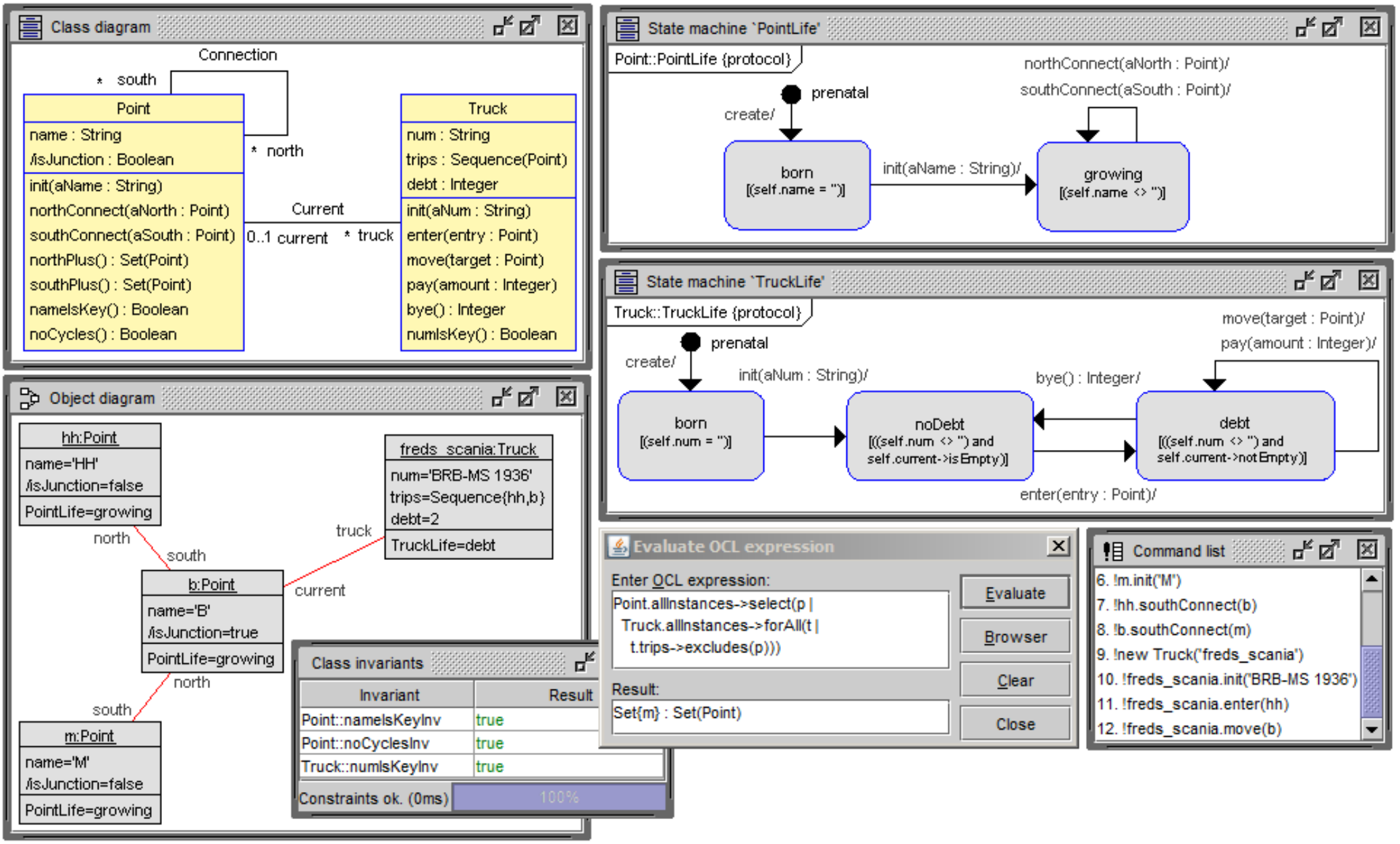
- new implementation features in a UML and OCL modeling tool:
  USE (Uml-based Specification Environment)

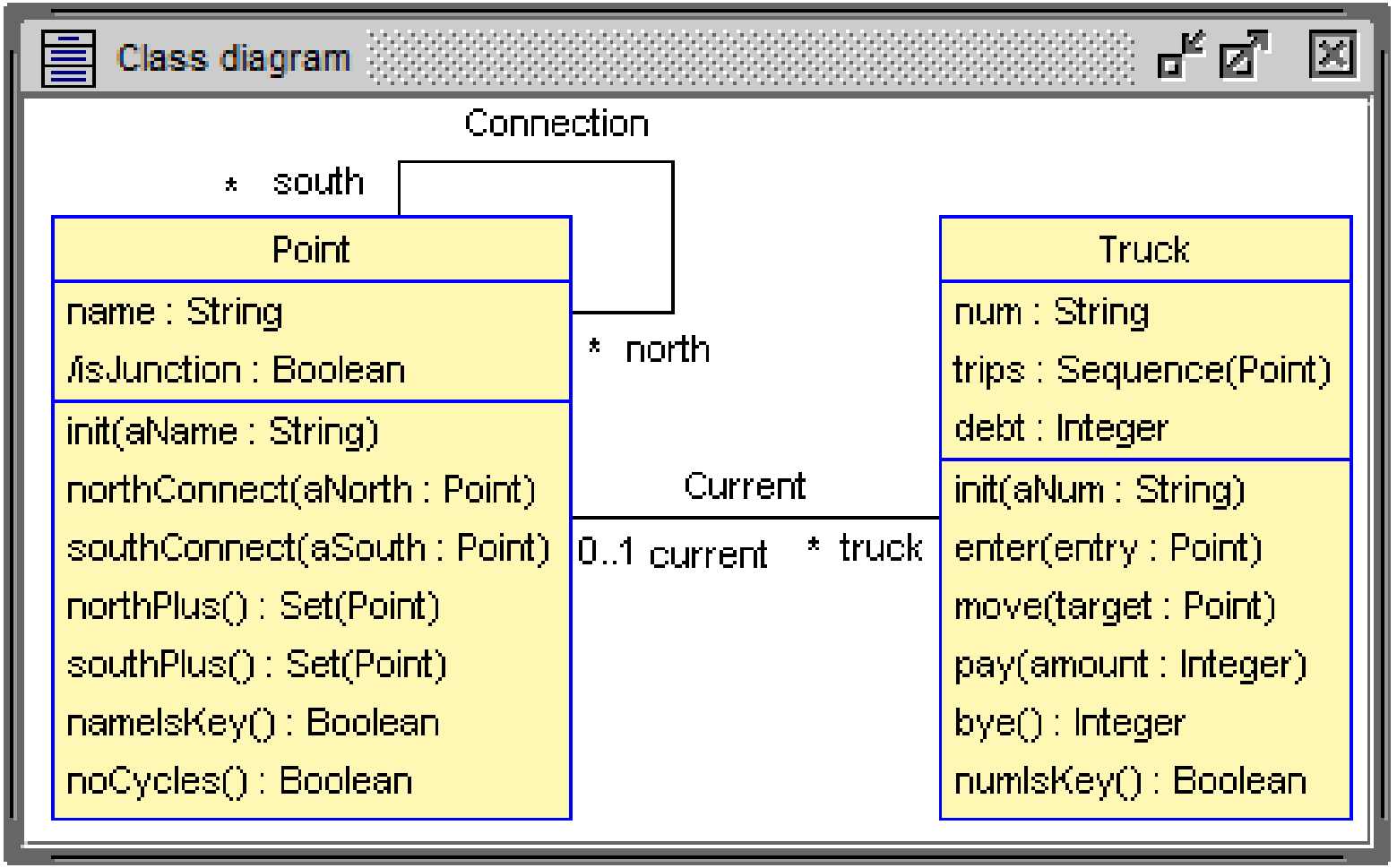
- sequence diagram lifelines are extended with
  states from statecharts

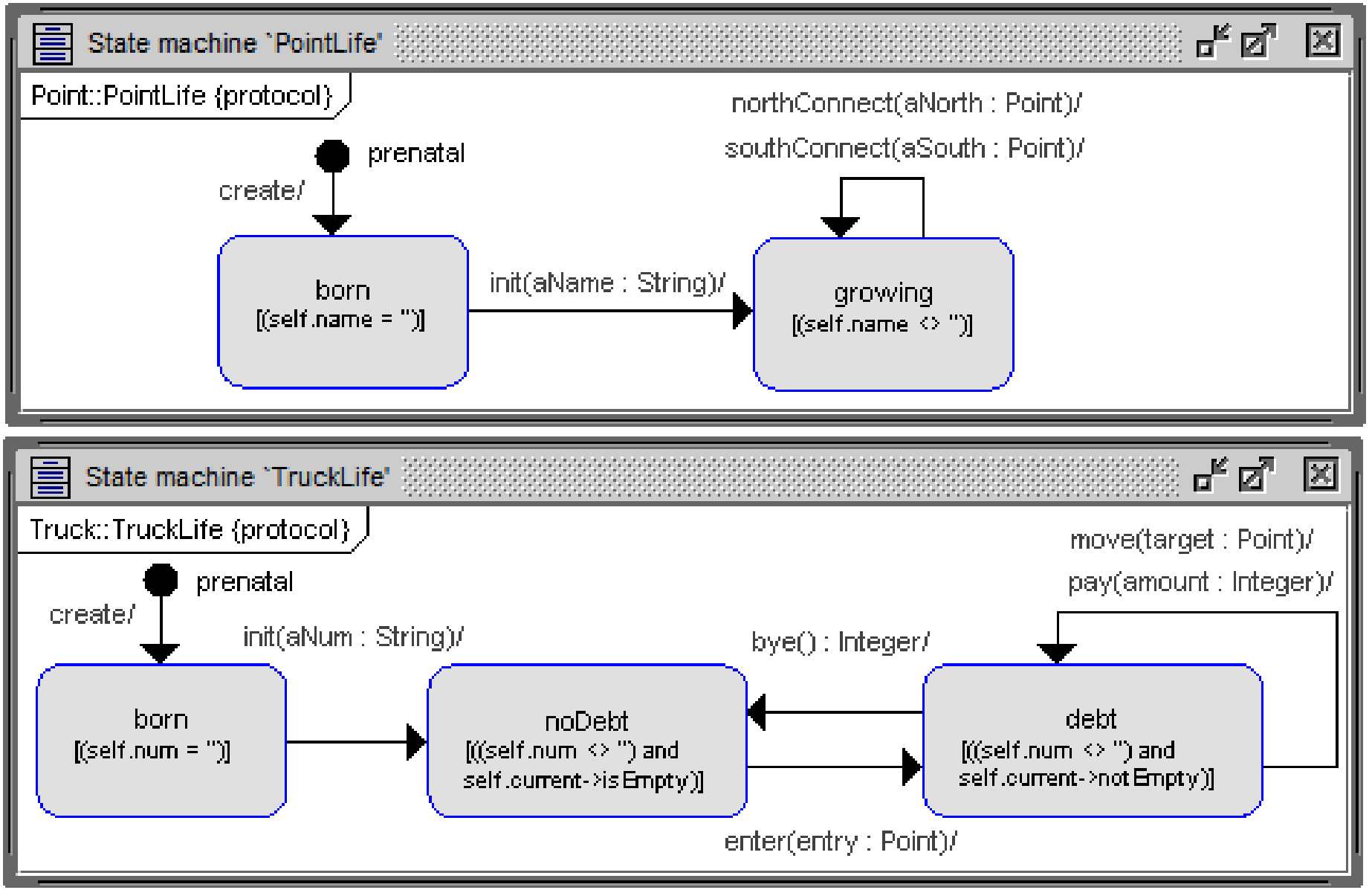
- communication diagrams introduced as an alternative to
  sequence diagrams

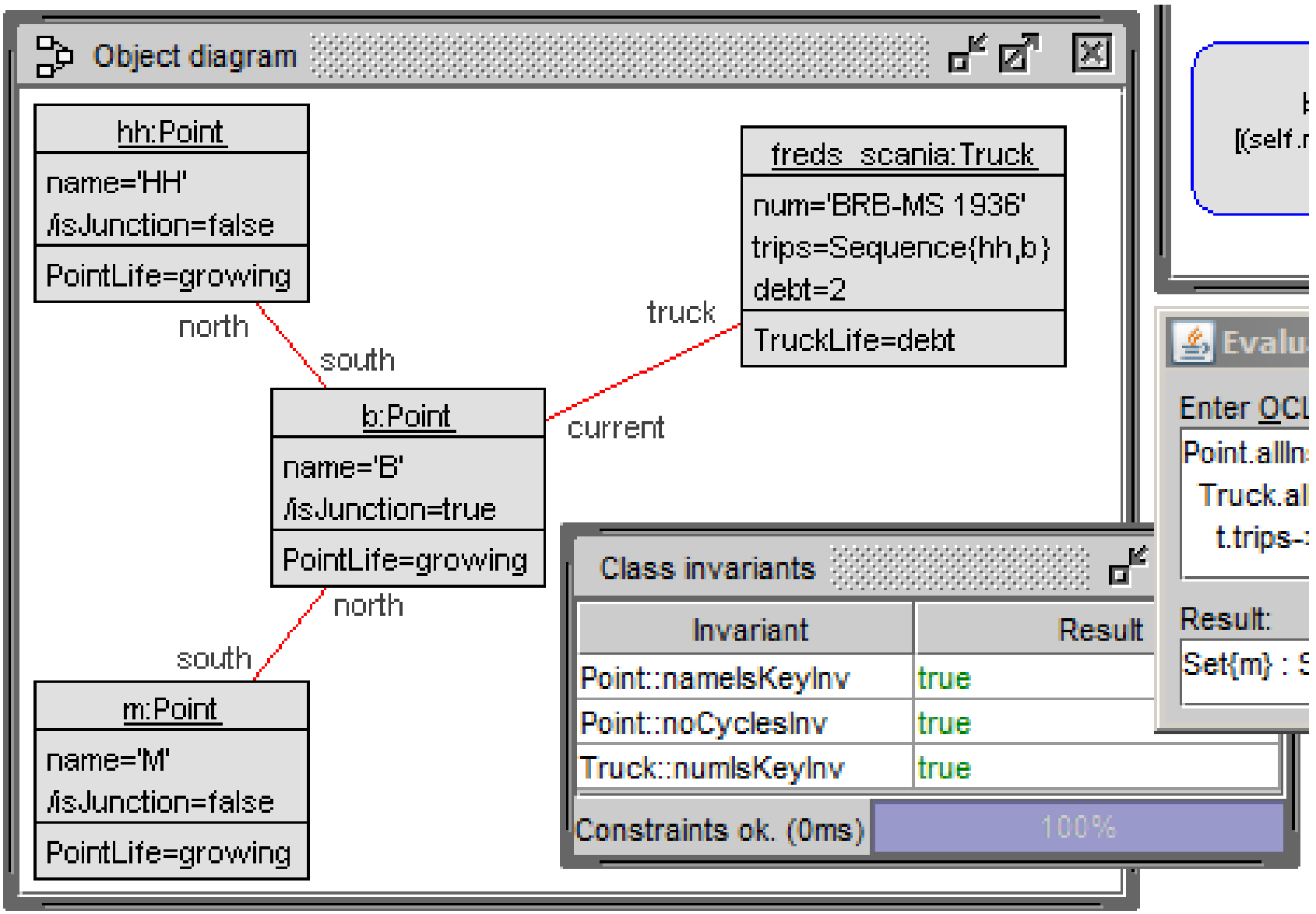
- assess introduced features and propose systematic set of features
  that should be available in both interaction diagrams

