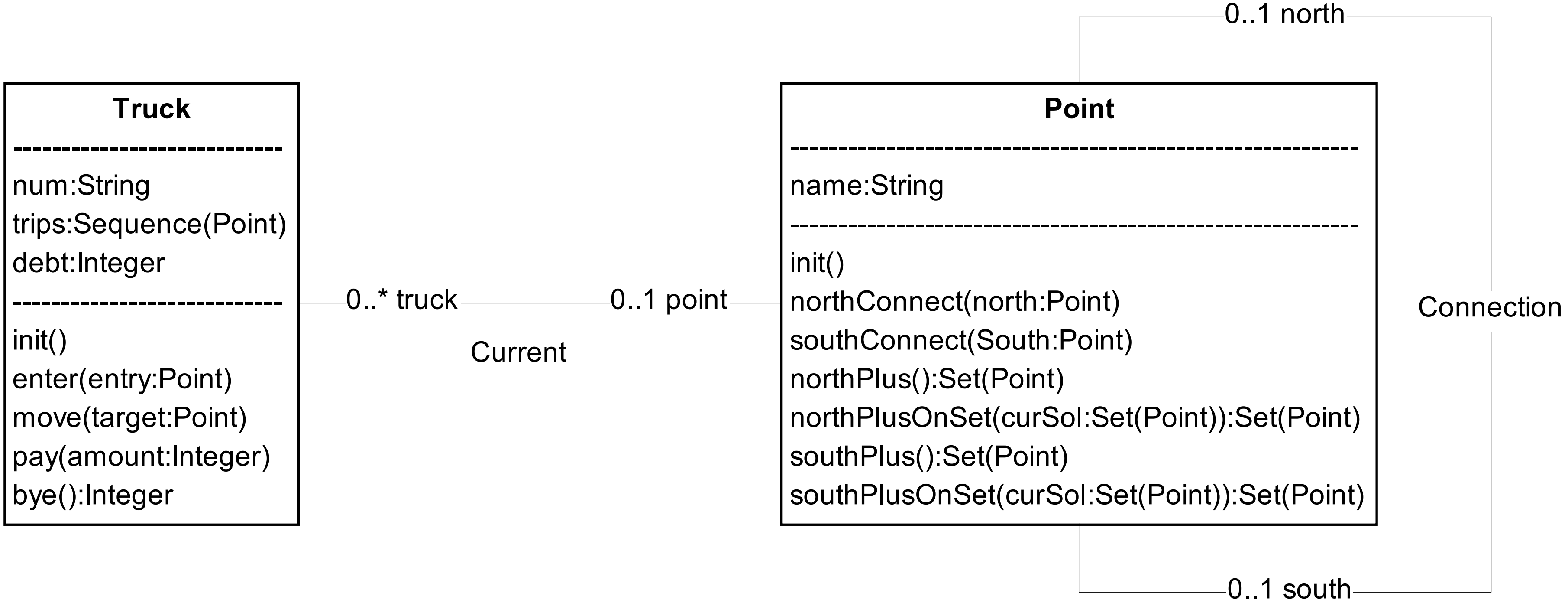
Toll Collect:
A UML Case Study realized with USE

martin gogolla

- Class diagram and statecharts
- Sequence and object diagram for underlying road graph
- Details of classes Point and Truck
- Associations Connection and Current
- Example scenario 'Fred drives from Hamburg to Munich'
- Overview on invariants and pre- and postconditions
- Details of invariants and pre- and postconditions
- Operations implementations as command sequences
- Excerpts from protocol file



