# Consistency, Independence and Consequences in UML and OCL Models

**Martin Gogolla, Mirco Kuhlmann, Lars Hamann**

University of Bremen
Computer Science Department
Database Systems Group
D-28334 Bremen, Germany
{gogolla | mk | lhamann}@informatik.uni-bremen.de

## Context

Improving software quality through model-centric development
in contrast to code-centric development

Supported by modeling languages and standards like
- UML (Unified Modeling Language) including
  OCL (Object Constraint Language)
- QVT (Queries, Views, Transformations)

Our focus is on OCL and UML class diagram features

Tool USE (UML-based Specification Environment)
- validation of UML and OCL models
- by building prototypical test cases
- through scenarios comprising UML object or sequence diagrams

Goal of USE: derive properties of a UML design from these test scenarios

## Consistency, Independence, Consequences

USE supports OCL constraint (invariant) checking
- consistency
- independence
- consequences

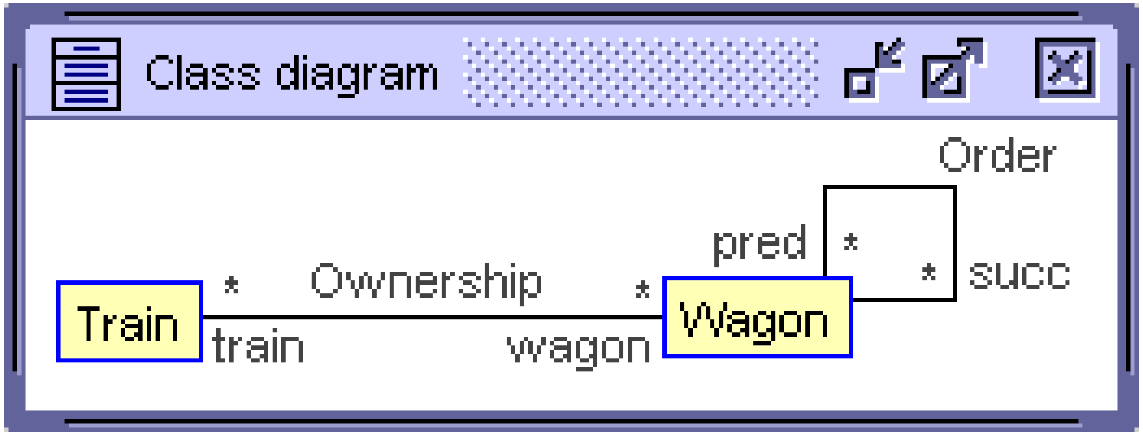
Consistency: constructing a positive test case in form of an object or sequence diagram such that all invariants do hold

Independence: no single invariant can be concluded from other stated invariants; keep UML models small and focussed; construction of counter test cases

Consequences (drawing conclusions): only basic properties formulated as invariants; other more advanced properties can be consequences; checked in USE by building counter test scenarios or by showing that a property is valid in a fixed search space (collection of UML object diagrams)

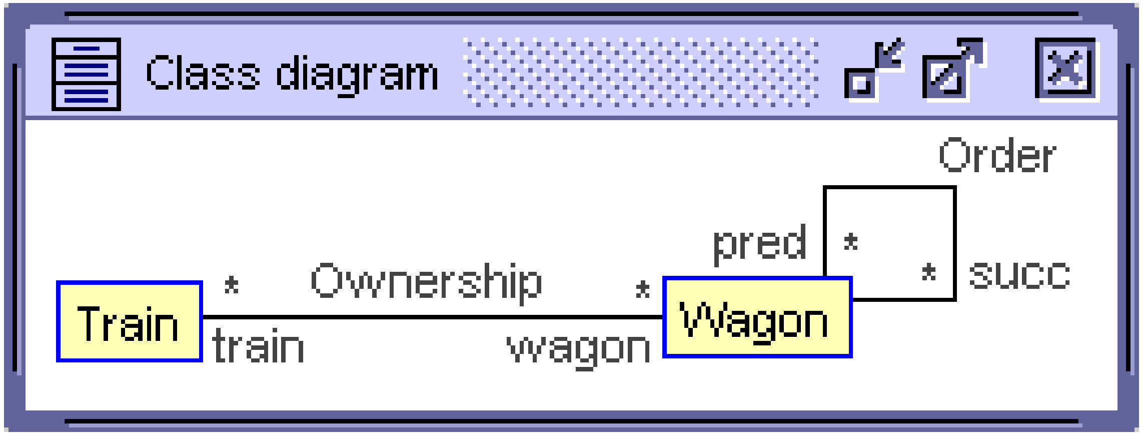
## Example model: Trains, wagons, and their formation - Invariants

```
context Train inv wagon1_n: self.wagon->size>=1
context Wagon inv train1_1: self.train->size=1
context Wagon inv succ0_1: self.succ->size<=1
context Wagon inv pred0_1: self.pred->size<=1

context Train inv oneWell:
 self.wagon->one(well| self.wagon->forAll(w|
  well.succPlus()->includesAll(w.succPlus())))
context Train inv noCycles:
 self.wagon->forAll(w|w.predPlus()->excludes(w))
context w1:Wagon inv trainComm:
 Wagon.allInstances->forAll(w2|
  w1.succ->includes(w2) implies w1.train=w2.train)
```