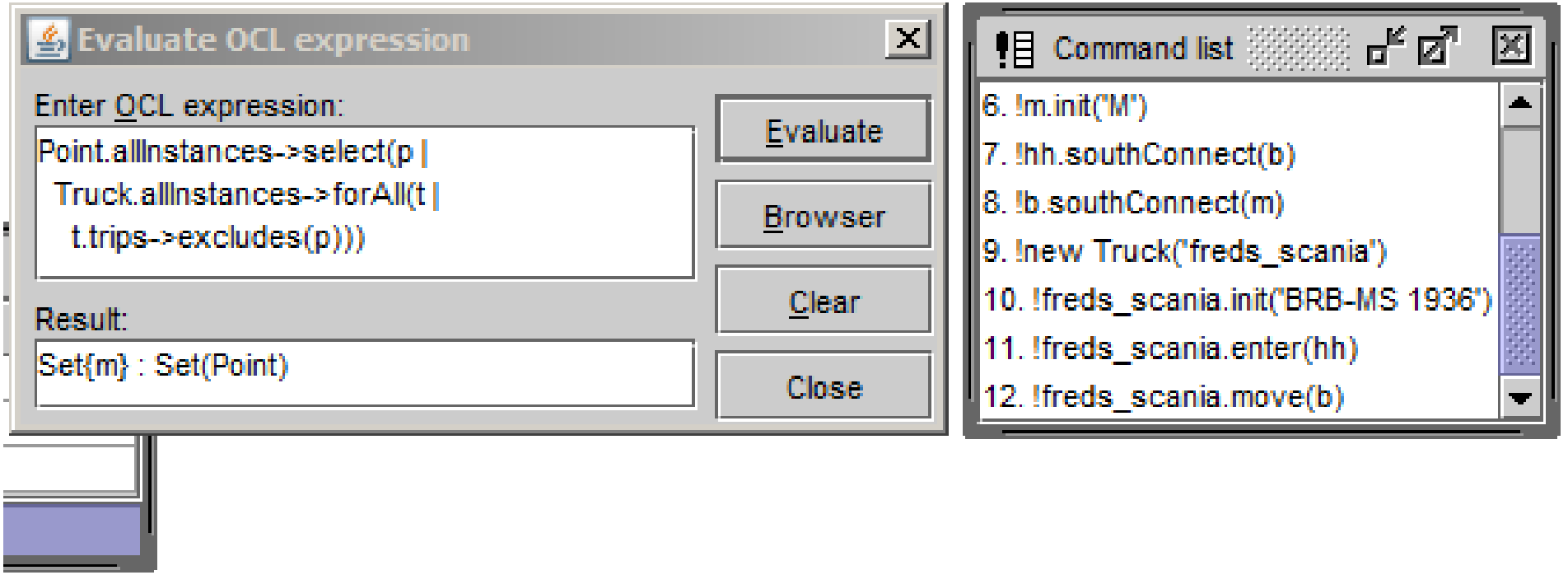
- emphasize the role that OCL plays for such a feature set

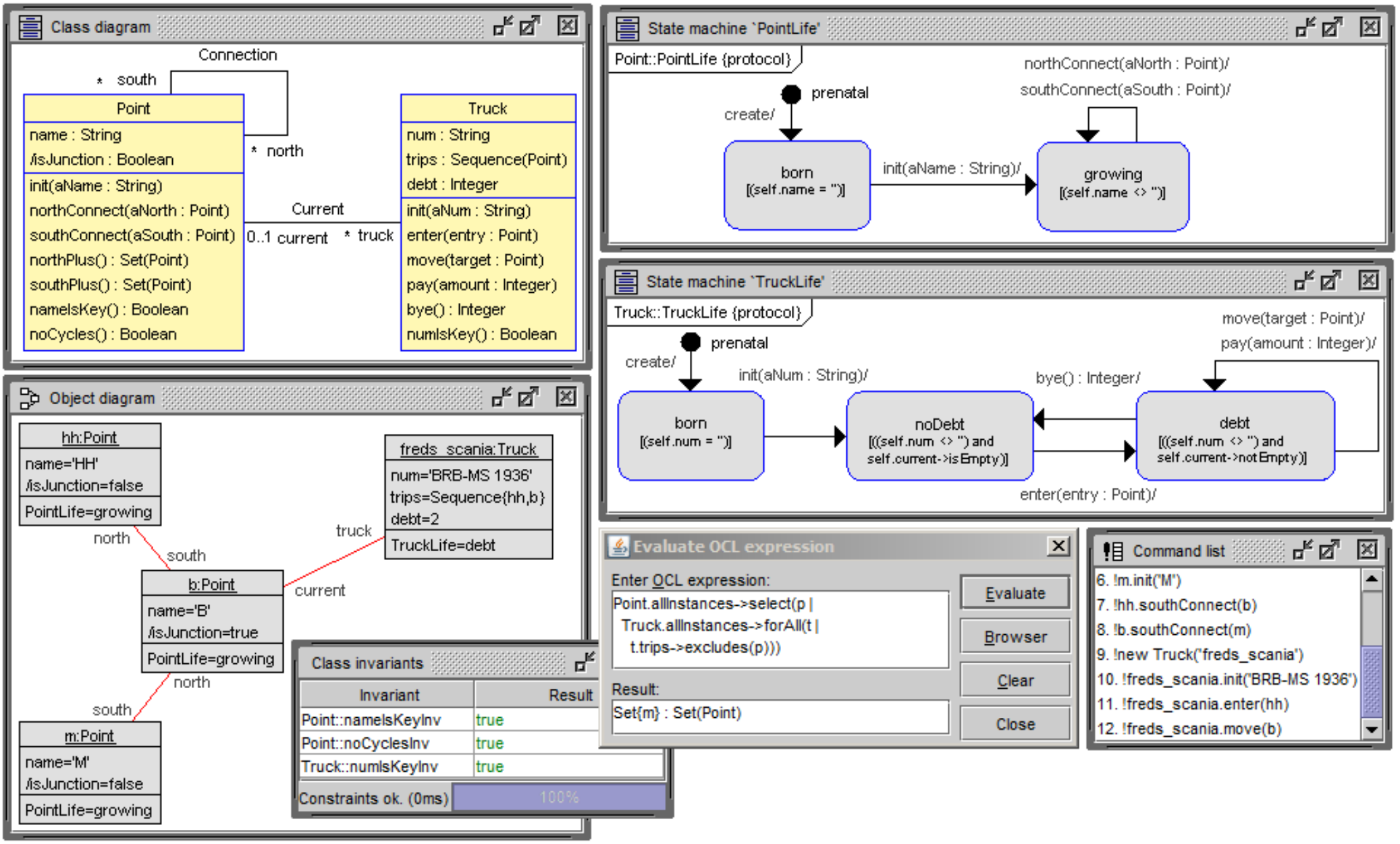

Class diagram
Connection
* south
* north
Point
name : String
/isJunction : Boolean
init(aName : String)
northConnect(aNorth : Point)
southConnect(aSouth : Point)
northPlus() : Set(Point)
southPlus() : Set(Point)
nameIsKey() : Boolean
noCycles() : Boolean
Current
0..1 current
* truck
Truck
num : String
trips : Sequence(Point)
debt : Integer
init(aNum : String)
enter(entry : Point)
move(target : Point)
pay(amount : Integer)
bye() : Integer
numIsKey() : Boolean
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/
born
[(self.num = '')]
init(aNum : String)/
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)/
Object diagram
hh:Point
name='HH'
/isJunction=false
PointLife=growing
north
south
b:Point
name='B'
/isJunction=true
PointLife=growing
north
south
m:Point
name='M'
/isJunction=false
PointLife=growing
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b}
debt=2
TruckLife=debt
truck
current
Class invariants
Invariant
Result
Point::nameIsKeyInv
true
Point::noCyclesInv
true
Truck::numIsKeyInv
true
Constraints ok. (0ms)
100%
Evaluate OCL expression
Enter OCL expression:
Point.allInstances->select(p |
Truck.allInstances->forAll(t |
t.trips->excludes(p)))
Result:
Set{m} : Set(Point)
Evaluate
Browser
Clear
Close
Command list
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)


Class diagram
Connection
* south
Point
name : String
/isJunction : Boolean
init(aName : String)
northConnect(aNorth : Point)
southConnect(aSouth : Point)
northPlus() : Set(Point)
southPlus() : Set(Point)
nameIsKey() : Boolean
noCycles() : Boolean
* north
Current
0..1 current
* truck
Truck
num : String
trips : Sequence(Point)
debt : Integer
init(aNum : String)
enter(entry : Point)
move(target : Point)
pay(amount : Integer)
bye() : Integer
numIsKey() : Boolean


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)/


Object diagram
hh:Point
name='HH'
/isJunction=false
PointLife=growing
north
south
b:Point
name='B'
/isJunction=true
PointLife=growing
north
south
m:Point
name='M'
/isJunction=false
PointLife=growing
truck
current
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b}
debt=2
TruckLife=debt
Class invariants
Invariant
Result
Point::nameIsKeyInv
true
Point::noCyclesInv
true
Truck::numIsKeyInv
true
Constraints ok. (0ms)
100%
Enter OCL
Result:
Set{m} : S


Evaluate OCL expression
Enter OCL expression:
Point.allInstances->select(p |
Truck.allInstances->forAll(t |
t.trips->excludes(p)))
Result:
Set{m} : Set(Point)
Evaluate
Browser
Clear
Close
Command list
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)


Class diagram
Connection
* south
* north
Point
name : String
/isJunction : Boolean
init(aName : String)
northConnect(aNorth : Point)
southConnect(aSouth : Point)
northPlus() : Set(Point)
southPlus() : Set(Point)
nameIsKey() : Boolean
noCycles() : Boolean
Current
0..1 current
* truck
Truck
num : String
trips : Sequence(Point)
debt : Integer
init(aNum : String)
enter(entry : Point)
move(target : Point)
pay(amount : Integer)
bye() : Integer
numIsKey() : Boolean
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/
born
[(self.num = '')]
init(aNum : String)/
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)/
Object diagram
hh:Point
name='HH'
/isJunction=false
PointLife=growing
north
south
b:Point
name='B'
/isJunction=true
PointLife=growing
north
south
m:Point
name='M'
/isJunction=false
PointLife=growing
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh,b}
debt=2
TruckLife=debt
truck
current
Class invariants
Invariant
Result
Point::nameIsKeyInv
true
Point::noCyclesInv
true
Truck::numIsKeyInv
true
Constraints ok. (0ms)
100%
Evaluate OCL expression
Enter OCL expression:
Point.allInstances->select(p |
Truck.allInstances->forAll(t |
t.trips->excludes(p)))
Result:
Set{m} : Set(Point)
Evaluate
Browser
Clear
Close
Command list
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)

Tasks for OCL within USE

- in class diagrams for
  - (a) class invariants
  - (b) operation contracts (pre- and postconditions)
  - (c) attribute and association derivation rules
  - (d) attribute initializations

- in protocol state machines for
  - (e) state invariants
  - (f) transition pre- and postconditions

- furthermore for
  - (g) ad-hoc OCL queries in object diagrams
  - (h) expressions within SOIL

Class diagrams and protocol machines enriched by invariants, operation contracts, statechart constraints and SOIL operation implementations determine system structure and behavior

Sequence and communication diagrams in USE for visualizing and analyzing specified test cases in form of scenarios;
interaction diagrams do not restrict system behavior,
but document, analyze, and help to understand the interactions

Aim of USE: support development by reasoning about model through

(a) **validation**, i.e., checking informal expectations
against formally given properties, for example, by stating
OCL queries against a reached system state

(b) **verification**, i.e., checking formal properties of the model,
for example by considering model consistency or
independence of invariants; USE supports making deductions from
stated model on the basis of finite search space of possible
system states

- USE supports the development of **tests** (scenarios)

- OCL operations **contracts**, i.e., pre- and postconditions, are general
OCL formulas

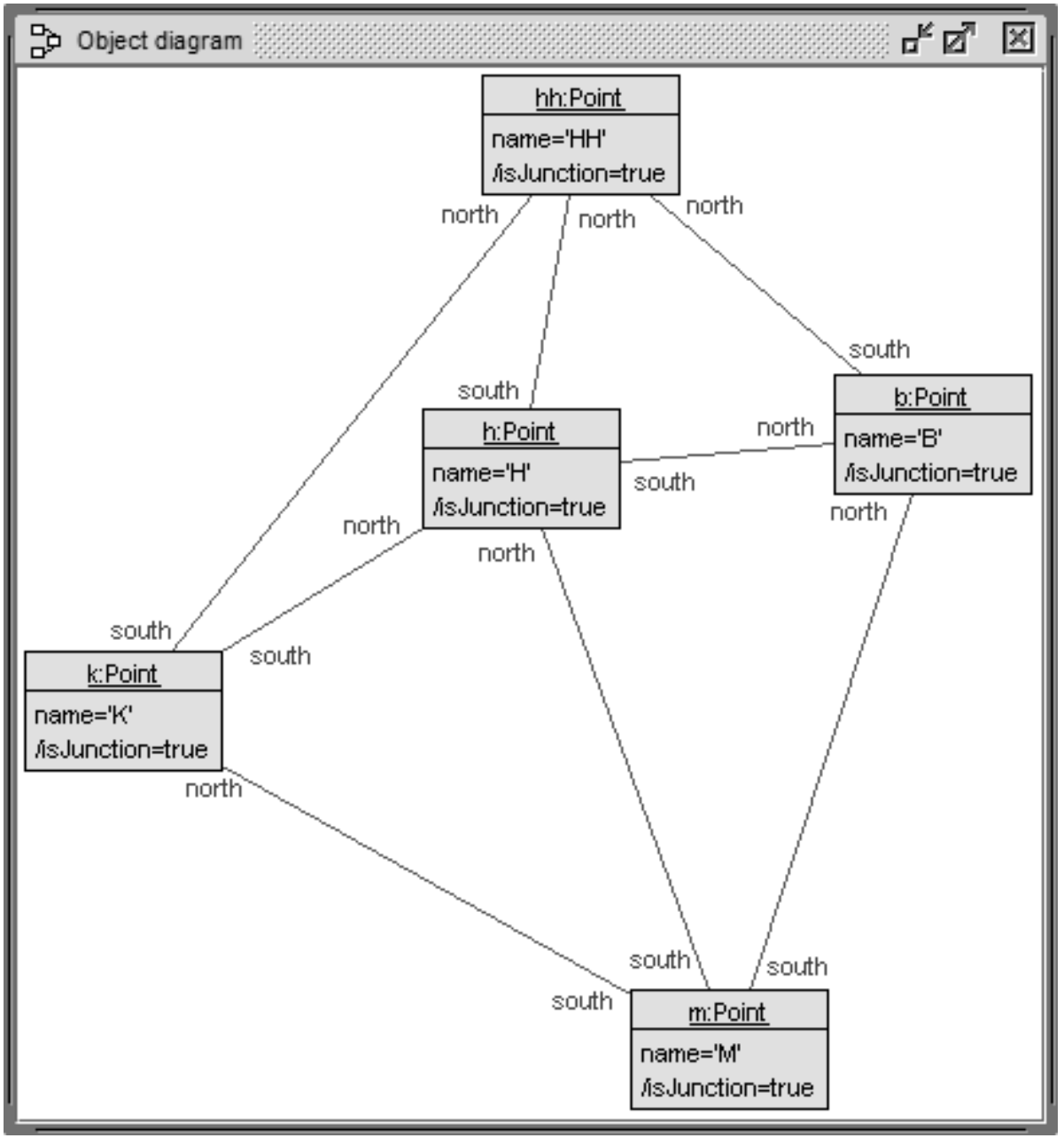
- in **postconditions**, one can refer with @pre to
attribute and association end values at precondition time;
postconditions formulate general requirements;
not restricted to changes to attribute and association end values;
postconditions need not to determine a unique post-state