Truck

----

num:String
trips:Sequence(Point)
debt:Integer

----

init()
enter(entry:Point)
move(target:Point)
pay(amount:Integer)
bye():Integer

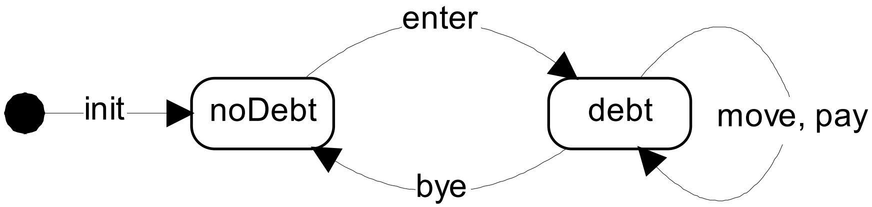
Point

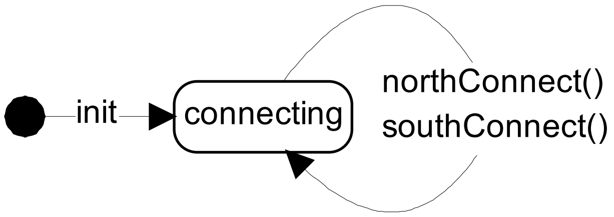
----

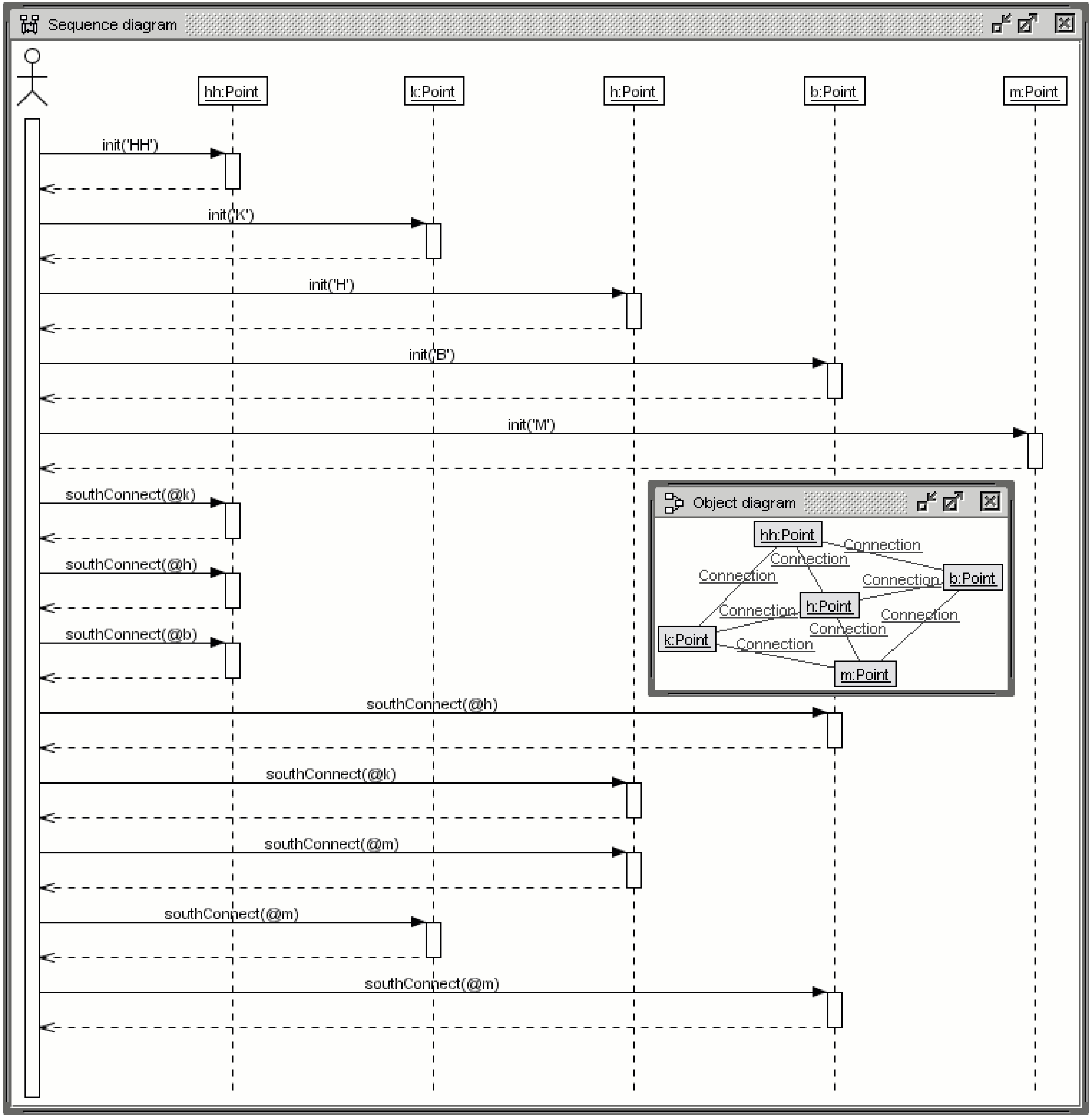
name:String

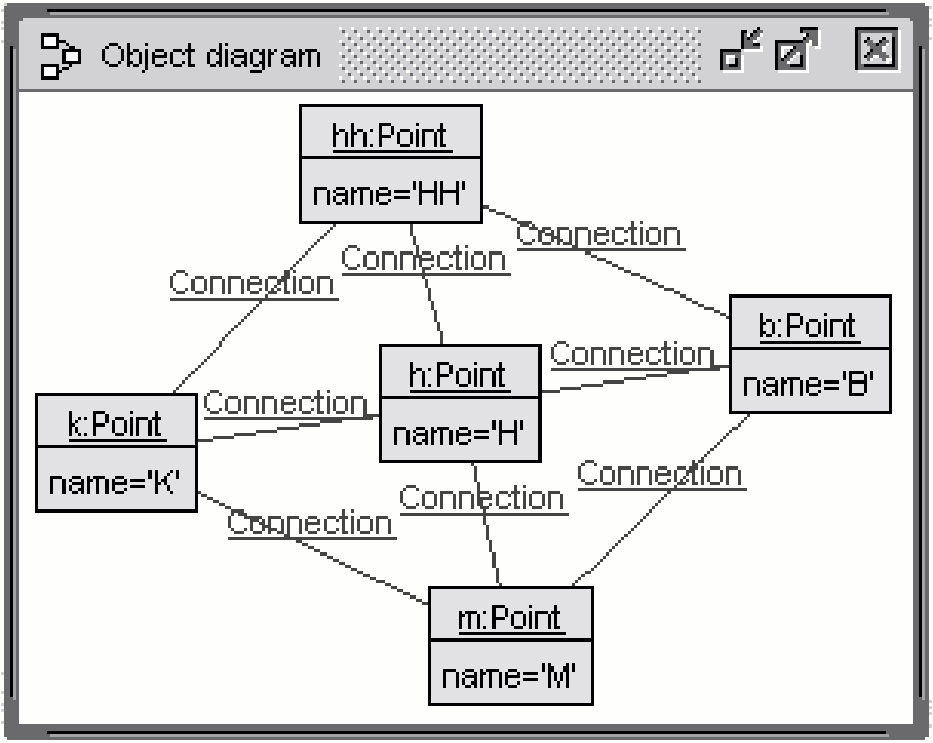
----

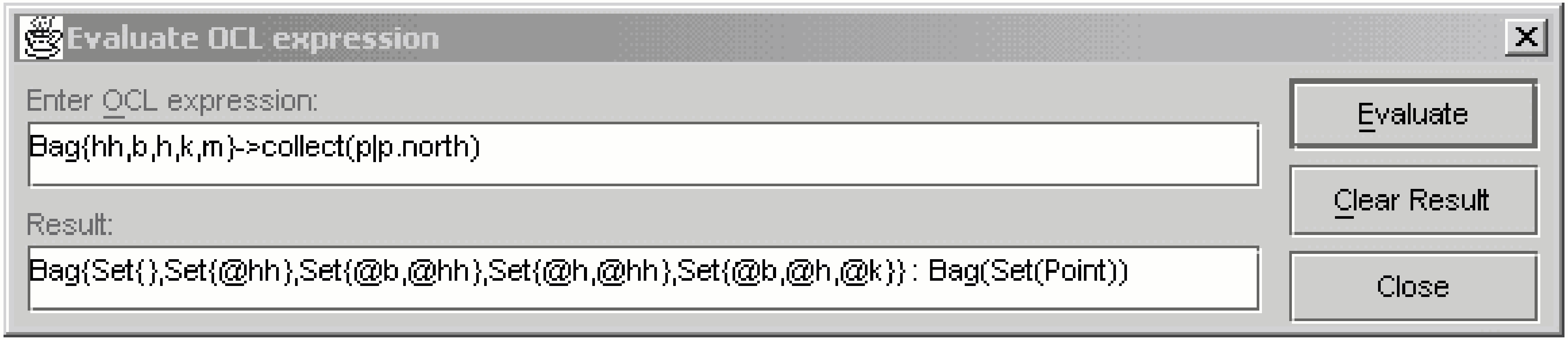
init()
northConnect(north:Point)
southConnect(South:Point)
northPlus():Set(Point)
northPlusOnSet(curSol:Set(Point)):Set(Point)
southPlus():Set(Point)
southPlusOnSet(curSol:Set(Point)):Set(Point)