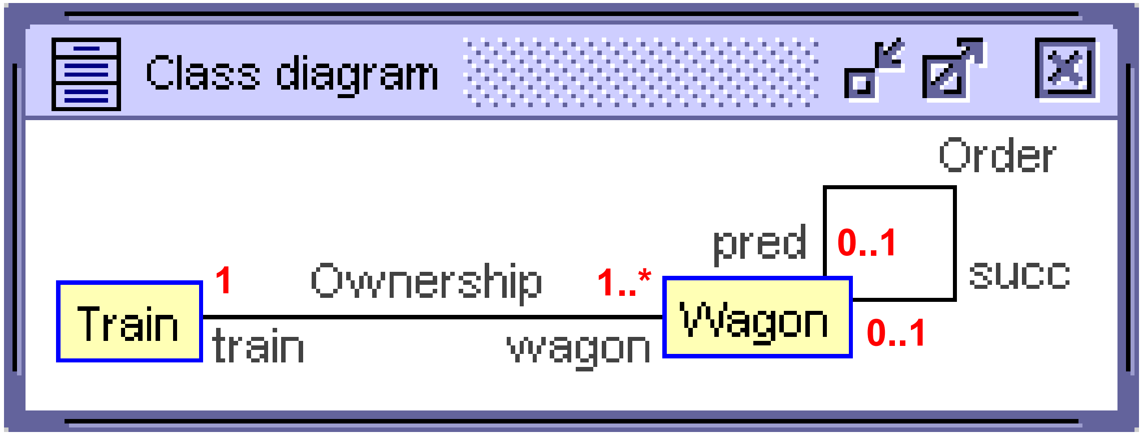
# Example model: Trains, wagons, and their formation - Operations