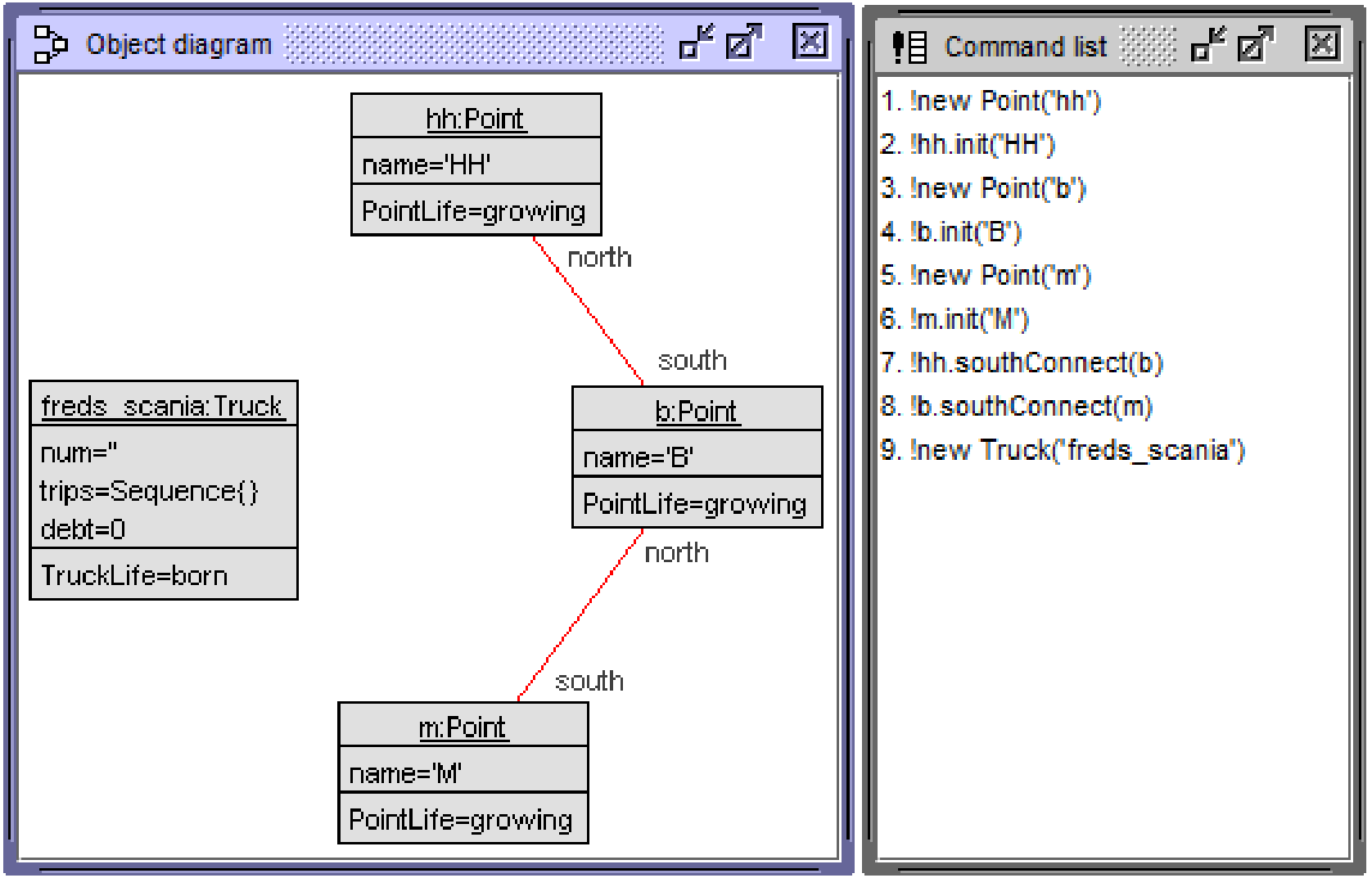
- **actual changes** by the SOIL operations that are checked
against the contract

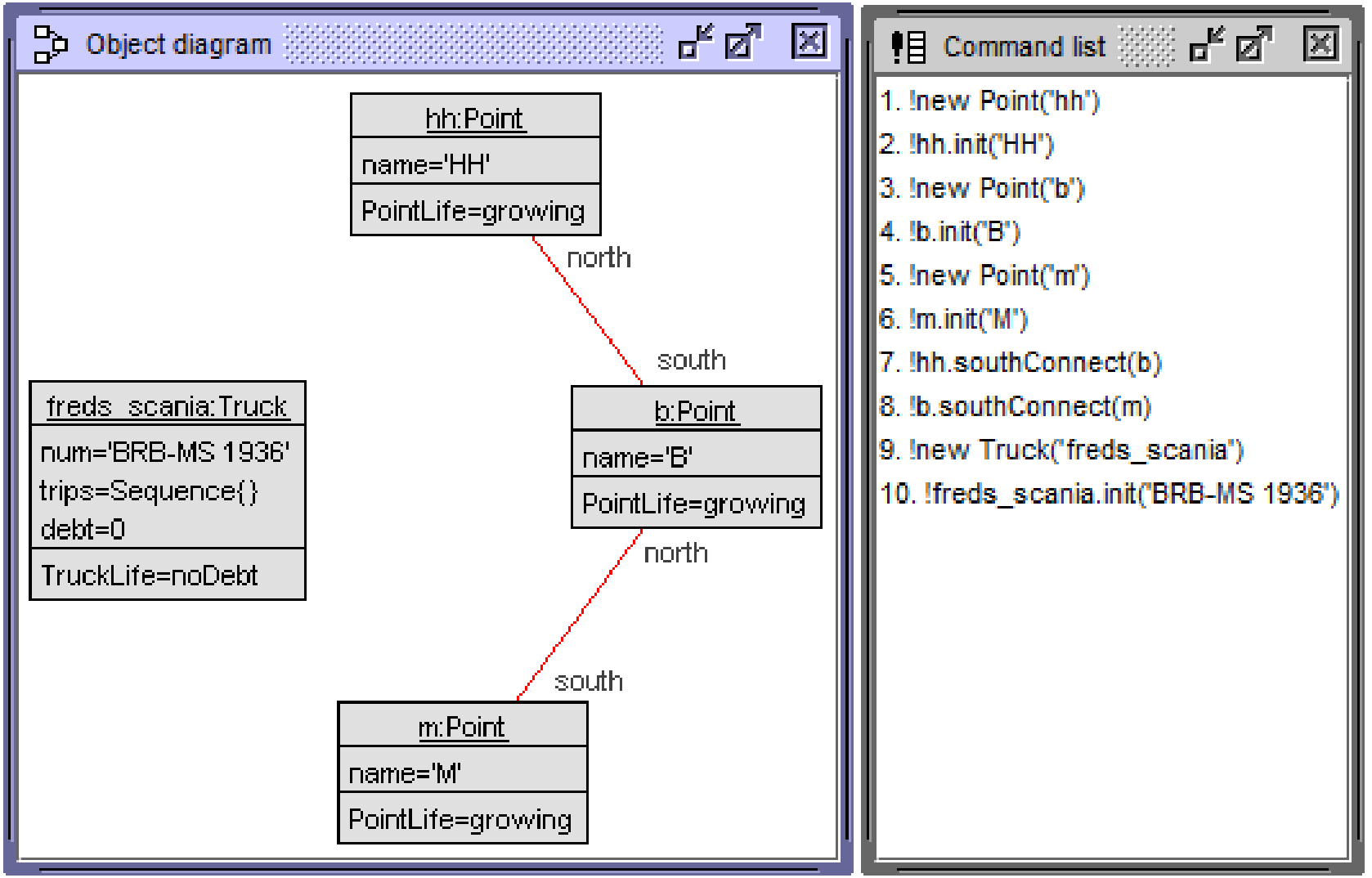
Example of operation implementation and pre- and postconditions

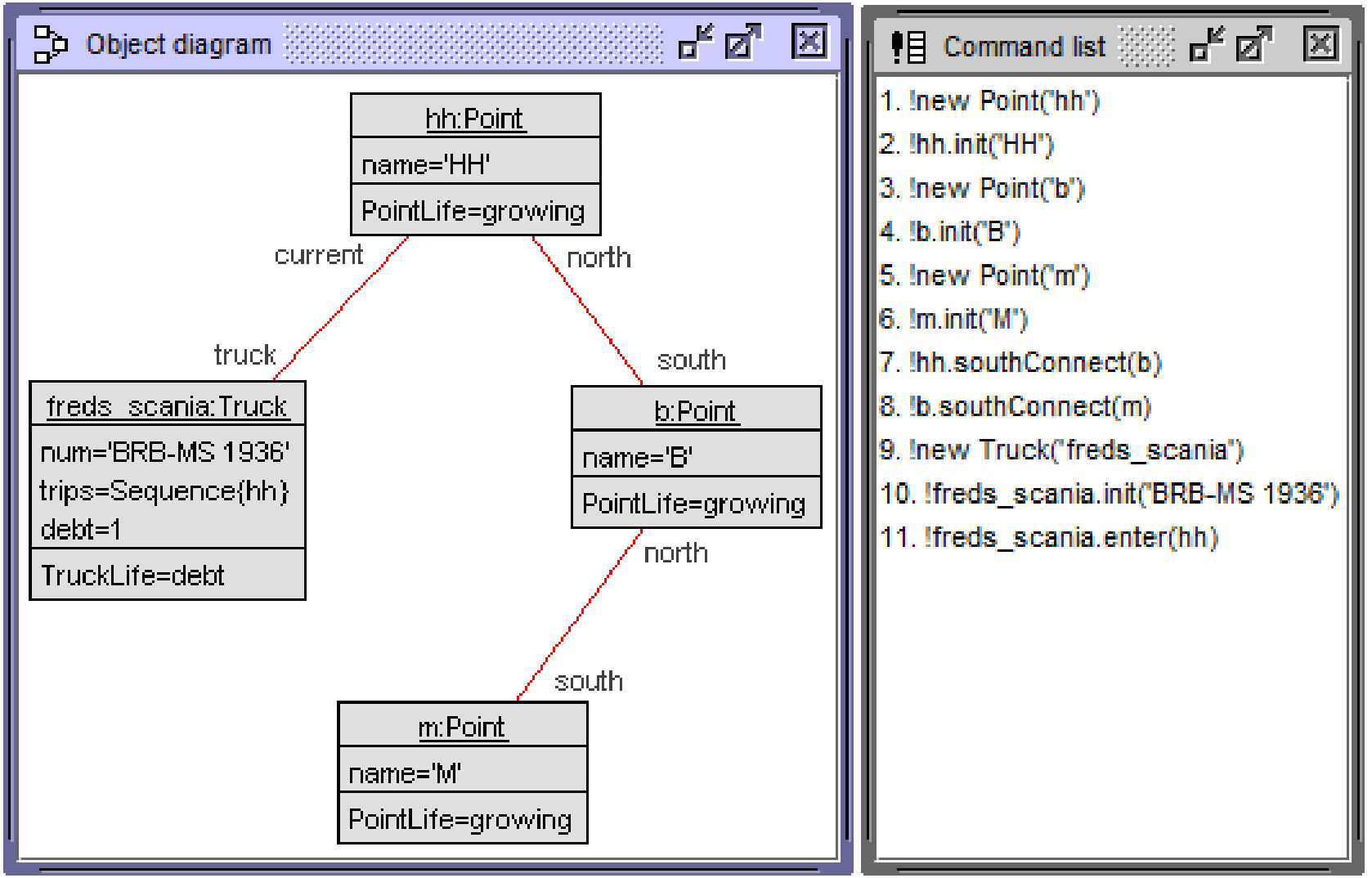
```
Truck::move(target:Point)
  begin
  self.trips:=self.trips->including(target);
  self.debt:=self.debt+1;
  delete (self,self.current) from Current;
  insert (self,target) into Current;
  end

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()
```

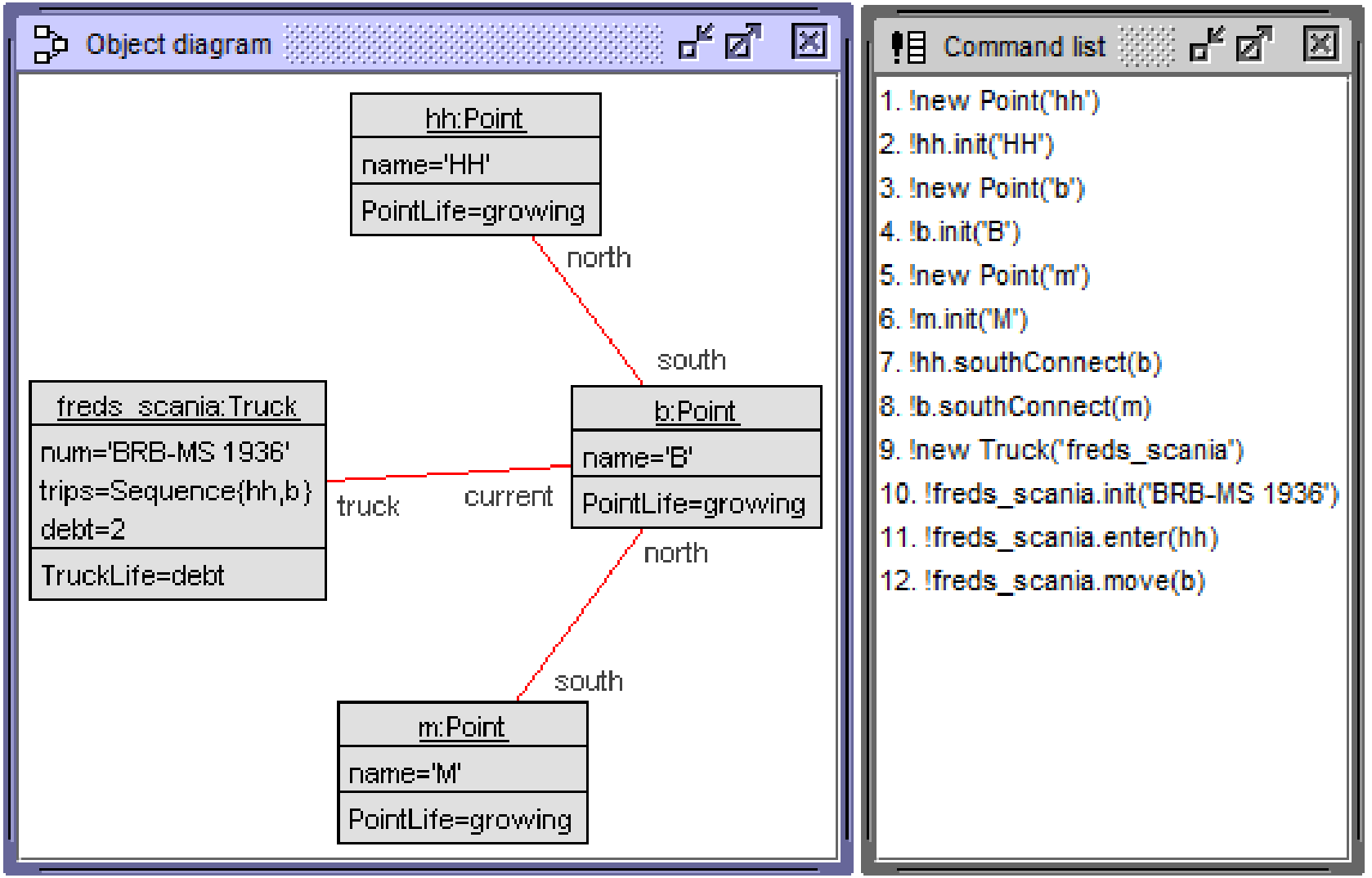
Example for motorway with west/east connections