Sequence diagram
hh:Point
k:Point
h:Point
b:Point
m:Point
init('HH')
init('K')
init('H')
init('B')
init('M')
southConnect(@k)
southConnect(@h)
southConnect(@b)
southConnect(@h)
southConnect(@k)
southConnect(@m)
southConnect(@m)
southConnect(@m)
Object diagram
hh:Point
b:Point
h:Point
k:Point
m:Point
Connection
Connection
Connection
Connection
Connection
Connection
Connection
Connection



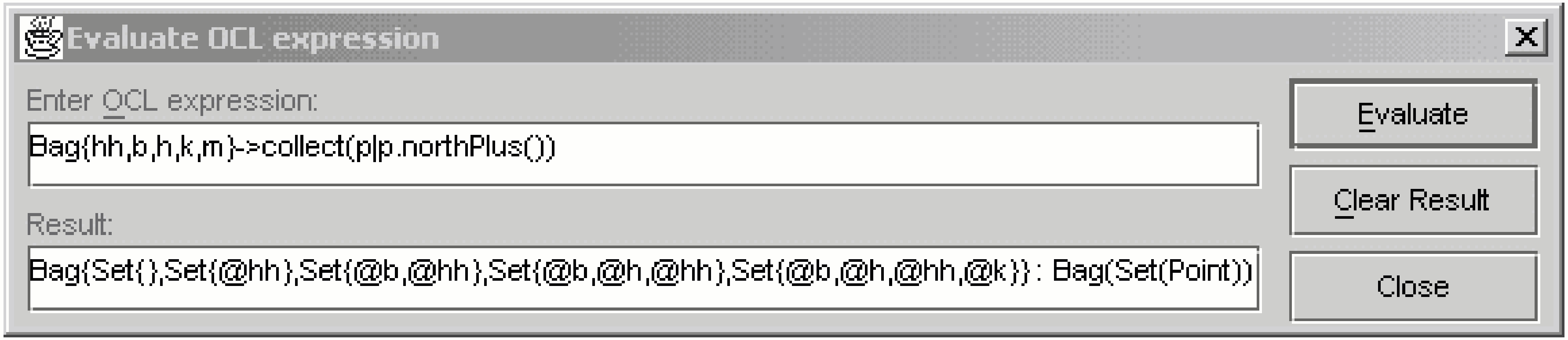


Enter OCL expression:

```
Bag{hh,b,h,k,m}->collect(p|p.north)
```

Result:

```
Bag{Set{},Set{@hh},Set{@b,@hh},Set{@h,@hh},Set{@b,@h,@k}} : Bag(Set(Point))
```



Enter OCL expression:

```
Bag{hh,b,h,k,m}->collect(p|p.northPlus())
```

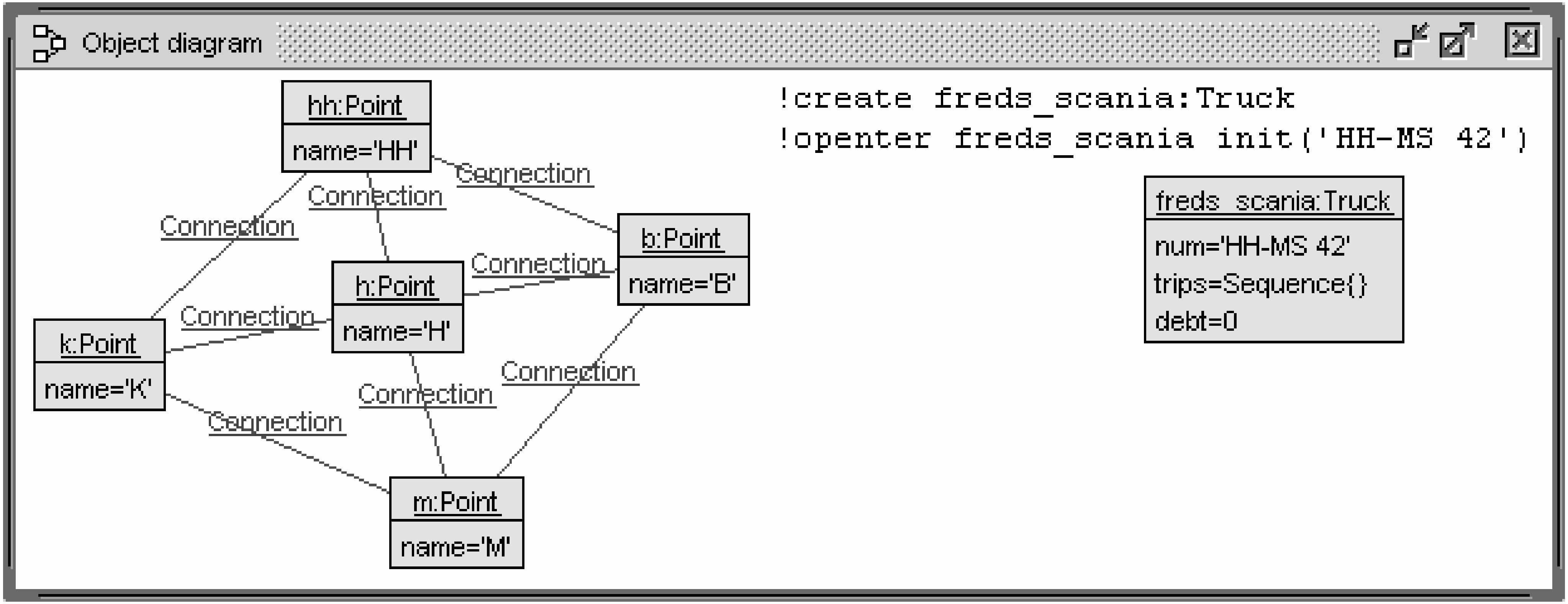
Result:

```
Bag{Set{},Set{@hh},Set{@b,@hh},Set{@b,@h,@hh},Set{@b,@h,@hh,@k}} : Bag(Set(Point))
```