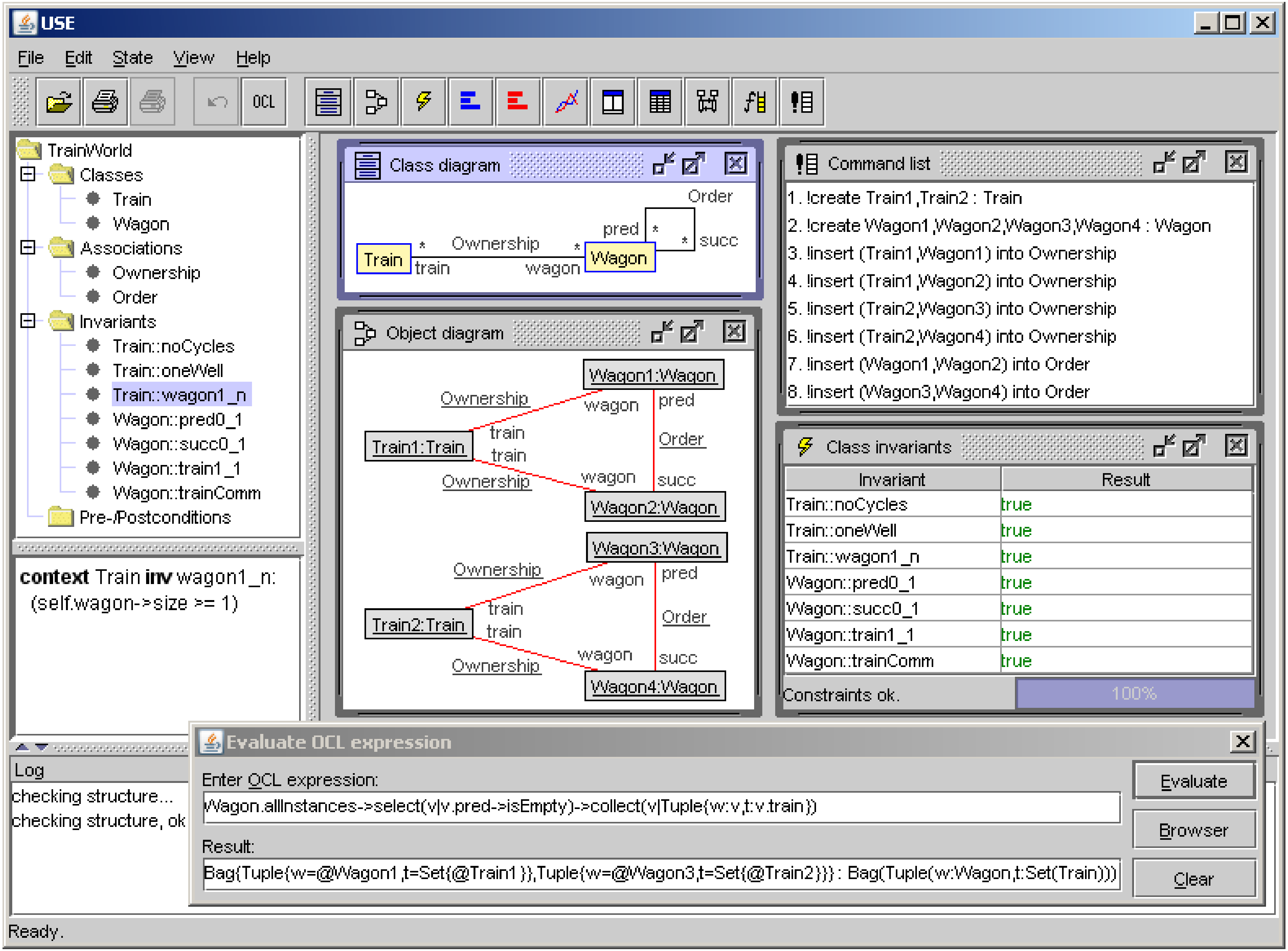
```
Wagon::succPlus():Set(Wagon)=self.succPlusOnSet(self.succ)
Wagon::succPlusOnSet(s:Set(Wagon)):Set(Wagon)=
 let oneStep:Set(Wagon)=s.succ->asSet in
 if s->includesAll(oneStep) then s
 else succPlusOnSet(s->union(oneStep)) endif

Train::allWagons():Set(Wagon)=
 self.wagon->union(self.wagon.predPlus()->asSet())->
  union(self.wagon.succPlus()->asSet())
```



## Example model: Explicit invariants vs. Multiplicities

```
context Train inv wagon1_n: self.wagon->size>=1
context Wagon inv train1_1: self.train->size=1
context Wagon inv succ0_1: self.succ->size<=1
context Wagon inv pred0_1: self.pred->size<=1

context Train inv oneWell:
 self.wagon->one(well| self.wagon->forAll(w|
  well.succPlus()->includesAll(w.succPlus())))
context Train inv noCycles:
 self.wagon->forAll(w|w.predPlus()->excludes(w))
context w1:Wagon inv trainComm:
 Wagon.allInstances->forAll(w2|
  w1.succ->includes(w2) implies w1.train=w2.train)
```





```
context Train inv wagon1_n:
  (self.wagon->size >= 1)
```

```
1. !create Train1,Train2 : Train
2. !create Wagon1,Wagon2,Wagon3,Wagon4 : Wagon
3. !insert (Train1,Wagon1) into Ownership
4. !insert (Train1,Wagon2) into Ownership
5. !insert (Train2,Wagon3) into Ownership
6. !insert (Train2,Wagon4) into Ownership
7. !insert (Wagon1,Wagon2) into Order
8. !insert (Wagon3,Wagon4) into Order
```

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

Constraints ok. 100%

```
checking structure...
checking structure, ok
```

Enter OCL expression:

```
Wagon.allInstances->select(v|v.pred->isEmpty)->collect(v|Tuple{w:v,t:v.train})
```

Result:

```
Bag{Tuple{w=@Wagon1,t=Set{@Train1}},Tuple{w=@Wagon3,t=Set{@Train2}}} : Bag(Tuple(w:Wagon,t:Set(Train)))
```

Ready.