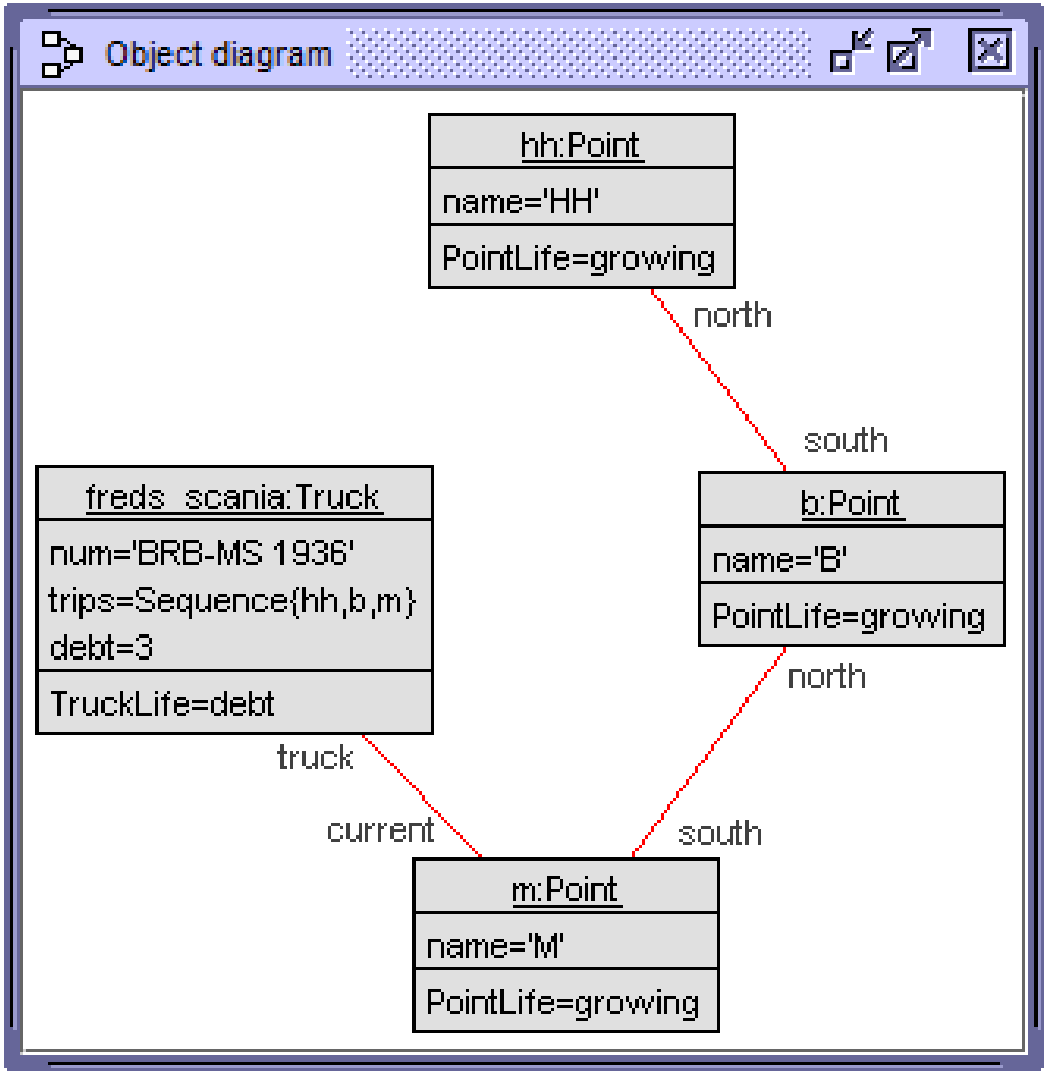



Object diagram
hh:Point
name='HH'
PointLife=growing
north
south
freds_scania:Truck
num=''
trips=Sequence{}
debt=0
TruckLife=born
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing
Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new 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')

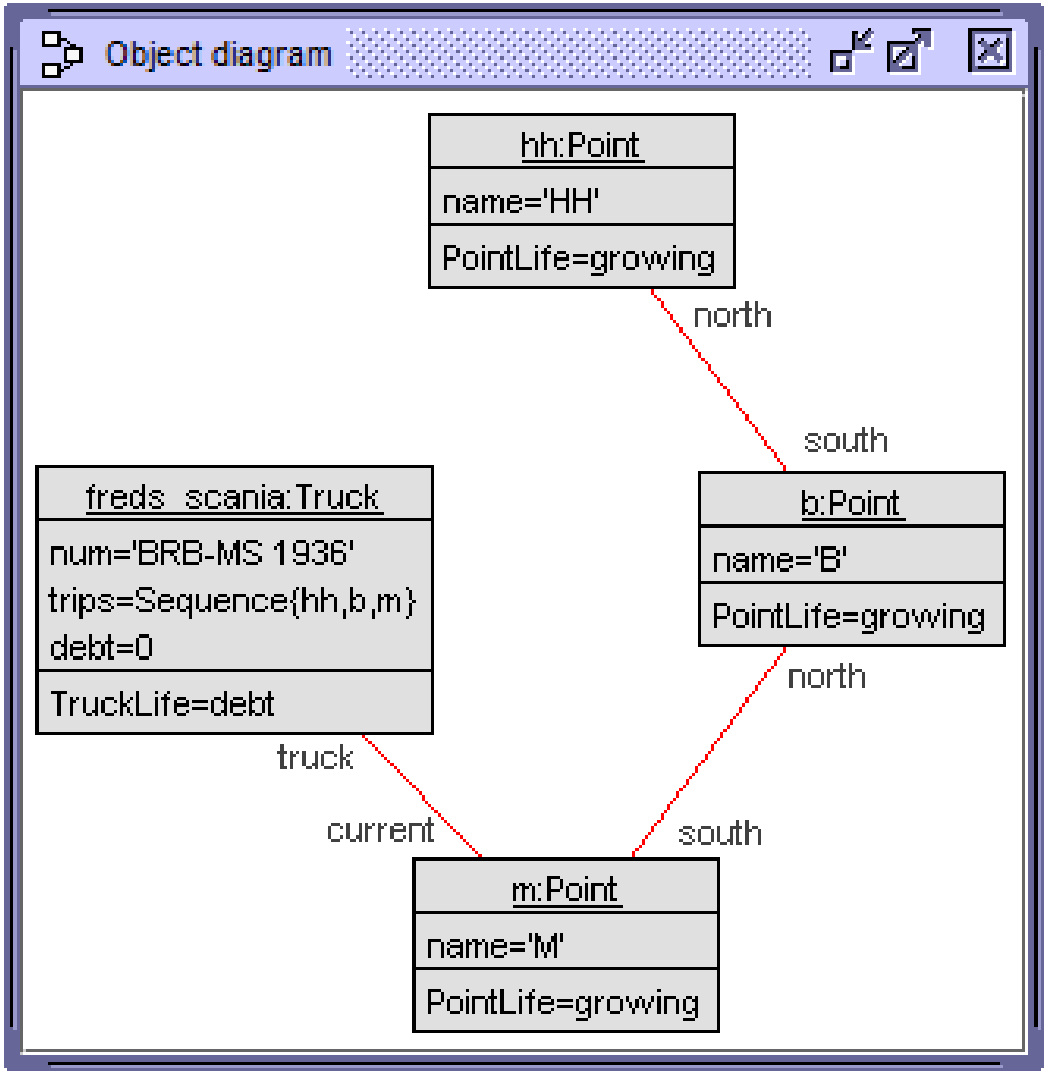

Object diagram
hh:Point
name='HH'
PointLife=growing
current
north
truck
south
freds_scania:Truck
num='BRB-MS 1936'
trips=Sequence{hh}
debt=1
TruckLife=debt
b:Point
name='B'
PointLife=growing
north
south
m:Point
name='M'
PointLife=growing
Command list
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
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)



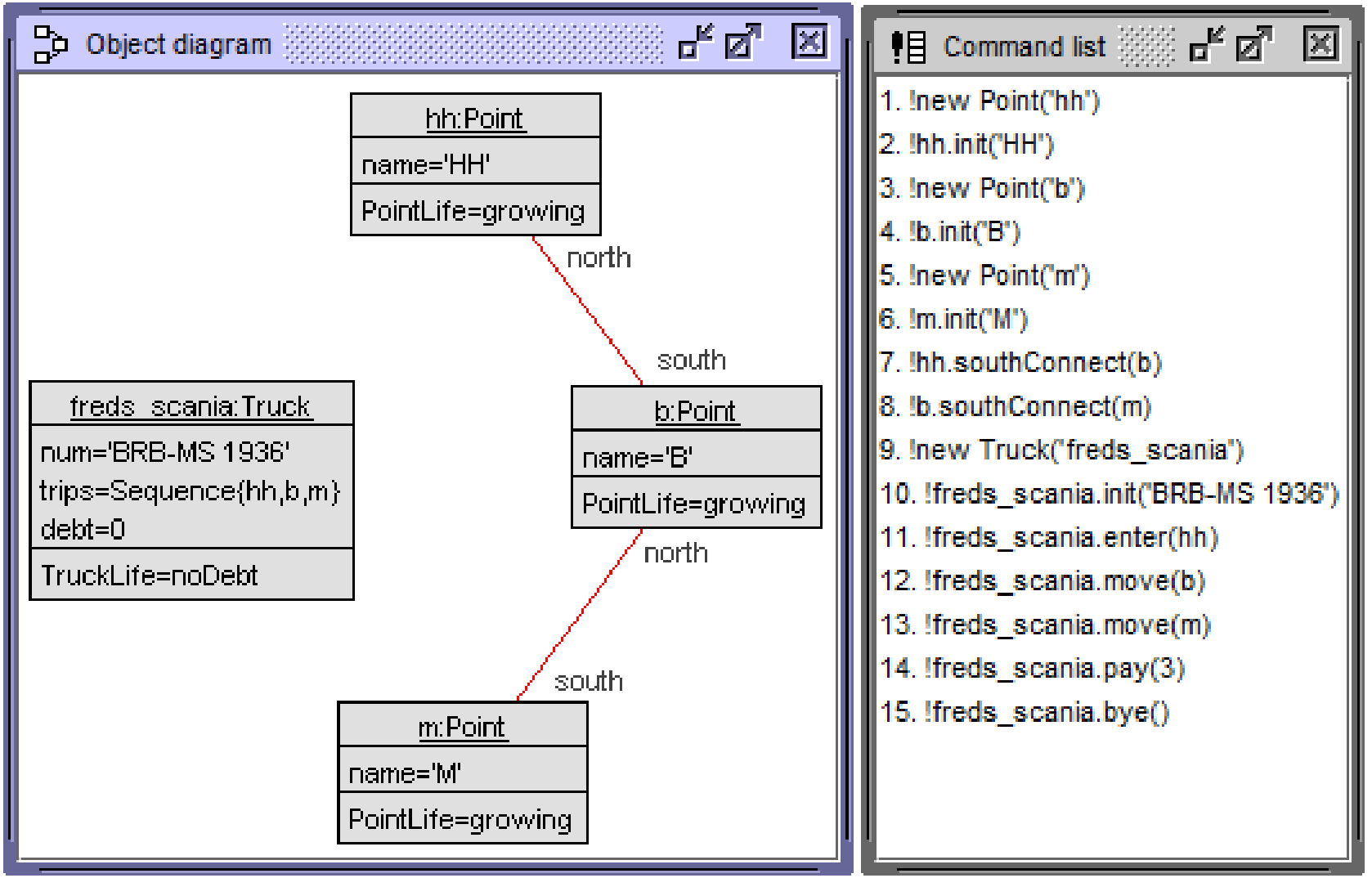
Command list

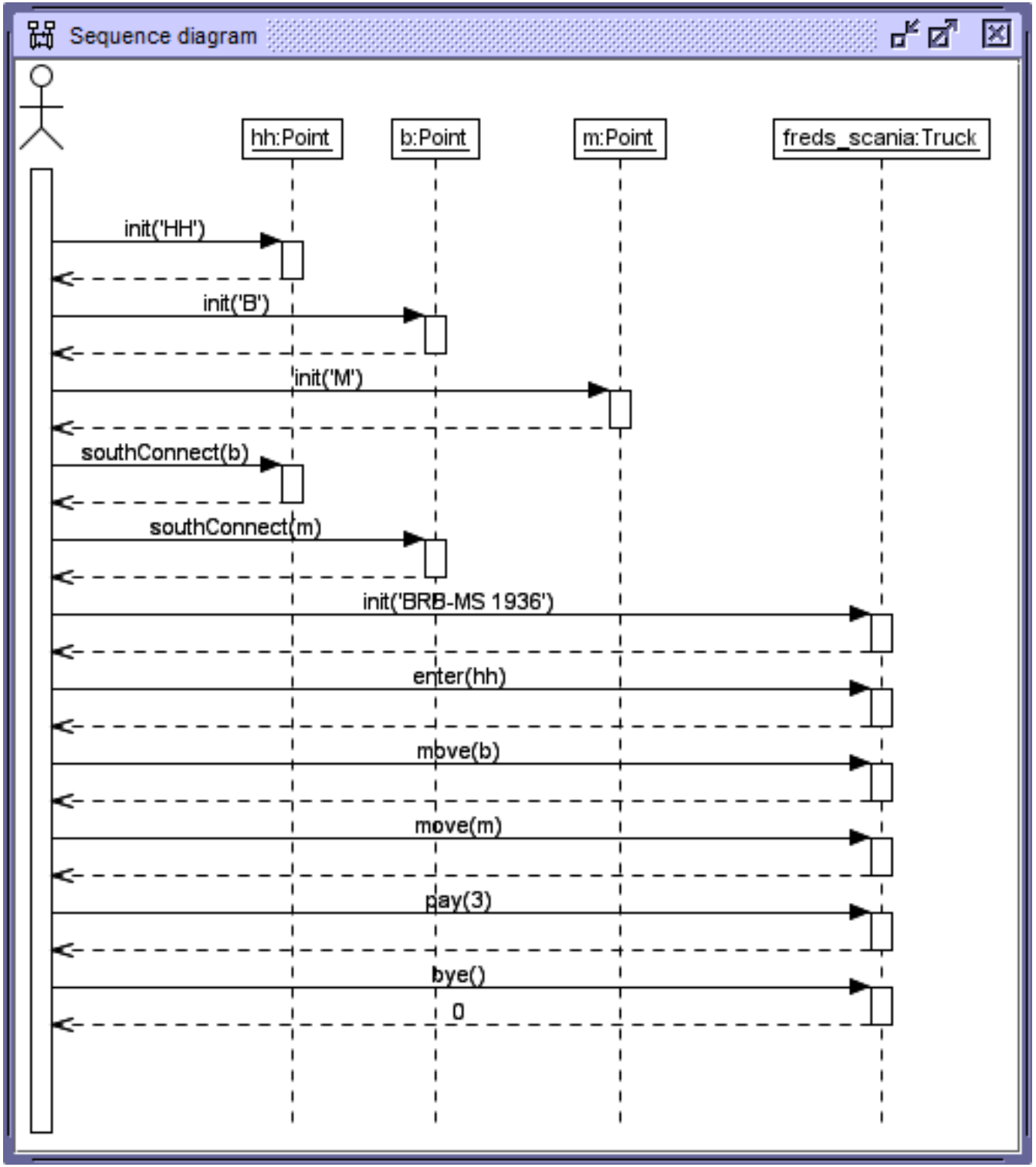
1. !new Point('hh')
2. !hh.init('HH')
3. !new Point('b')
4. !b.init('B')
5. !new Point('m')
6. !m.init('M')
7. !hh.southConnect(b)
8. !b.southConnect(m)
9. !new Truck('freds_scania')
10. !freds_scania.init('BRB-MS 1936')
11. !freds_scania.enter(hh)
12. !freds_scania.move(b)
13. !freds_scania.move(m)