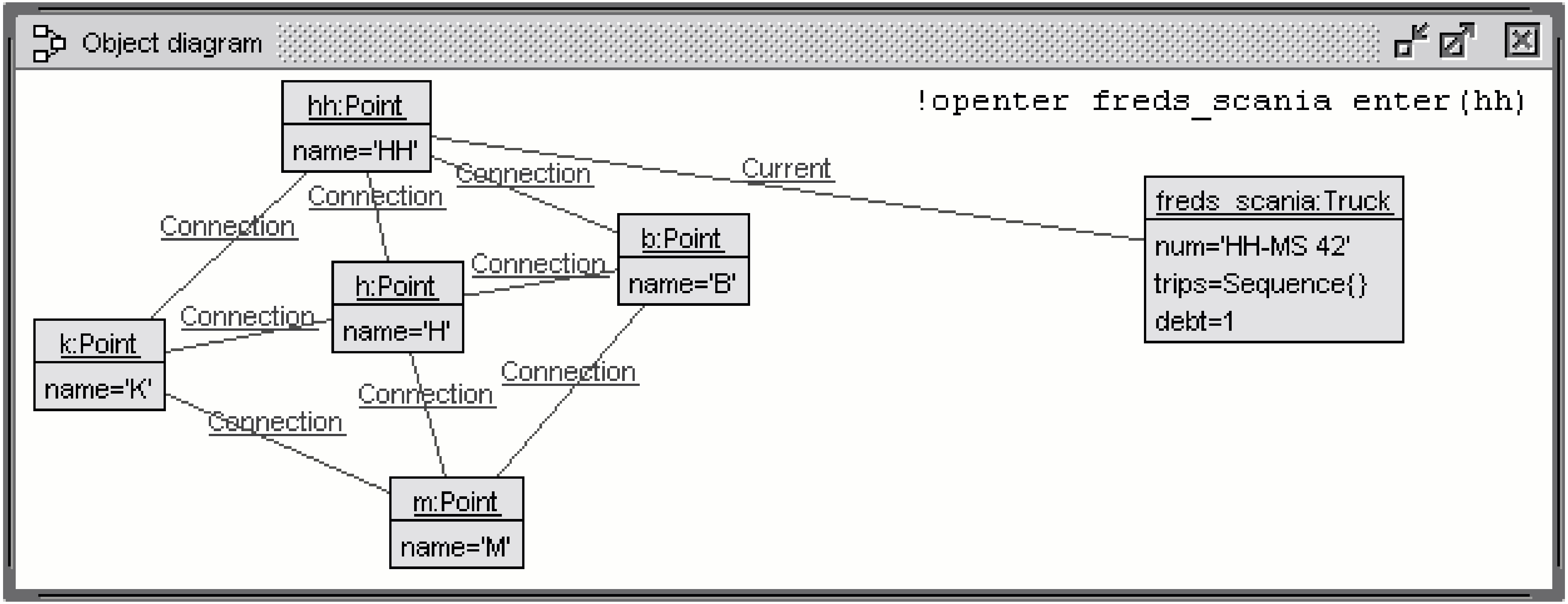
```
------------------------------------------------------------ class Point
class Point
attributes
  name:String
operations
  init(aName:String)
  northConnect(aNorth:Point)
  southConnect(aSouth:Point)
  ----------------------------------------------------------------------
  northPlus():Set(Point)=
    northPlusOnSet(self.north)
  northPlusOnSet(curSol:Set(Point)):Set(Point)= -- current solution
    let oneStep:Set(Point)=
      curSol->collect(p|p.north)->flatten->asSet in
    if oneStep->exists(p|curSol->excludes(p))
      then northPlusOnSet(curSol->union(oneStep))
      else curSol endif
  southPlus() ...
  southPlusOnSet(curSol:Set(Point)) ...
  ----------------------------------------------------------------------
  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)))
end
```

```
------------------------------------------------------------ class Truck
class Truck
attributes
  num:String
  trips:Sequence(Point)
  debt:Integer
operations
  init(aNum:String)
  enter(entry:Point)
  move(target:Point)
  pay(amount:Integer)
  bye():Integer
  numIsKey():Boolean=
    Truck.allInstances->forAll(self,self2|
      self<>self2 implies self.num<>self2.num)
end
```

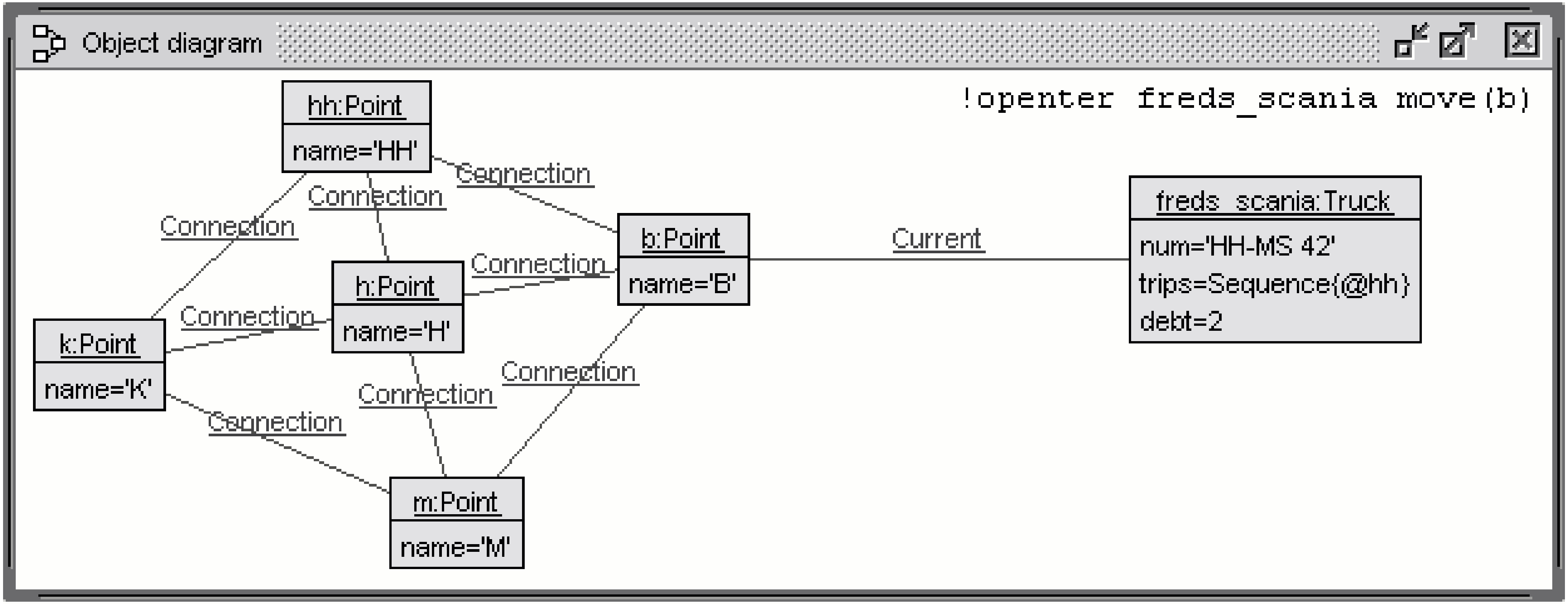
```
---------------------------------------------------- association Current
association Current between
  Truck[0..*]
  Point[0..1]
end
------------------------------------------------- association Connection
association Connection between
  Point[0..*] role north
  Point[0..*] role south
end
```