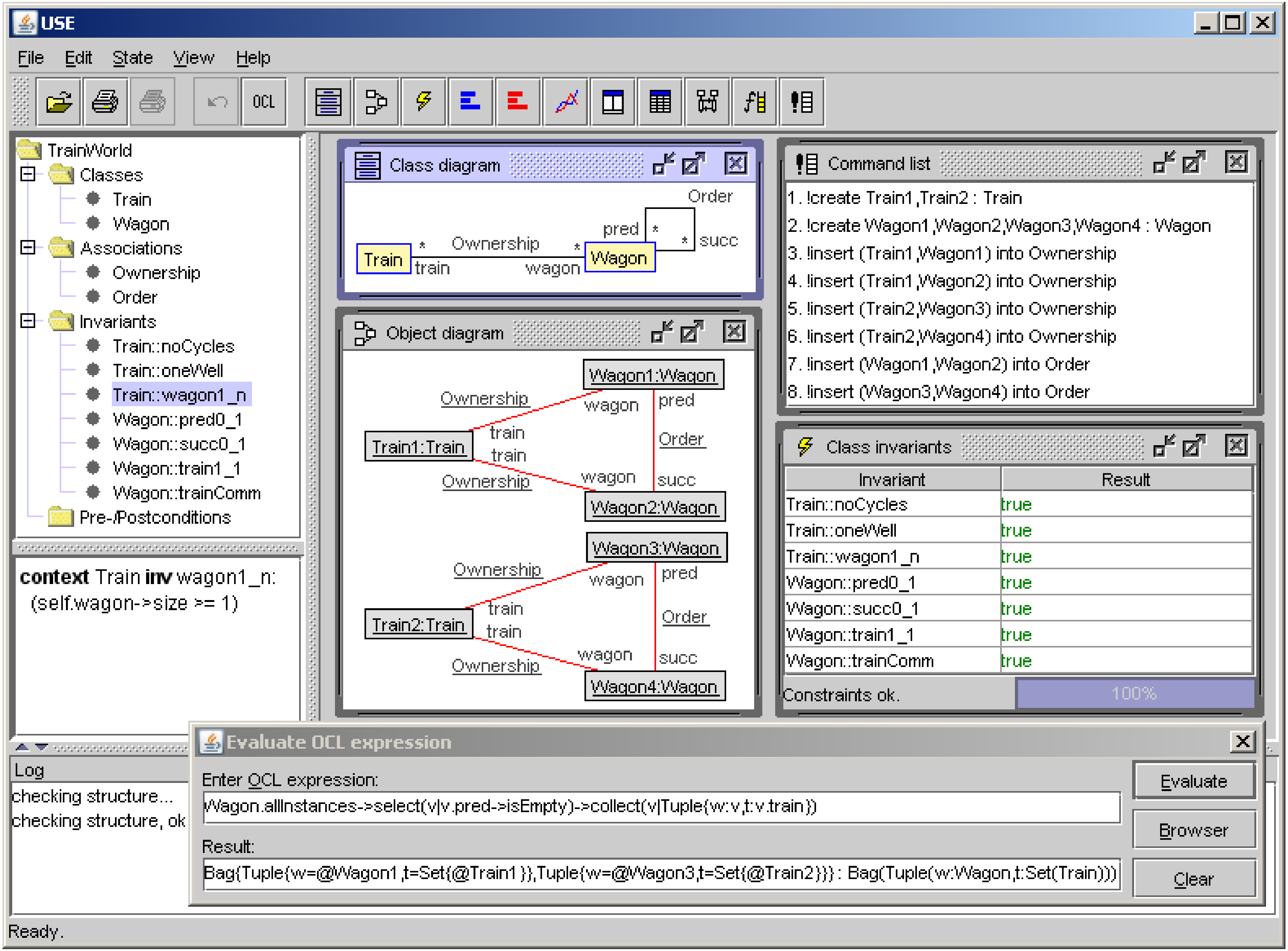
**Interaction with USE**

- Graphical User Interface (GUI)
- Command Line Interface (CLI)

**Views in USE**

- Project browser overview
- Project browser detail
- Class diagram
- Object diagram
- Log view for model-inherent constraints
- Command list
- Class invariant evaluation
- OCL expression evaluation