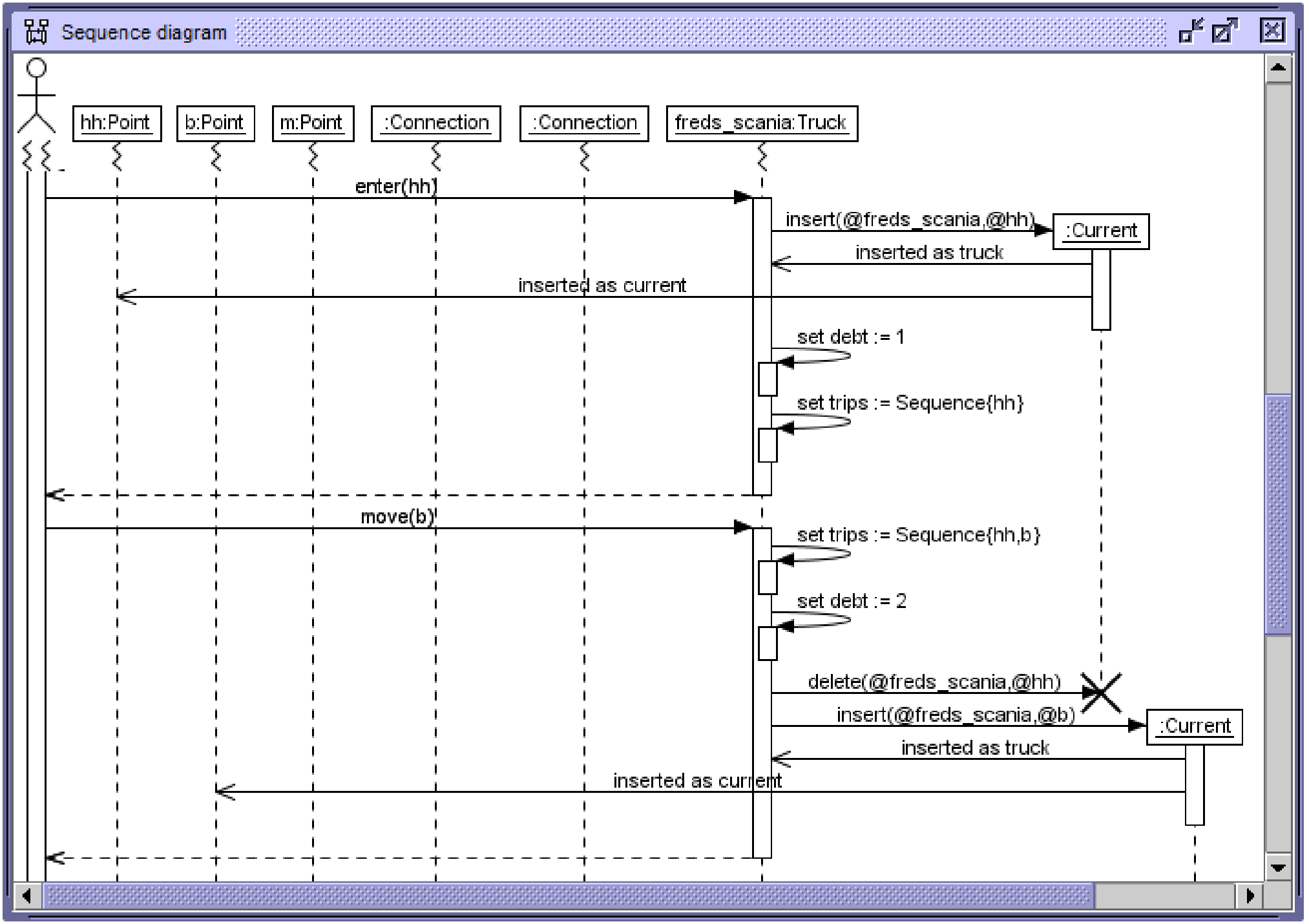


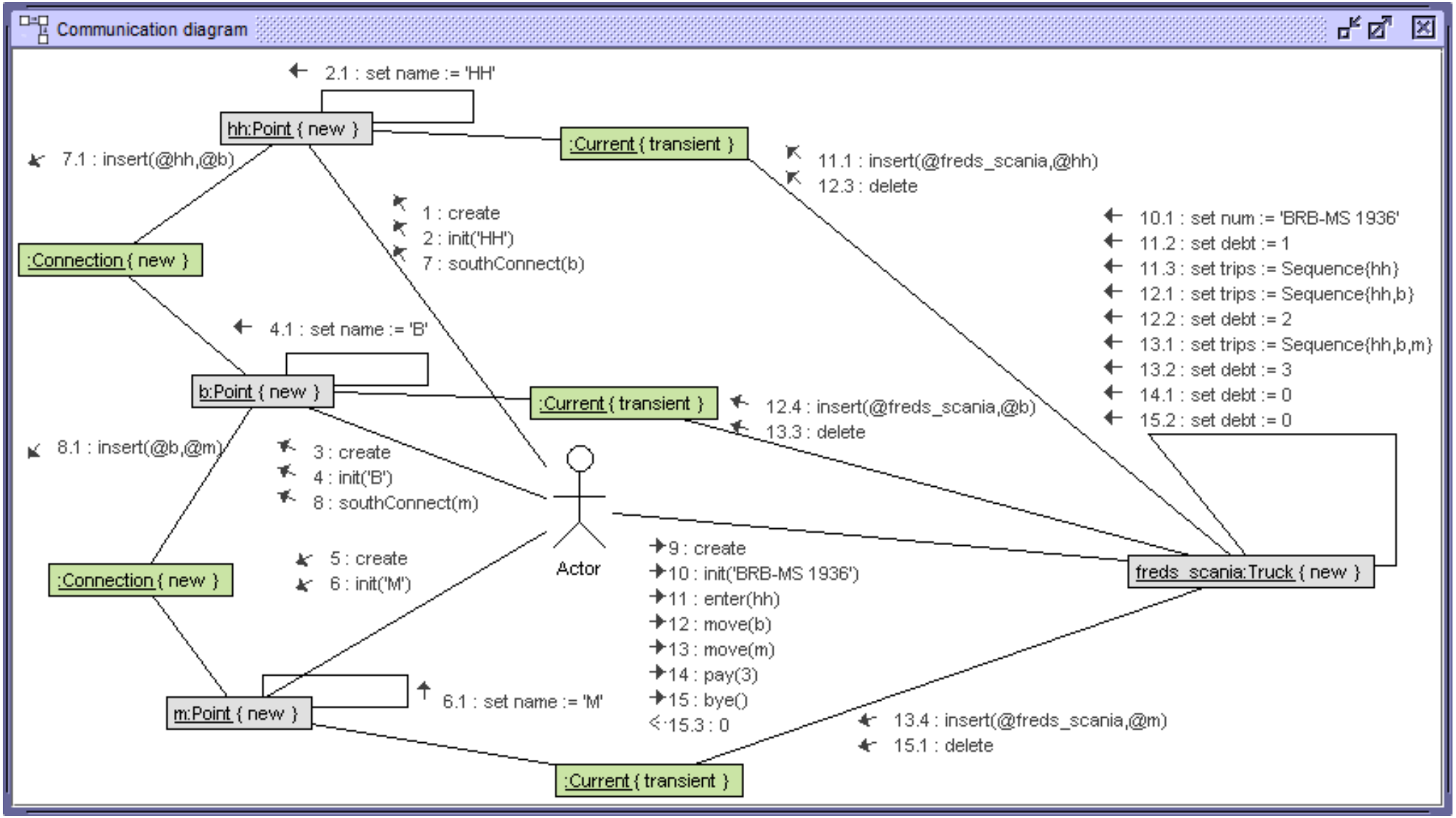
Command list

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()

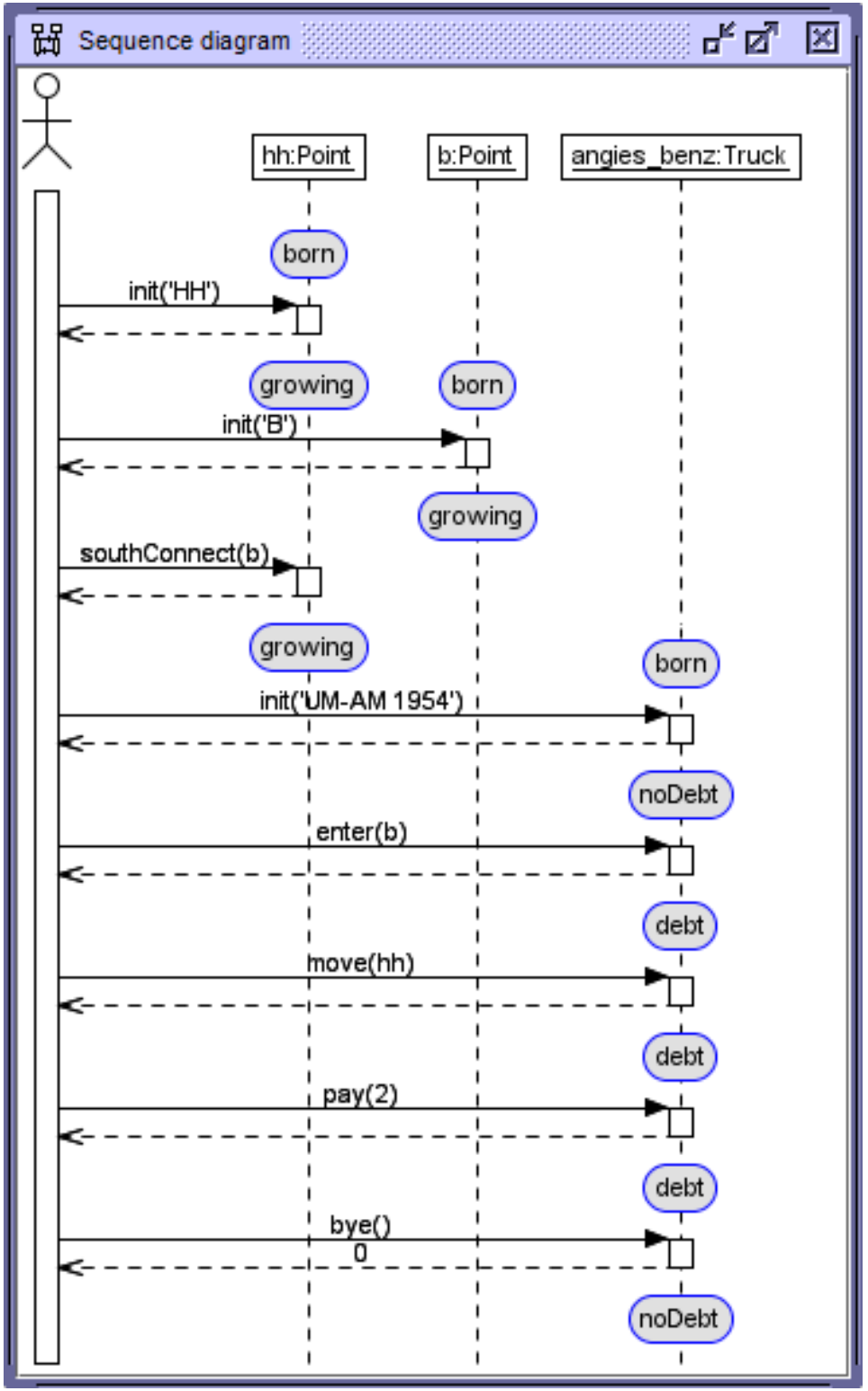

Sequence diagram
hh:Point
b:Point
m:Point
freds_scania:Truck
init('HH')
init('B')
init('M')
southConnect(b)
southConnect(m)
init('BRB-MS 1936')
enter(hh)
move(b)
move(m)
pay(3)
bye()
0


Sequence diagram
hh:Point
b:Point
m:Point
:Connection
:Connection
freds_scania:Truck
enter(hh)
insert(@freds_scania,@hh)
:Current
inserted as truck
inserted as current
set debt := 1
set trips := Sequence{hh}
move(b)
set trips := Sequence{hh,b}
set debt := 2
delete(@freds_scania,@hh)
insert(@freds_scania,@b)
:Current
inserted as truck
inserted as current

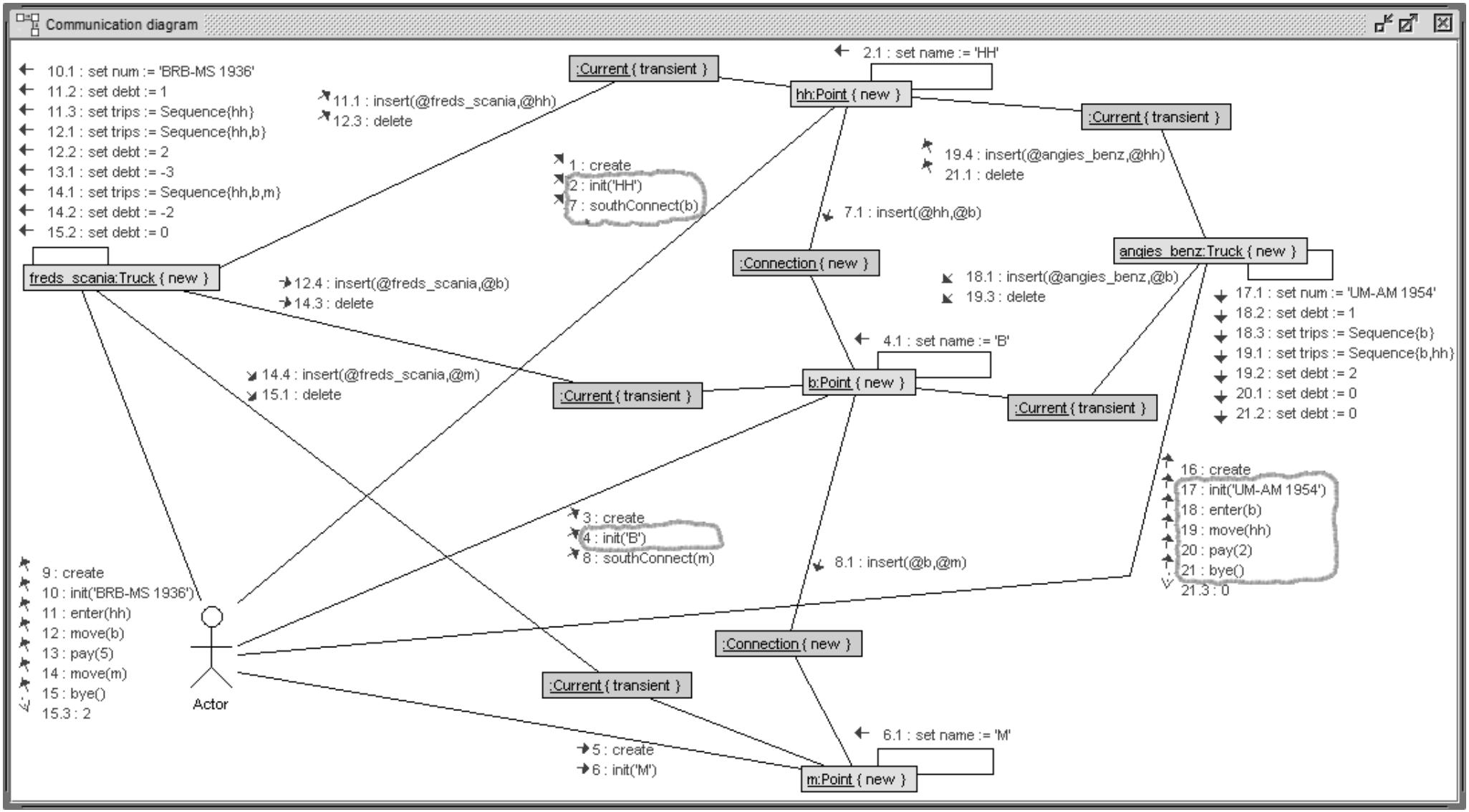

Communication diagram
2.1 : set name := 'HH'
hh:Point { new }
:Current { transient }
7.1 : insert(@hh,@b)
11.1 : insert(@freds_scania,@hh)
12.3 : delete
1 : create
2 : init('HH')
7 : southConnect(b)
10.1 : set num := 'BRB-MS 1936'
11.2 : set debt := 1
11.3 : set trips := Sequence{hh}
12.1 : set trips := 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
:Connection { new }
freds_scania:Truck { new }
m:Point { new }
6.1 : set name := 'M'
13.4 : insert(@freds_scania,@m)
15.1 : delete
:Current { transient }