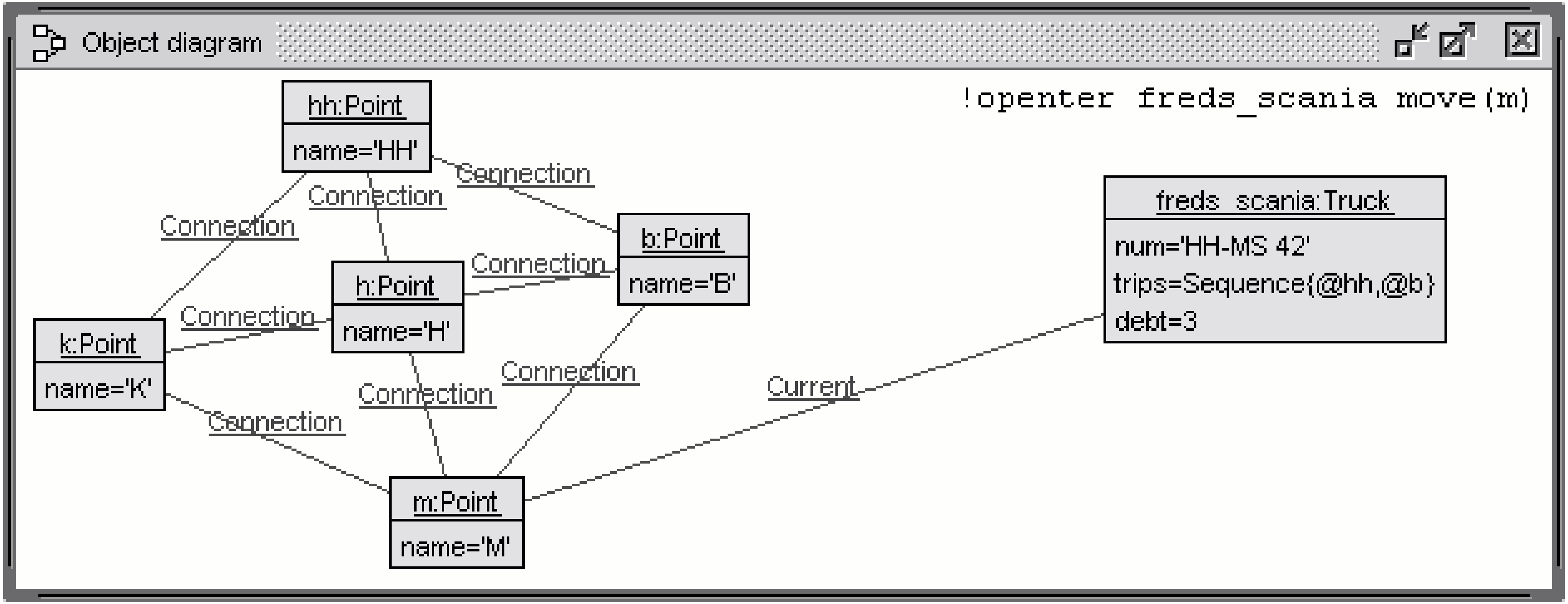
```
!create freds_scania:Truck
!openter freds_scania init('HH-MS 42')
```