- Sequence diagram
- Object properties
- Class extent
- Evaluation browser



```
1. !create Train1,Train2 : Train
2. !create Wagon1,Wagon2,Wagon3,Wagon4 : Wagon
3. !insert (Train1,Wagon1) into Ownership
4. !insert (Train1,Wagon2) into Ownership
5. !insert (Train2,Wagon3) into Ownership
6. !insert (Train2,Wagon4) into Ownership
7. !insert (Wagon1,Wagon2) into Order
8. !insert (Wagon3,Wagon4) into Order
```

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

Constraints ok. 100%

```
context Train inv wagon1_n:
  (self.wagon->size >= 1)
```

```
checking structure...
checking structure, ok
```

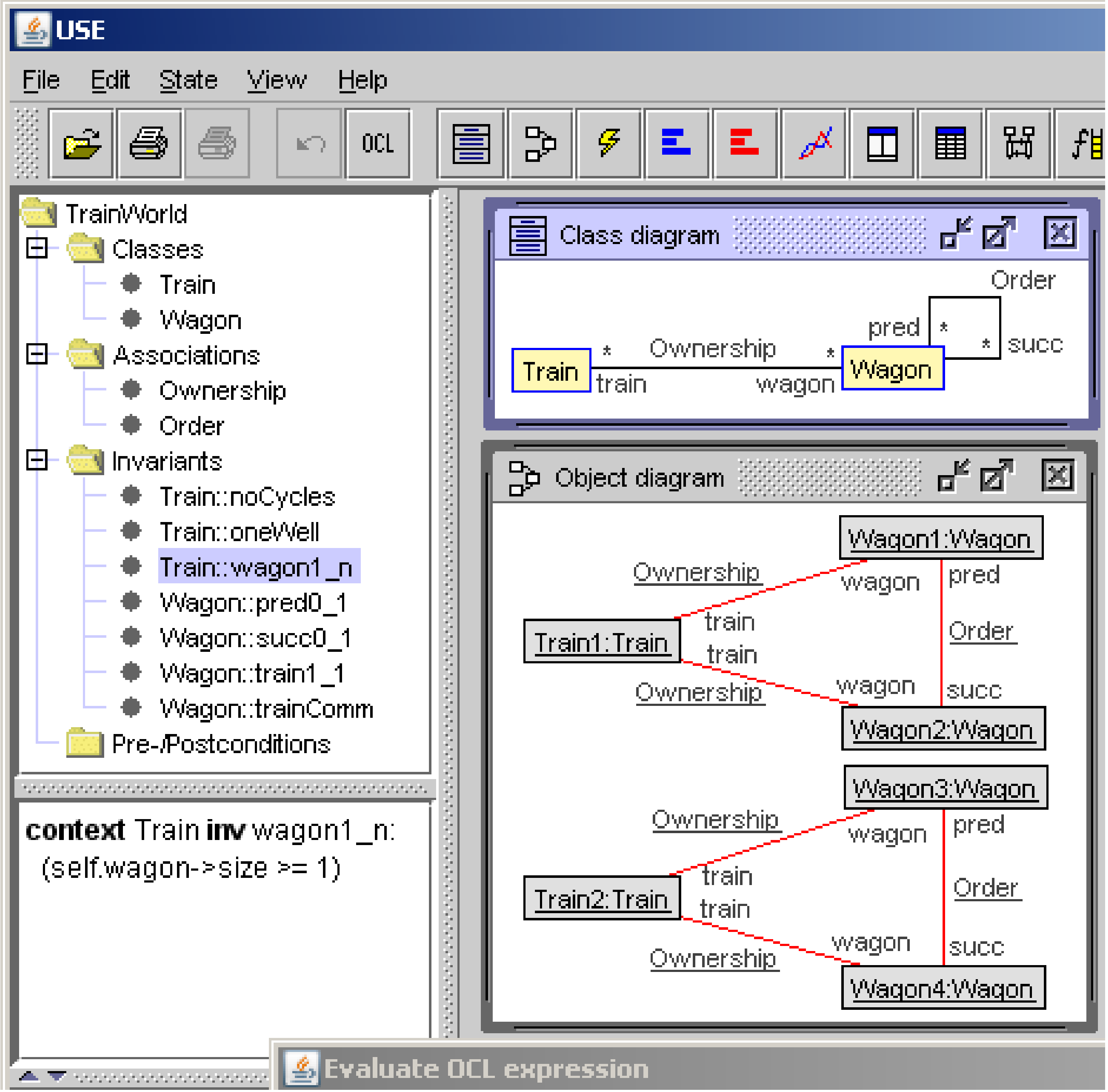
Enter OCL expression:

```
Wagon.allInstances->select(v|v.pred->isEmpty)->collect(v|Tuple{w:v,t:v.train})
```