Messages 1..21 appearing in following communication diagrams

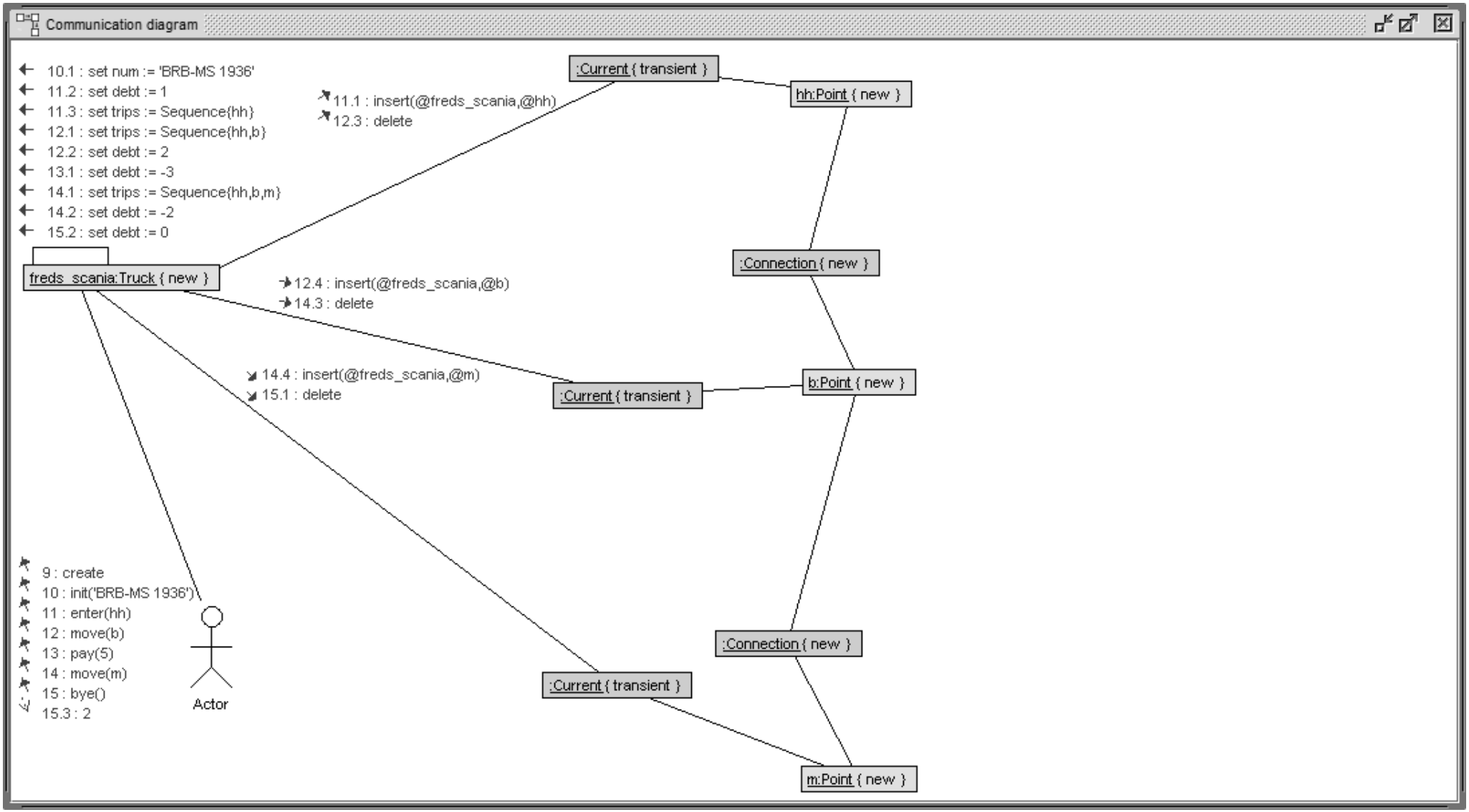
```
create hh:Point
hh.init('HH')
create b:Point
b.init('B')
create m:Point
m.init('M')

hh.southConnect(b)
b.southConnect(m)

create freds_scania:Truck
freds_scania.init('BRB-MS 1936')
freds_scania.enter(hh)
freds_scania.move(b)
freds_scania.pay(5)
freds_scania.move(m)
freds_scania.bye()

create angies_benz:Truck
angies_benz.init('UM-AM 1954')
angies_benz.enter(b)
angies_benz.move(hh)
angies_benz.pay(2)
angies_benz.bye()
```


Sequence diagram
hh:Point
b:Point
angies_benz:Truck
born
init('HH')
growing
born
init('B')
growing
southConnect(b)
growing
born
init('UM-AM 1954')
noDebt
enter(b)
debt
move(hh)
debt
pay(2)
debt
bye()
0
noDebt

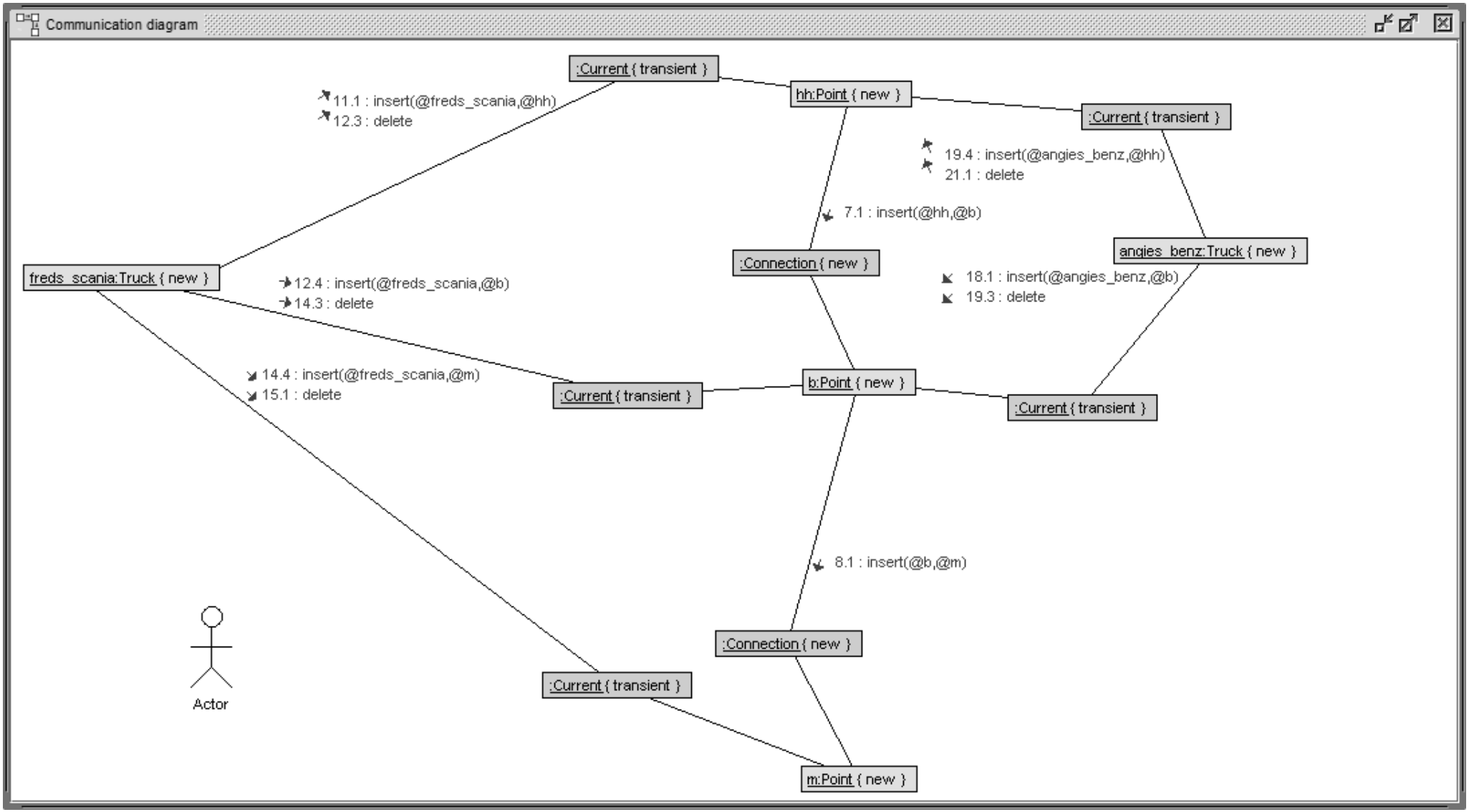
Messages 1..21 - all details displayed



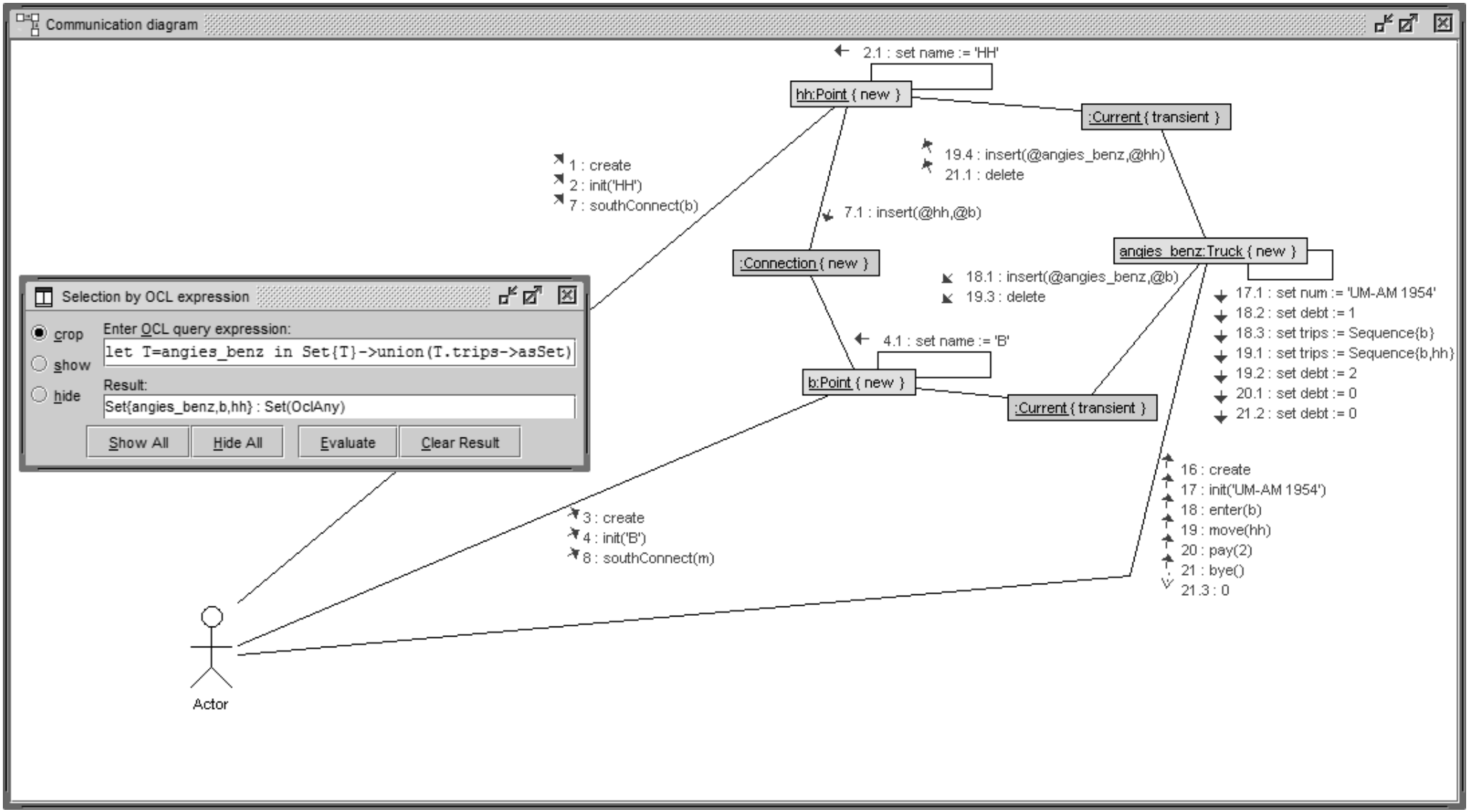
Messages 1..21 - only messages 9..15 displayed