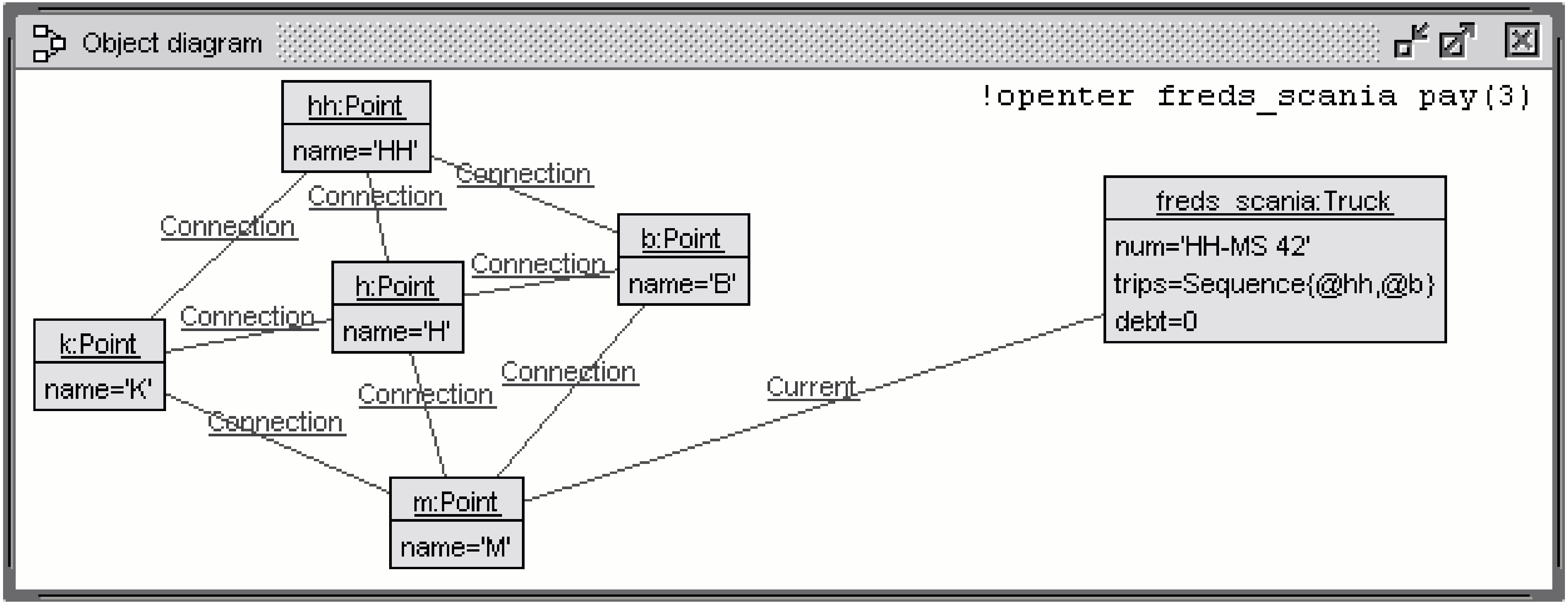
```
!openter freds_scania enter(hh)
```



```
!openter freds_scania move(b)
```