Result:

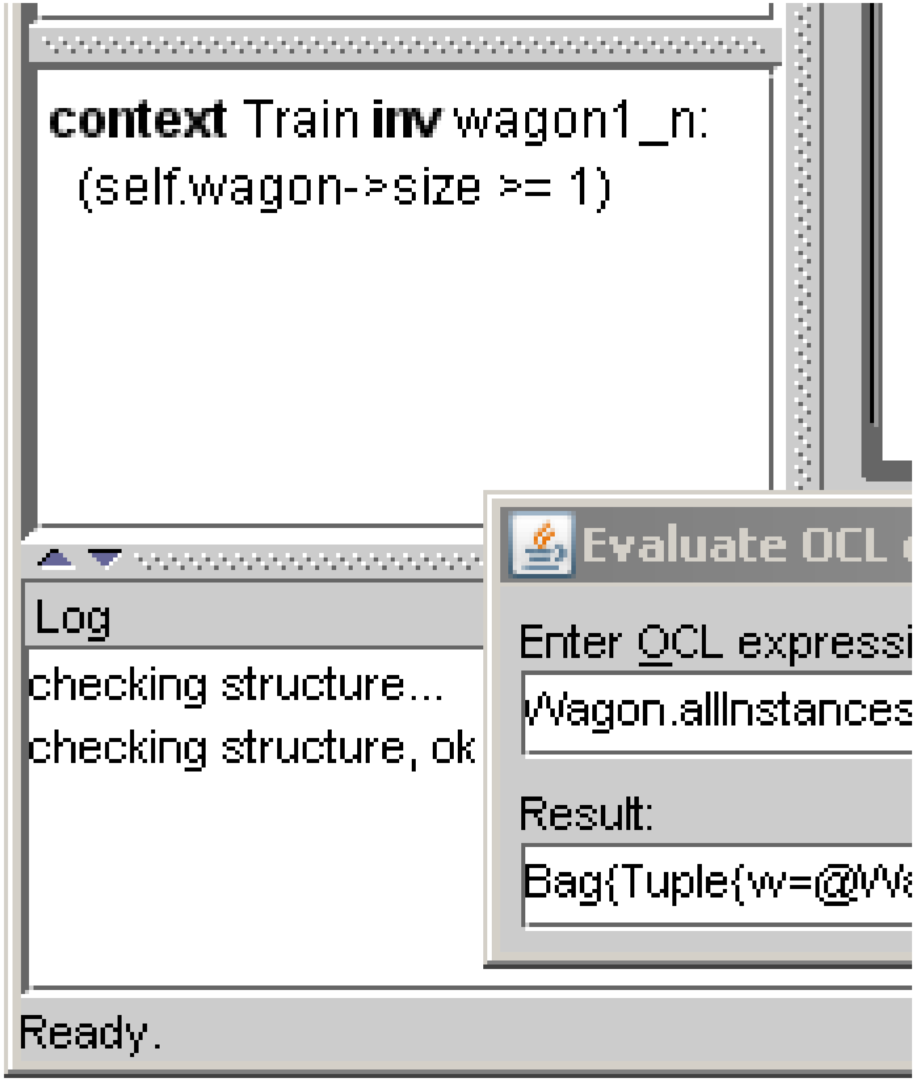
```
Bag{Tuple{w=@Wagon1,t=Set{@Train1}},Tuple{w=@Wagon3,t=Set{@Train2}}} : Bag(Tuple(w:Wagon,t:Set(Train)))
```

Ready.