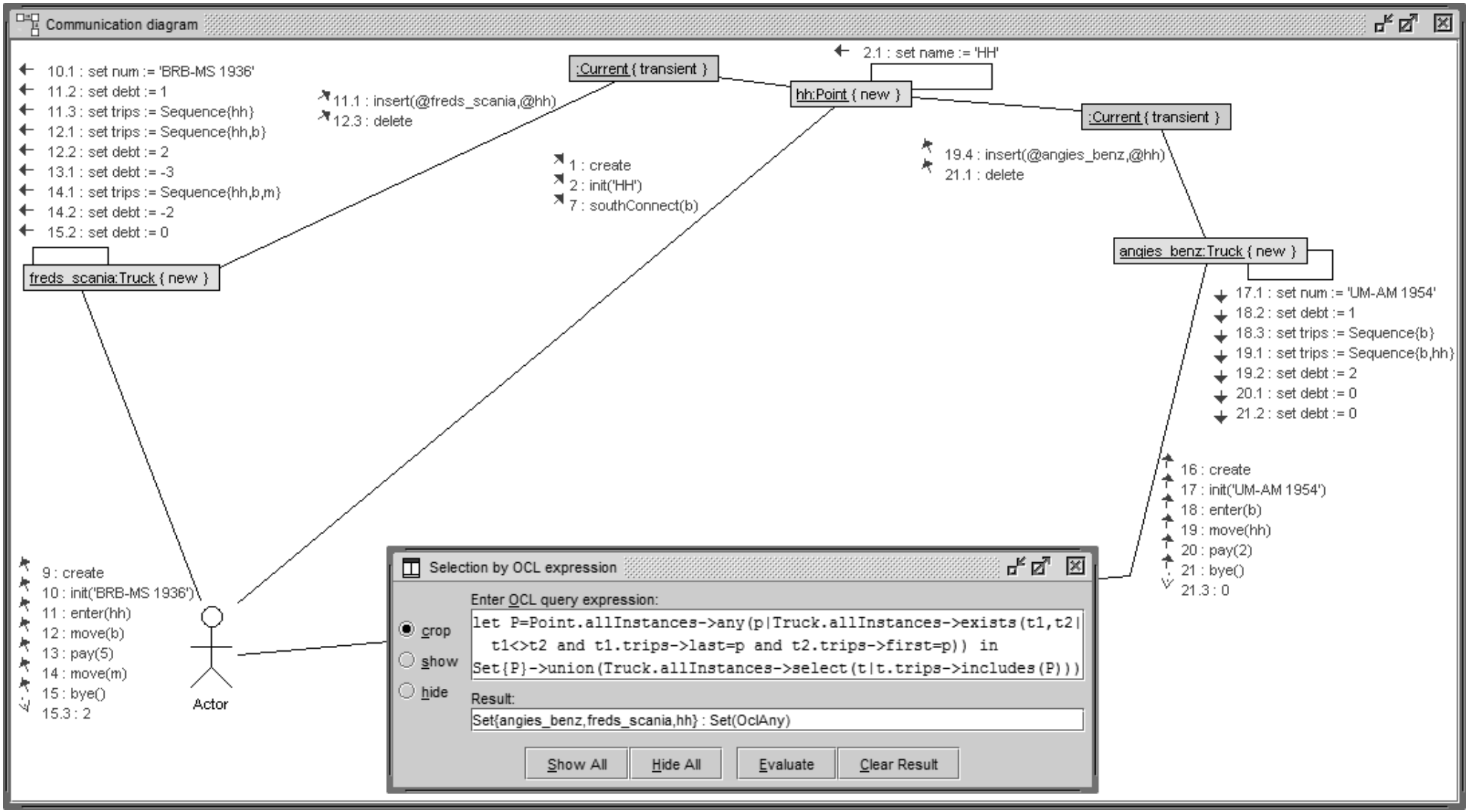
Messages 1..21 - only link insertion and deletion displayed



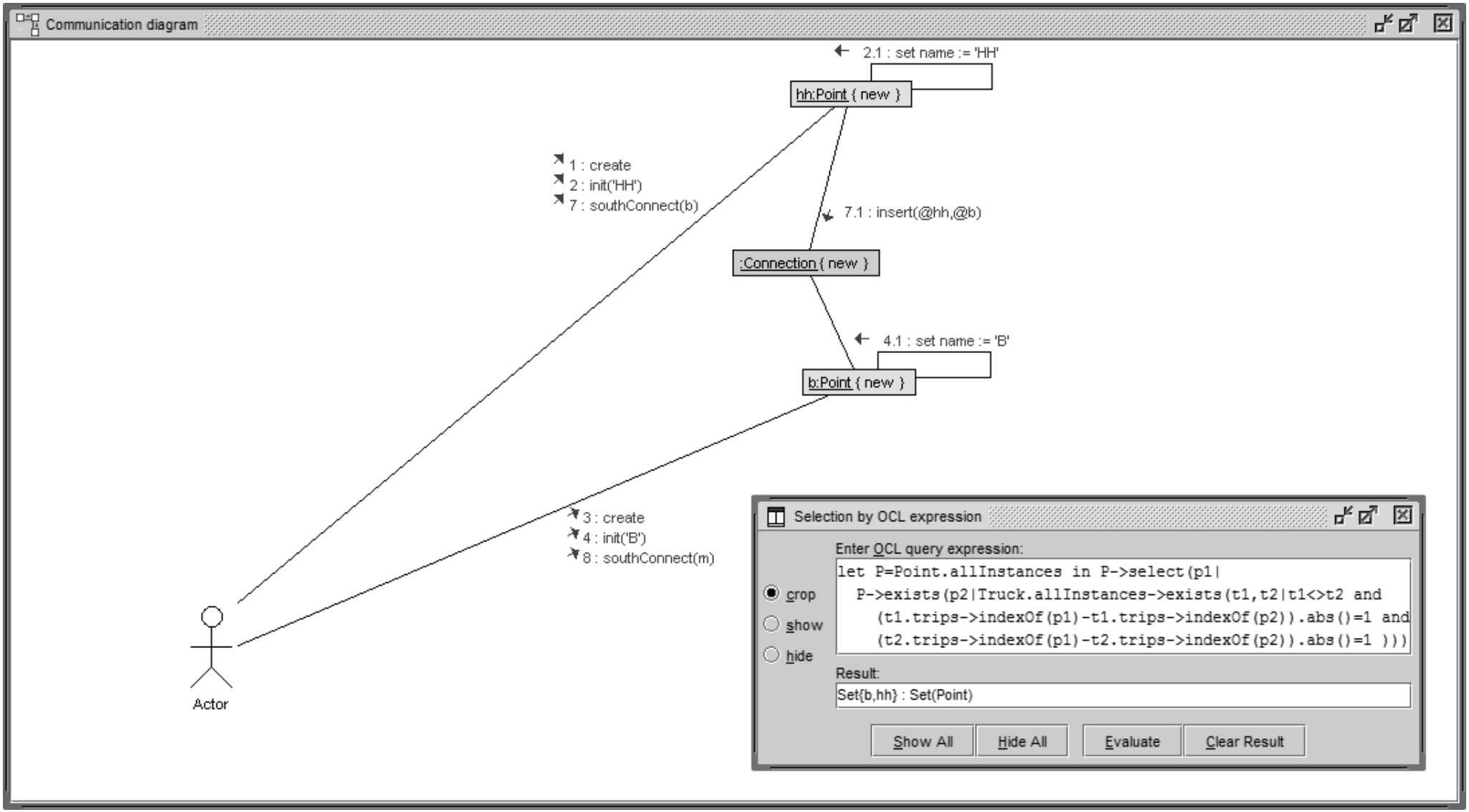
Messages 1..21 - OCL select for truck object identity



Messages 1..21 - OCL select for trucks with coinciding last&first point on trip



Messages 1..21 - OCL select for point connection used twice



Common selection criteria for sequence and communications diagrams

Selection focusing on objects: Objects could be selected through

- **Interactive hide or show for objects**

- **Objects satisfying resp. violating an OCL invariant during interaction**

- **Objects satisfying resp. violating an ad-hoc OCL formula during interaction**

Common selection criteria for sequence and communications diagrams

Selection focusing on messages: Messages could be selected through

- **Interactive hide or show for messages**

- **Selection through an OCL object query identifying the sending object**

- **Selection through a satisfied resp. violated OCL pre- or postcondition**

- **Selection through a satisfied resp. violated ad-hoc OCL formula at pre- or postcondition time during an operation call**

- **Selection by message kind: object creation, object destruction, link insertion, link deletion, attribute assignment, operation call**

- **Selection by message number depth**

- **Determination of a message interval defined by**
  - **interactively fixed start message and end message**
  - **start OCL formula and end OCL formula**
  - **a statechart start state and a statechart end state for a fixed object**

Contribution discussed

- how to handle UML interaction diagrams in a model validation tool

- pointed to the link between protocol machine and interaction diagrams

- developed a desirable feature set for both kinds of
  UML interaction diagrams, namely sequence and communication diagrams

Future work

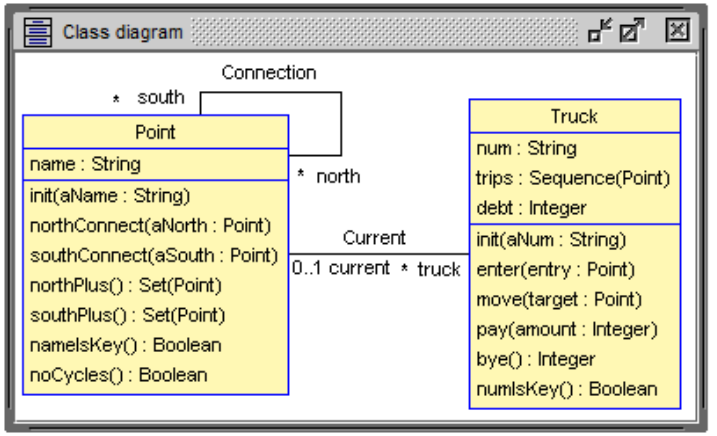
- complete our current implementation with the missing features
  in both interaction diagrams

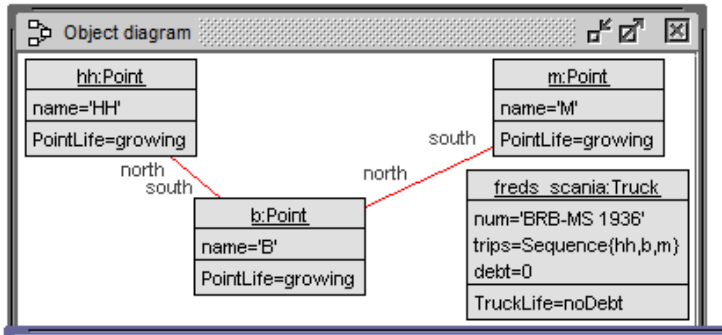
- message selection and message interval selection could offer
  promising analysis options

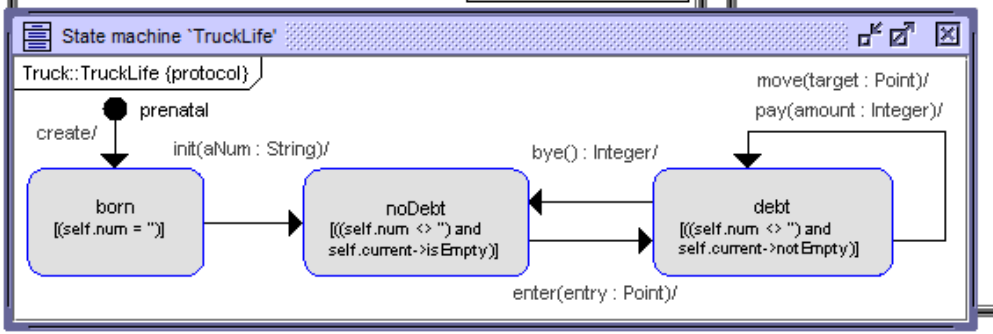
- extend the options for interaction analysis with
  temporal OCL query features

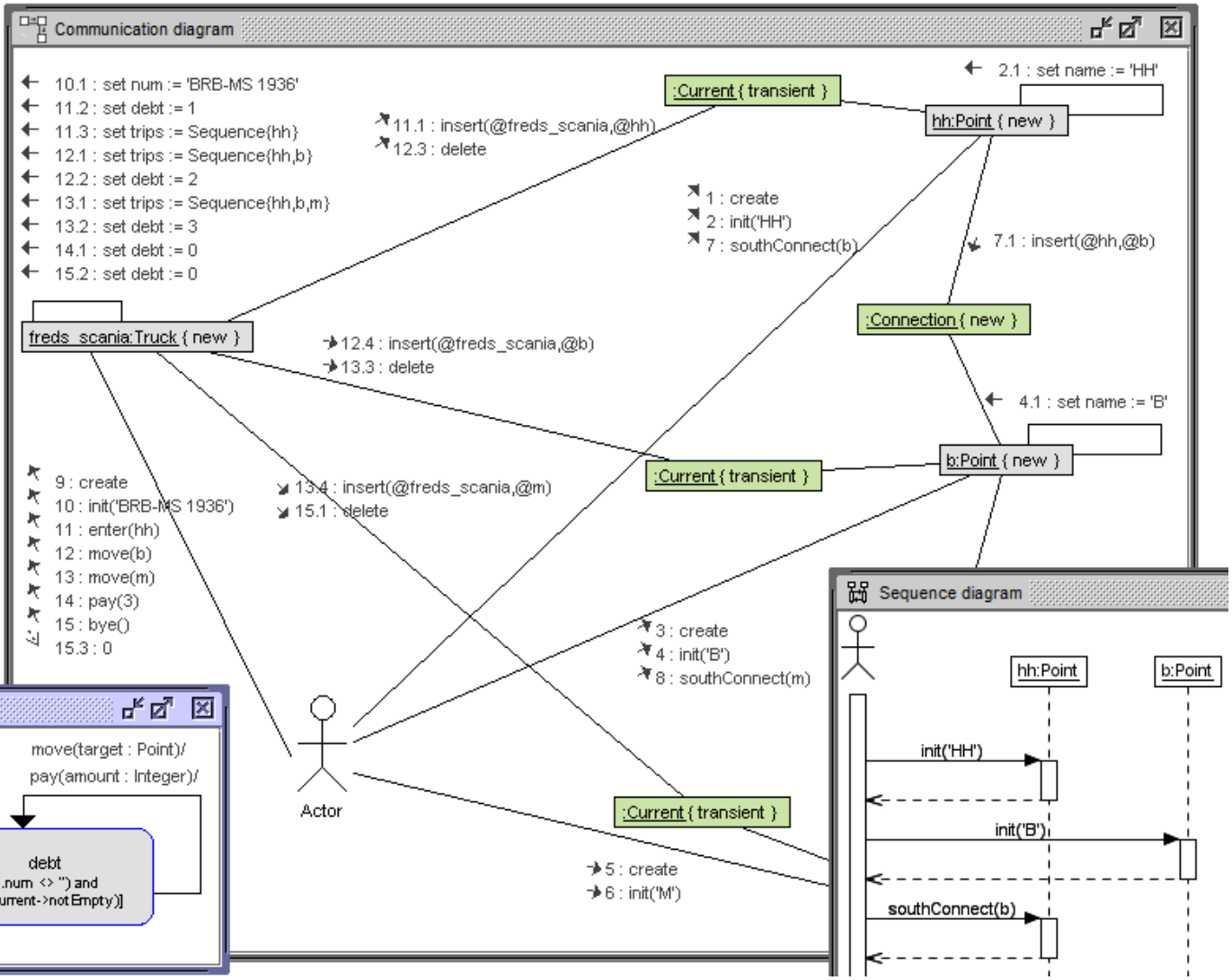
- larger examples and case studies need to validate already
  existing and planned features for better support of interaction diagrams

Thanks for your attention!