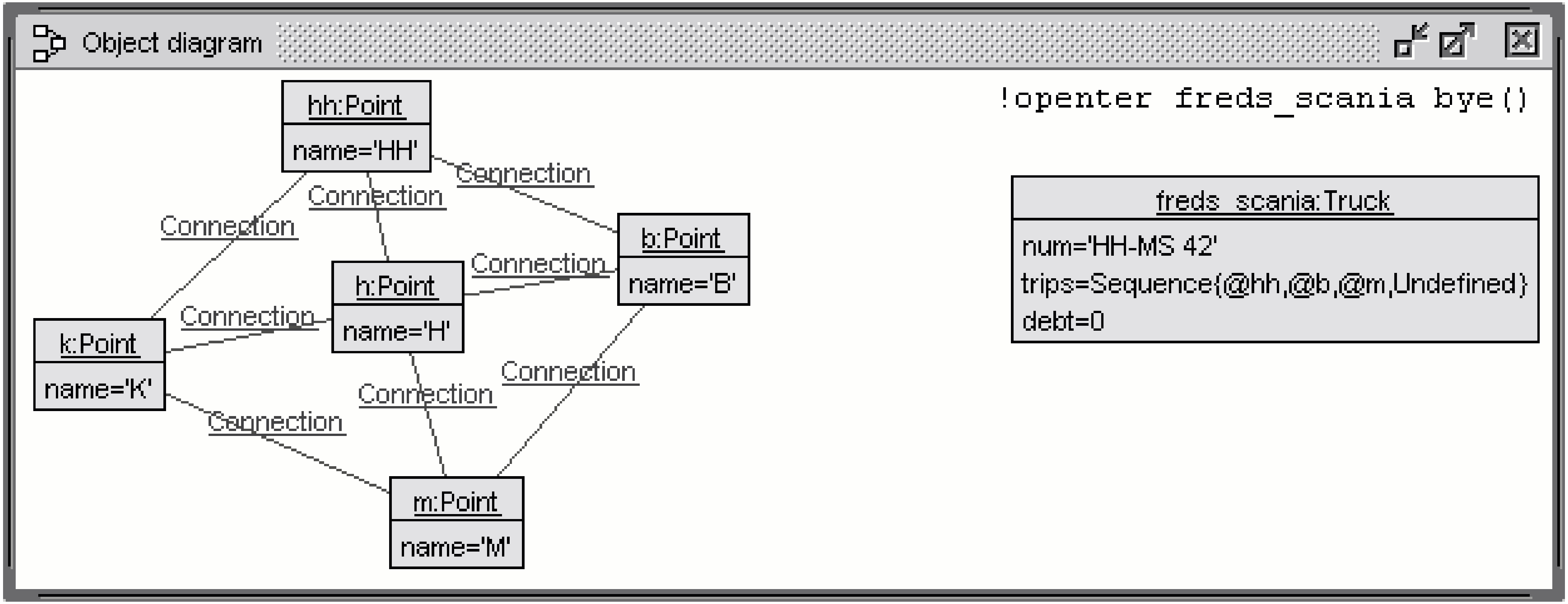


```
!openter freds_scania move(m)
```



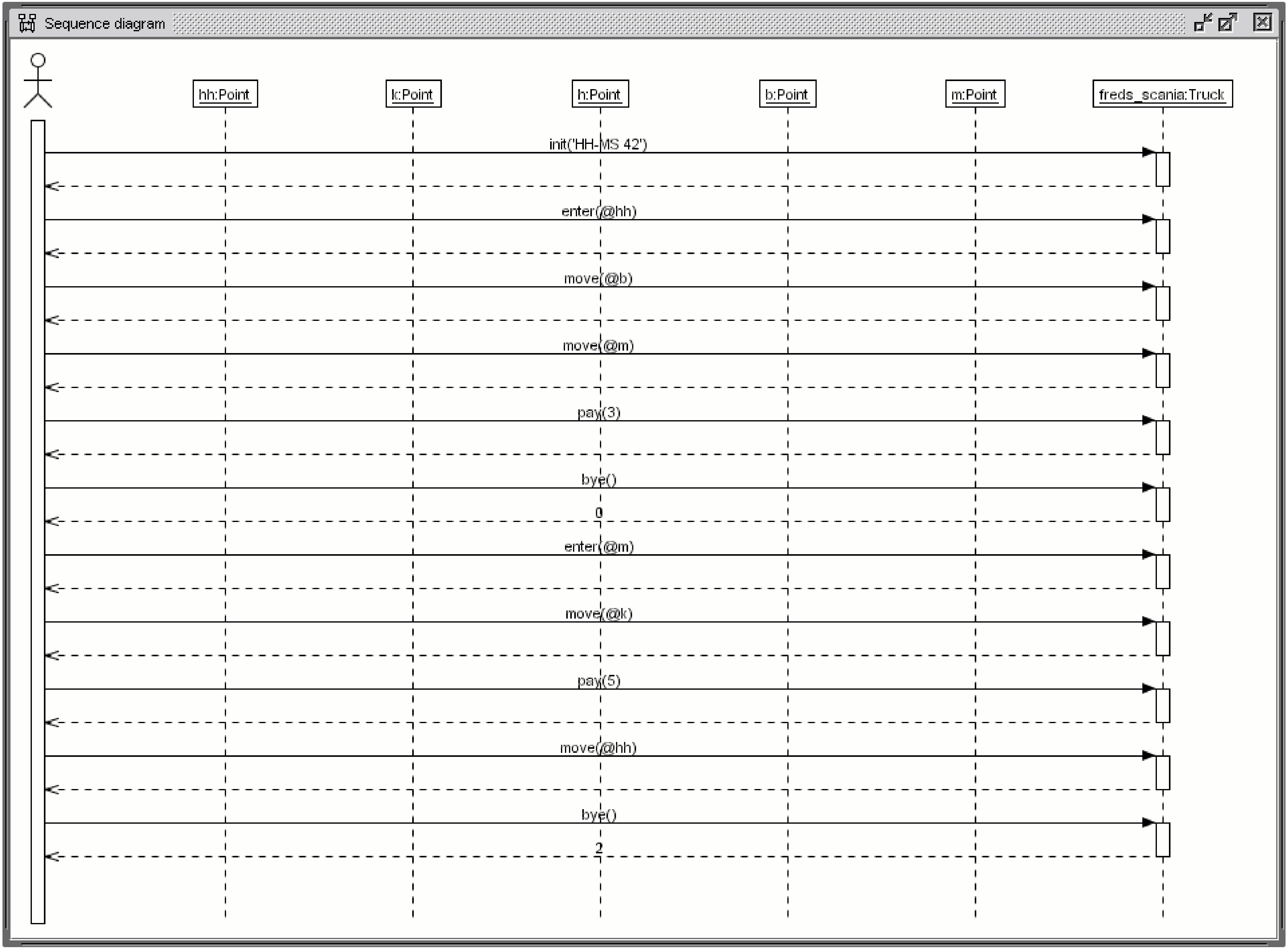
```
!openter freds_scania pay(3)
```

num='HH-MS 42'
trips=Sequence{@hh,@b}
debt=0



```
!openter freds_scania bye()
```