TrainWorld
- Classes
  - Train
  - Wagon
- Associations
  - Ownership
  - Order
- Invariants
  - Train::noCycles
  - Train::oneWell
  - Train::wagon1_n
  - Wagon::pred0_1
  - Wagon::succ0_1
  - Wagon::train1_1
  - Wagon::trainComm
- Pre-/Postconditions

```
context Train inv wagon1_n:
  (self.wagon->size >= 1)
```



```
context Train inv wagon1_n:
  (self.wagon->size >= 1)
```

Log

```
checking structure...
checking structure, ok
```

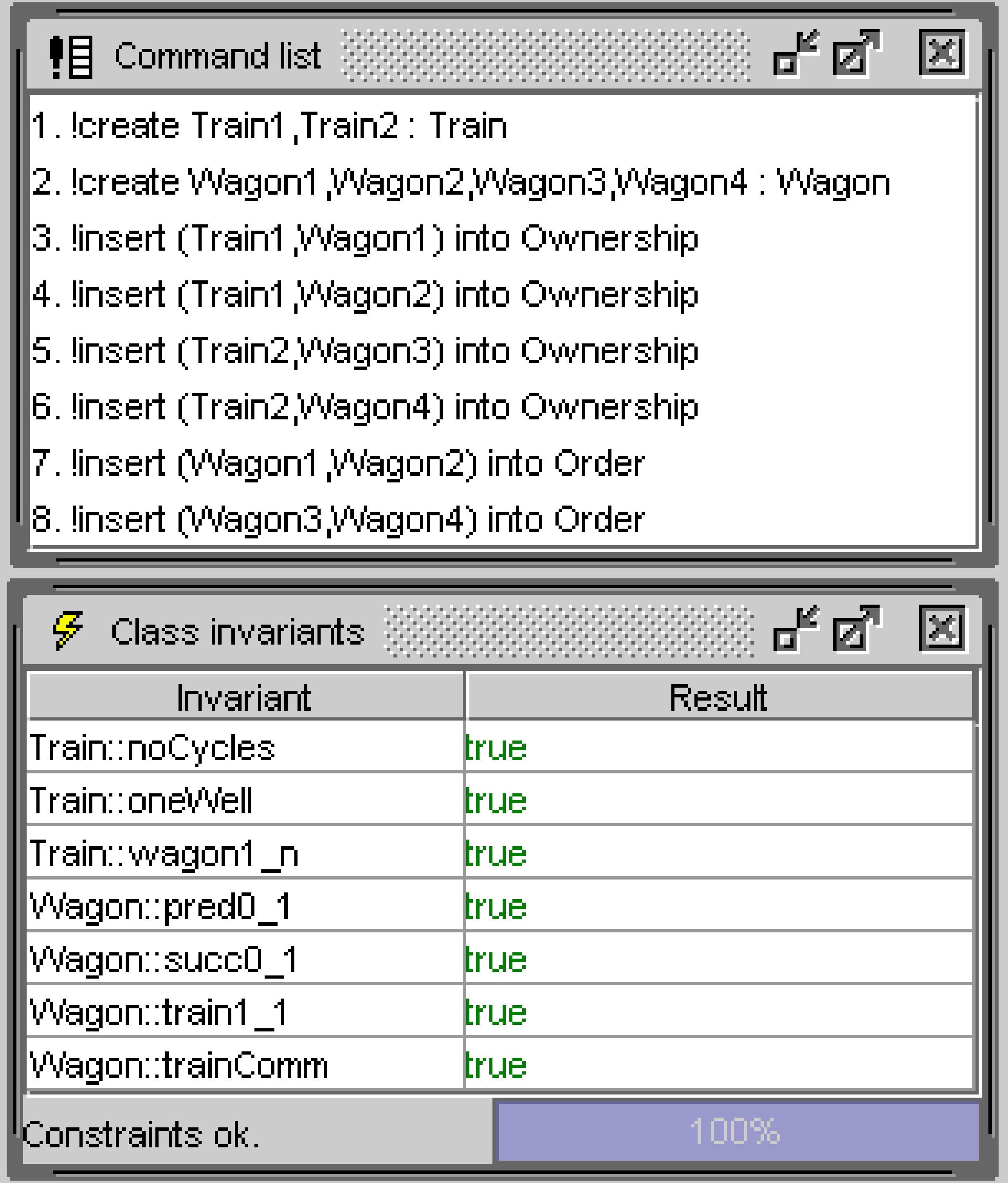
Ready.