Class diagram
Connection
* south
* north
Point
name : String
init(aName : String)
northConnect(aNorth : Point)
southConnect(aSouth : Point)
northPlus() : Set(Point)
southPlus() : Set(Point)
nameIsKey() : Boolean
noCycles() : Boolean
Current
0..1 current * truck
Truck
num : String
trips : Sequence(Point)
debt : Integer
init(aNum : String)
enter(entry : Point)
move(target : Point)
pay(amount : Integer)
bye() : Integer
numIsKey() : Boolean


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


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)/


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

Realized UML/OCL concepts in USE [concepts in brackets not in example]

- class, attribute, datatype, operation, operation signature

- class invariant

- query operation definition with OCL;
  non-query operation with state changes through definition in SOIL

- attribute initialization, derived attribute, [derived association]

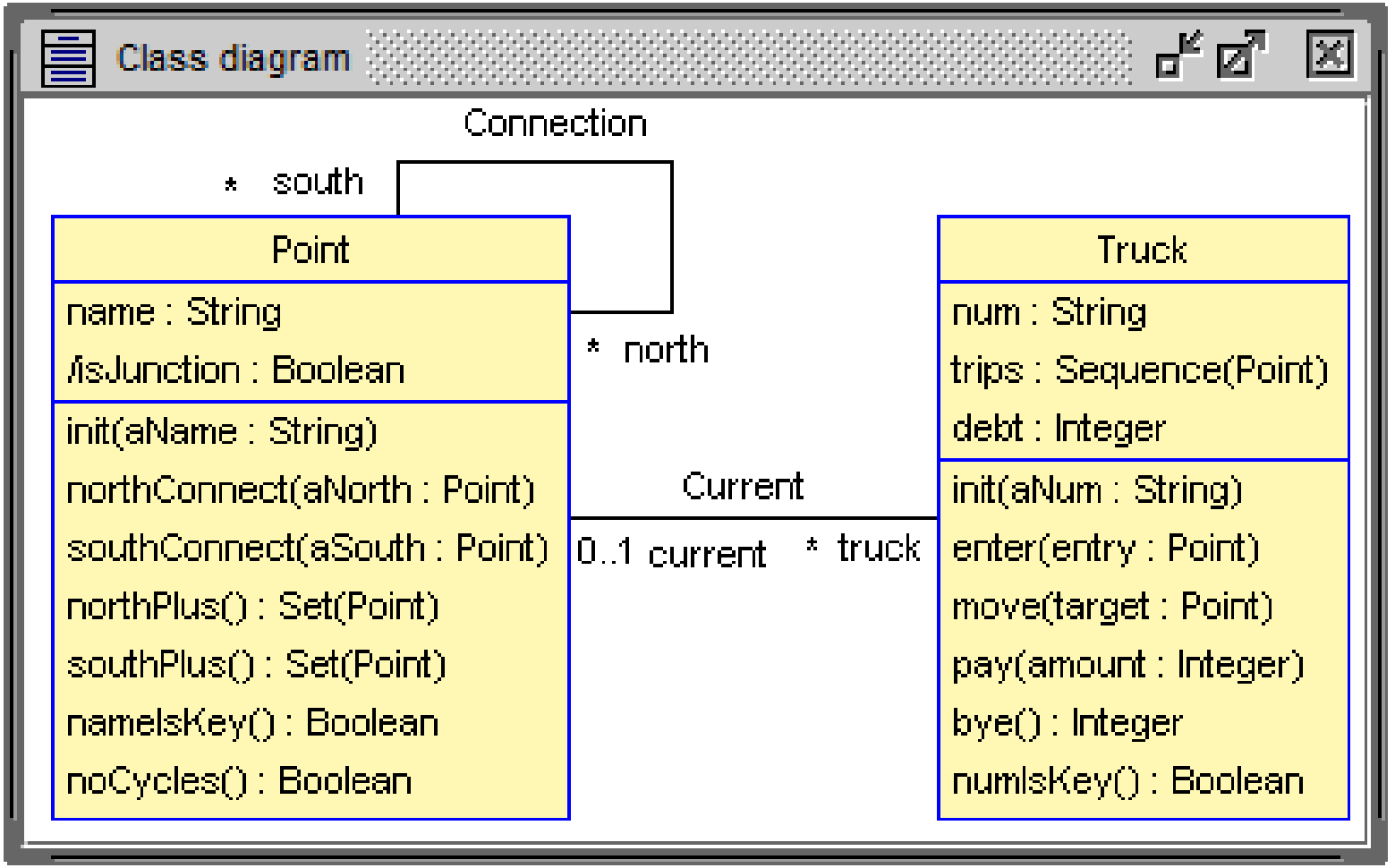
- operation contract, i.e., pre- and postconditions

- association, role names, multiplicities,
  [aggregation, composition, qualified association]

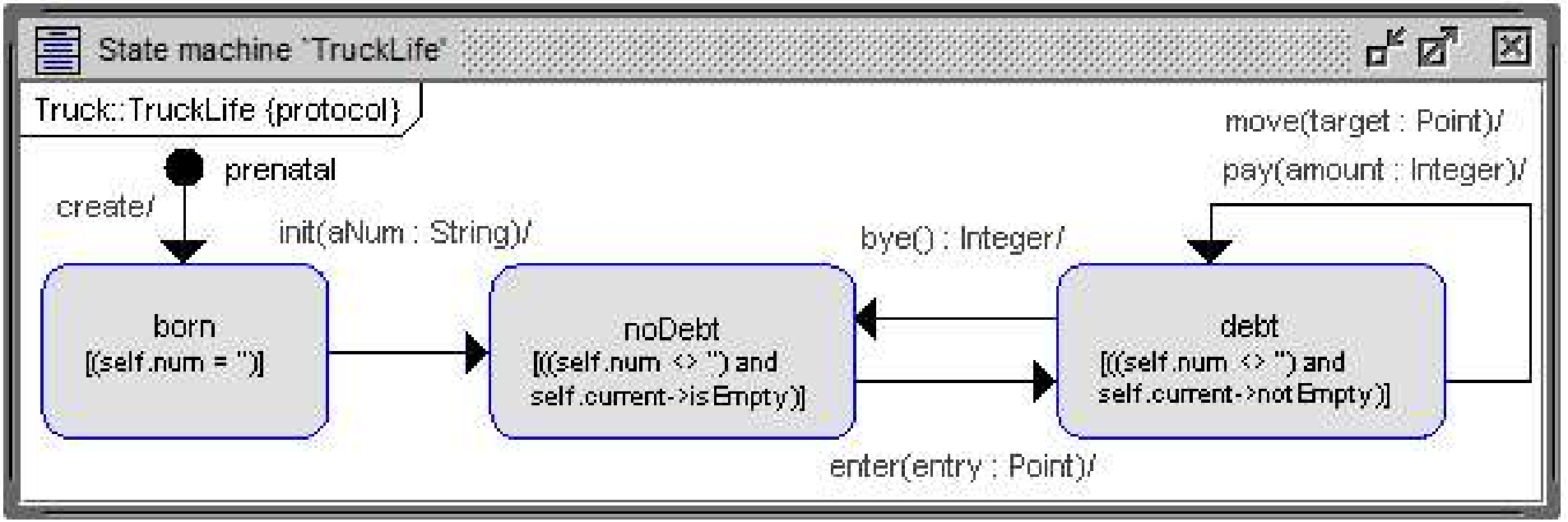
- [generalization]
  [subsets constraint, redefines constraint, union constraint]

- protocol state machines with states and transitions
  state invariants, [transition pre- and postcondition]

```
---------------------------------------------------- model TollCollect
model TollCollect
------------------------------------------------------------ class Truck
class Truck
attributes
  num:String              init: ''
  trips:Sequence(Point) init: Sequence{}
  debt:Integer            init: 0
operations
  init(aNum:String)
    begin self.num:=aNum end
  enter(entry:Point)
    begin insert (self,entry) into Current; self.debt:=1;
    self.trips:=self.trips->including(self.current) end
  move(target:Point)
    begin self.trips:=self.trips->including(target);
    self.debt:=self.debt+1; delete (self,self.current) from Current;
    insert (self,target) into Current end
  pay(amount:Integer)
    begin self.debt:=self.debt-amount end
  bye():Integer
    begin delete (self,self.current) from Current;
    result:=self.debt.abs(); self.debt:=0 end
  ------------------------------------------------------------------
```

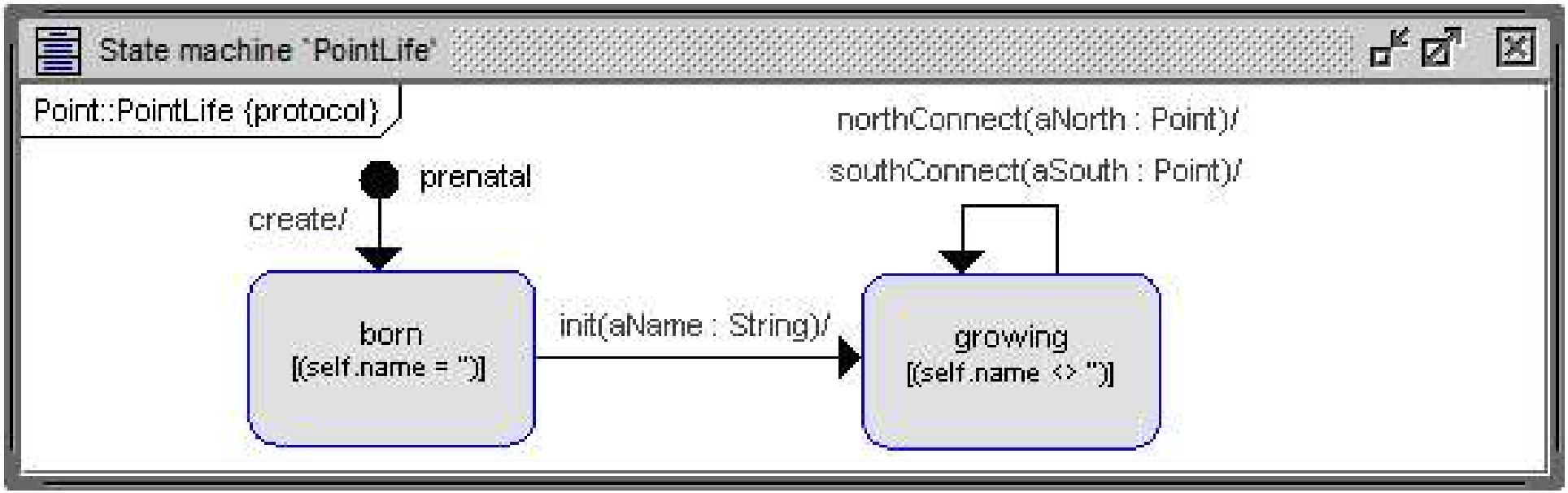

Class diagram
Connection
* south
Point
name : String
/isJunction : Boolean
init(aName : String)
northConnect(aNorth : Point)
southConnect(aSouth : Point)
northPlus() : Set(Point)
southPlus() : Set(Point)
nameIsKey() : Boolean
noCycles() : Boolean
* north
Current
0..1 current
* truck
Truck
num : String
trips : Sequence(Point)
debt : Integer
init(aNum : String)
enter(entry : Point)
move(target : Point)
pay(amount : Integer)
bye() : Integer
numIsKey() : Boolean

```
  ------------------------------------------------------------------
  numIsKey():Boolean=
    Truck.allInstances->forAll(self,self2|
      self<>self2 implies self.num<>self2.num)
  ------------------------------------------------------------------
statemachines
  psm TruckLife
  states
    prenatal:initial
    born   [num='']
    noDebt [num<>'' and current->isEmpty]
    debt   [num<>'' and current->notEmpty]
  transitions
    prenatal -> born   { create }
    born     -> noDebt { init() }
    noDebt   -> debt   { enter() }
    debt     -> debt   { move() }
    debt     -> debt   { pay() }
    debt     -> noDebt { bye() }
  end
end
```


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)/

```
------------------------------------------------------------ class Point
class Point
attributes
  name:String init: ''
  --northSize:Integer derived: self.northPlus()->size()
  --southSize:Integer derived: self.southPlus()->size()
operations
  init(aName:String)
    begin self.name:=aName end
  northConnect(aNorth:Point)
    begin insert (aNorth,self) into Connection end
  southConnect(aSouth:Point)
    begin insert (self,aSouth) into Connection end
  ------------------------------------------------------------------
  northPlus():Set(Point)=self.north->closure(p|p.north)
  southPlus():Set(Point)=self.south->closure(p|p.south)
  ------------------------------------------------------------------
  nameIsKey():Boolean=
    Point.allInstances->forAll(self,self2|
      self<>self2 implies self.name<>self2.name)
  noCycles():Boolean=
    Point.allInstances->forAll(self|
      not(self.northPlus()->includes(self)))
  ------------------------------------------------------------------
```

```
statemachines
  psm PointLife
  states
    prenatal:initial
    born    [name='']
    growing [name<>'']
  transitions
    prenatal -> born    { create }
    born     -> growing { init() }
    growing  -> growing { northConnect() }
    growing  -> growing { southConnect() }
  end
end

------------------------------------------------- association Current
association Current between
  Truck[0..*] role truck
  Point[0..1] role current
end
------------------------------------------------- association Connection
association Connection between
  Point[0..*] role north
  Point[0..*] role south
end
```


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

```
---------------------------------------------------------- constraints
constraints
---------------------------------------------------------- invariants
context Truck inv numIsKeyInv:
  numIsKey()
context Point inv nameIsKeyInv:
  nameIsKey()
context Point inv noCyclesInv:
  noCycles()
---------------------------------------------------------- Point::init
context Point::init(aName:String)
pre freshPoint:
  self.name='' and self.north->isEmpty and self.south->isEmpty
pre aNameOk:
  aName<>'' and aName<>null
post nameAssigned:
  aName=self.name
post allPointInvs:
  nameIsKey() and noCycles()
```

```
---------------------------------------------- Point::northConnect
context Point::northConnect(aNorth:Point)
pre aNorthDefined:
  aNorth.isDefined
pre freshConnection:
  self.north->excludes(aNorth) and self.south->excludes(aNorth)
pre notSelfLink:
  self<>aNorth
pre noCycleIntroduced:
  aNorth.northPlus()->excludes(self)
post connectionAssigned:
  self.north->includes(aNorth)
post allPointInvs:
  nameIsKey() and noCycles()
---------------------------------------------------- Truck::init
context Point::southConnect(aSouth:Point)
pre aSouthDefined:
  aSouth.isDefined
pre freshConnection:
  self.south->excludes(aSouth) and self.south->excludes(aSouth)
pre notSelfLink:
  self<>aSouth
pre noCycleIntroduced:
  aSouth.southPlus()->excludes(self)
post connectionAssigned:
  self.south->includes(aSouth)
post allPointInvs:
  nameIsKey() and noCycles()
```

```
---------------------------------------------------------- Truck::init
context Truck::init(aNum:String)
pre freshTruck:
  self.num='' and self.trips=Sequence{} and self.debt=0 and
  self.current->isEmpty
pre aNumOk:
  aNum<>'' and aNum<>null
post numAssigned:
  aNum=self.num
post allTruckInvs:
  numIsKey()
---------------------------------------------------------- Truck::enter
context Truck::enter(entry:Point)
pre noDebt:
  0=self.debt
pre currentEmpty:
  self.current->isEmpty
pre entryOk:
  entry<>null
post debtAssigned:
  1=self.debt
post tripsUpdated:
  self.trips@pre->including(entry)=self.trips
post currentAssigned:
  entry=self.current
post allTruckInvs:
  numIsKey()
```

```
------------------------------------------------------------ Truck::move
context 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()
------------------------------------------------------------ Truck::pay
context Truck::pay(amount:Integer)
pre amountPositive:
  amount>0
pre currentExists:
  self.current->notEmpty
post debtReduced:
  (self.debt@pre-amount)=(self.debt)
post allTruckInvs:
  numIsKey()
```

```
------------------------------------------------------------ Truck::move
context 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()

------------------------------------------------------------ Truck::pay
context Truck::pay(amount:Integer)
pre amountPositive: amount>0
pre currentExists: self.current->notEmpty
post debtReduced: (self.debt@pre-amount)=(self.debt)
post allTruckInvs: numIsKey()

------------------------------------------------------------ Truck::bye
context Truck::bye():Integer
pre currentExists: self.current->notEmpty
pre noDebt: self.debt<=0
post resultEqualsOverPayment: self.debt@pre.abs()=result
post zeroDebt: self.debt=0
post currentEmpty: self.current->isEmpty
post allTruckInvs: numIsKey()
```