num='HH-MS 42'
trips=Sequence{@hh,@b,@m,Undefined}
debt=0


Sequence diagram
hh:Point
k:Point
h:Point
b:Point
m:Point
freds_scania:Truck
init('HH-MS 42')
enter(@hh)
move(@b)
move(@m)
pay(3)
bye()
0
enter(@m)
move(@k)
pay(5)
move(@hh)
bye()
2

```
Invariants (Overview)
----------

context Truck inv numIsKey  - num is key attribute for Truck
context Point inv nameIsKey - name is key attribute for Point
context Point inv noCycles  - no cycles in the Point graph

Pre- and postconditions for Point operations
--------------------------------------------

context Point::init(aName:String)
pre  freshPoint   - Point is unused
post nameAssigned - attribute name was assigned
post allPointInvs - all Point invariants hold

context Point::northConnect(aNorth:Point)
pre  aNorthDefined      - param aNorth not undefined
pre  freshConnection    - intended link not in self's north/south links
pre  notSelfLink        - intended link not a self loop
pre  insertionOk        - intended link induces not a cycle
post connectionAssigned - intended link existing
post allPointInvs       - all Point invariants hold

context Point::southConnect(aSouth:Point)
- analogously to northConnect
```

```
Pre- and postconditions for Truck operations -A-
--------------------------------------------

context Truck::init(aNum:String)
pre  freshTruck          - Truck is unused
post numTripsDebtAssigned - Truck attributes were assigned
post allTruckInvs        - all Truck invariants hold

context Truck::enter(entry:Point)
pre  tripsOk         - trips empty or last is undefined
pre  currentEmpty    - not currently connected to a Point
post debtAssigned    - attribute debt initialized
post currentAssigned - currently connected to entry
post allTruckInvs    - all Truck invariants hold

context Truck::move(target:Point)
pre  currentExists   - currently connected to a Point
pre  targetReachable - target connected to current Point
post currentAssigned - currently connected to target
post allTruckInvs    - all Truck invariants hold
```

```
Pre- and postconditions for Truck operations -B-
--------------------------------------------

context Truck::pay(amount:Integer)
pre  amountPositive - amount is positive
pre  currentExists  - currently connected to a Point
post debtReduced    - debt was reduced by amount
post allTruckInvs   - all Truck invariants hold

context Truck::bye():Integer
pre  currentExists           - currently connected to a Point
pre  noDebt                  - no debt due
post returnEqualsOverPayment - overpayment is returned to Truck
post currentEmpty            - not currently connected to a Point
post allTruckInvs            - all Truck invariants hold
```

```
------------------------------------------------------------- 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=oclUndefined(String) and
  self.north->isEmpty and self.south->isEmpty
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:
  not(self=aNorth)
pre insertionOk:
  not(aNorth.northPlus()->includes(self))
post connectionAssigned:
  self.north->includes(aNorth)
post allPointInvs:
  nameIsKey() and noCycles()
---------------------------------------------------- Point::southConnect
context Point::southConnect(aSouth:Point)
  ...
```

```
------------------------------------------------------------ Truck::init
context Truck::init(aNum:String)
pre freshTruck:
  self.num=oclUndefined(String) and
  self.trips=oclUndefined(Sequence(Point)) and
  self.debt=oclUndefined(Integer)
post numTripsDebtAssigned:
  aNum=self.num and
  oclEmpty(Sequence(Point))=self.trips and
  0=self.debt
post allTruckInvs:
  numIsKey()
----------------------------------------------------------- Truck::enter
context Truck::enter(entry:Point)
pre tripsOk:
  self.trips=oclEmpty(Sequence(Point)) or
  self.trips->last=oclUndefined(Point)
pre currentEmpty:
  self.point->isEmpty
post debtAssigned:
  1=self.debt
post currentAssigned:
  entry=self.point
post allTruckInvs:
  numIsKey()
```

```
------------------------------------------------------------- Truck::move
context Truck::move(target:Point)
pre currentExists:
  self.point->size=1
pre targetReachable:
  self.point.north->union(self.point.south)->includes(target)
post currentAssigned:
  target=self.point
post allTruckInvs:
  numIsKey()
------------------------------------------------------------- Truck::pay
context Truck::pay(amount:Integer)
pre amountPositive:
  amount>0
pre currentExists:
  self.point->size=1
post debtReduced:
  (self.debt@pre-amount)=(self.debt)
post allTruckInvs:
  numIsKey()
```

```
------------------------------------------------------------- Truck::bye
context Truck::bye():Integer
pre currentExists:
  self.point->size=1
pre noDebt:
  self.debt<=0
post returnEqualsOverPayment:
  self.debt.abs=result
post currentEmpty:
  self.point->isEmpty
post allTruckInvs:
  numIsKey()
```

```
Command sequences -A-
-----------------

-- Point::init(aName:String)
!set self.name:=aName

-- Point::northConnect(aNorth:Point)
!insert (aNorth,self) into Connection

-- Point::southConnect(aSouth:Point)
!insert (self,aSouth) into Connection
```

```
Command sequences -B-
-----------------

-- Truck::init(aNum:String)
!set self.num:=aNum
!set self.trips:=oclEmpty(Sequence(Point))
!set self.debt:=0

-- Truck::enter(entry:Point)
!insert (self,entry) into Current
!set self.debt:=1

-- Truck::move(target:Point)
!set self.trips:=self.trips->including(self.point)
!set self.debt:=self.debt+1
!delete (self,self.point) from Current
!insert (self,target) into Current

-- Truck::pay(amount:Integer)
!set self.debt:=self.debt-amount

-- Truck::bye():Integer
!set self.trips:=self.trips->including(self.point)
!set self.trips:=self.trips->including(oclUndefined(Point))
!delete (self,self.point) from Current
```

```
Protocol -A-
--------

use> !create hh:Point
use> !openter hh init('HH')
     precondition `freshPoint' is true
use> read Point_init.cmd
     Point_init.cmd> -- Point::init(aName:String)
     Point_init.cmd> !set self.name:=aName
use> !opexit
     postcondition `nameAssigned' is true
     postcondition `allPointInvs' is true


use> !openter hh southConnect(k)
     precondition `aSouthDefined' is true
     precondition `freshConnection' is true
     precondition `notSelfLink' is true
     precondition `insertionOk' is true
use> read Point_southConnect.cmd
     Point_southConnect.cmd> -- Point::southConnect(aSouth:Point)
     Point_southConnect.cmd> !insert (self,aSouth) into Connection
use> !opexit
     postcondition `connectionAssigned' is true
     postcondition `allPointInvs' is true
```

```
Protocol -B-
--------

use> !create freds_scania:Truck
use> !openter freds_scania init('HH-MS 42')
     precondition `freshTruck' is true
use> read Truck_init.cmd
     Truck_init.cmd> -- Truck::init(aNum:String)
     Truck_init.cmd> !set self.num:=aNum
     Truck_init.cmd> !set self.trips:=oclEmpty(Sequence(Point))
     Truck_init.cmd> !set self.debt:=0
use> !opexit
     postcondition `numTripsDebtAssigned' is true
     postcondition `allTruckInvs' is true


use> !openter freds_scania enter(hh)
     precondition `tripsOk' is true
     precondition `currentEmpty' is true
use> read Truck_enter.cmd
     Truck_enter.cmd> -- Truck::enter(entry:Point)
     Truck_enter.cmd> !insert (self,entry) into Current
     Truck_enter.cmd> !set self.debt:=1
use> !opexit
     postcondition `debtAssigned' is true
     postcondition `currentAssigned' is true
     postcondition `allTruckInvs' is true
```

```
Protocol -C-
--------

use> !openter freds_scania move(b)
     precondition `currentExists' is true
     precondition `targetReachable' is true
use> read Truck_move.cmd
     Truck_move.cmd> -- Truck::move(target:Point)
     Truck_move.cmd> !set self.trips:=self.trips->including(self.point)
     Truck_move.cmd> !set self.debt:=self.debt+1
     Truck_move.cmd> !delete (self,self.point) from Current
     Truck_move.cmd> !insert (self,target) into Current
use> !opexit
     postcondition `currentAssigned' is true
     postcondition `allTruckInvs' is true


use> !openter freds_scania pay(3)
     precondition `amountPositive' is true
     precondition `currentExists' is true
use> read Truck_pay.cmd
     Truck_pay.cmd> -- Truck::pay(amount:Integer)
     Truck_pay.cmd> !set self.debt:=self.debt-amount
use> !opexit
     postcondition `debtReduced' is true
     postcondition `allTruckInvs' is true
```

```
Protocol -D-
--------

use> !openter freds_scania bye()
     precondition `currentExists' is true
     precondition `noDebt' is true
use> read Truck_bye.cmd
     Truck_bye.cmd> -- Truck::bye():Integer
     Truck_bye.cmd> !set self.trips:=self.trips->including(self.point)
     Truck_bye.cmd> !set self.trips:=
     Truck_bye.cmd>   self.trips->including(oclUndefined(Point))
     Truck_bye.cmd> !delete (self,self.point) from Current
use> !opexit self.debt.abs
     postcondition `returnEqualsOverPayment' is true
     postcondition `currentEmpty' is true
     postcondition `allTruckInvs' is true
```

Concluding remarks

- Case study of a small application employed UML diagrams: Class, statechart, object, sequence diagrams

- Intensive use of OCL for structural and behavioral facets

- Comprehensive modeling (invariants and pre- and postconditions) and Implementation (command sequences)

- Implementation meets modeling

- Invariants and pre- and postcondition are checked during validation

- Intensive use of associations for static data structure description (Connection) and for dynamic object properties (Current)

- Invariants (also) realized by operation postconditions through corresponding operation calls