Enter OCL expressi

Wagon.allInstances

Result:

Bag{Tuple{w=@Wa

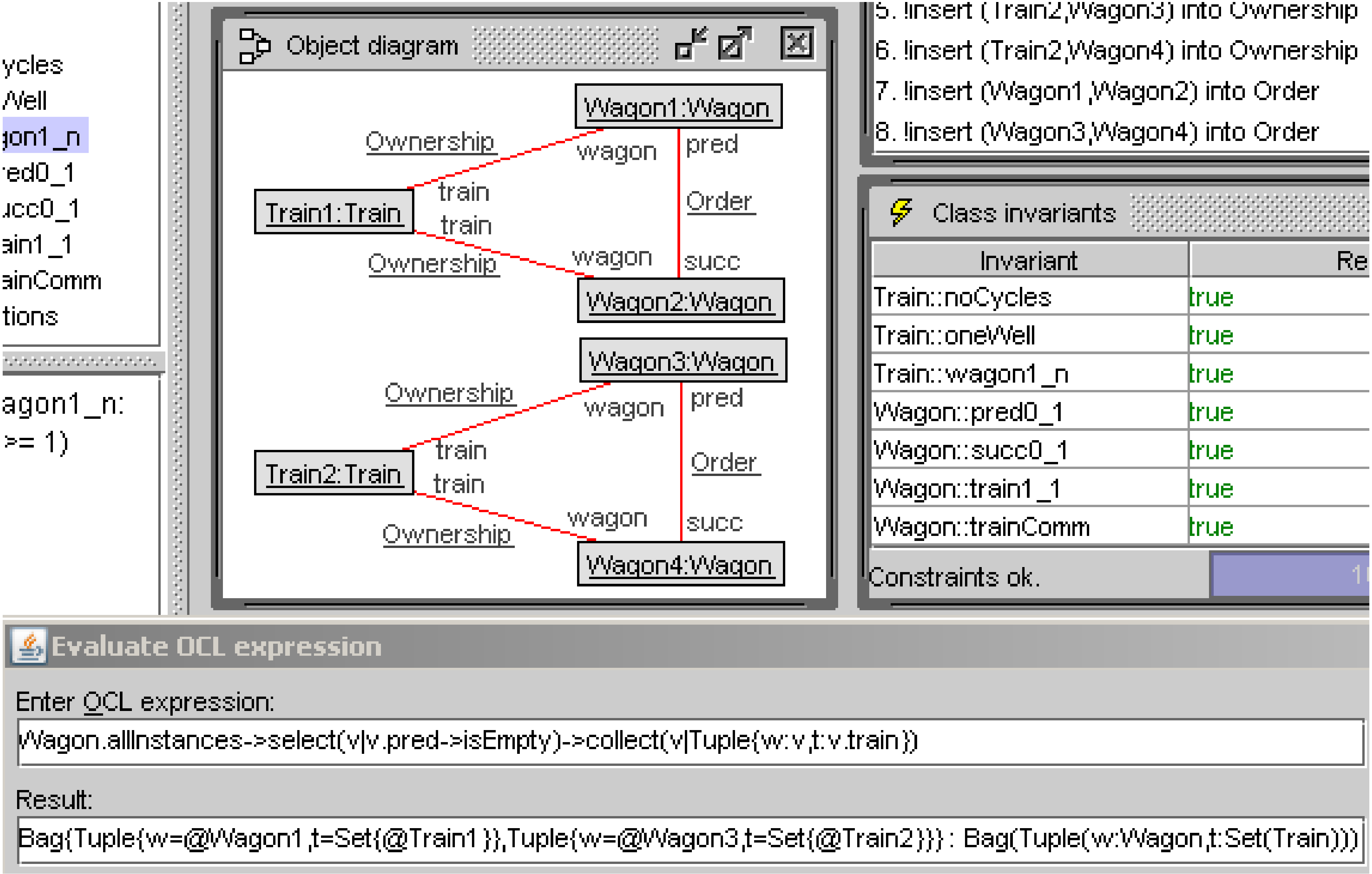


```
1. !create Train1 ,Train2 : Train
2. !create Wagon1 ,Wagon2,Wagon3,Wagon4 : Wagon
3. !insert (Train1 ,Wagon1 ) into Ownership
4. !insert (Train1 ,Wagon2) into Ownership
5. !insert (Train2,Wagon3) into Ownership
6. !insert (Train2,Wagon4) into Ownership
7. !insert (Wagon1 ,Wagon2) into Order
8. !insert (Wagon3,Wagon4) into Order
```

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

Constraints ok.

100%



```
5. !insert (Train2,Wagon3) into Ownership
6. !insert (Train2,Wagon4) into Ownership
7. !insert (Wagon1,Wagon2) into Order
8. !insert (Wagon3,Wagon4) into Order
```

| Invariant | Re |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

Constraints ok.

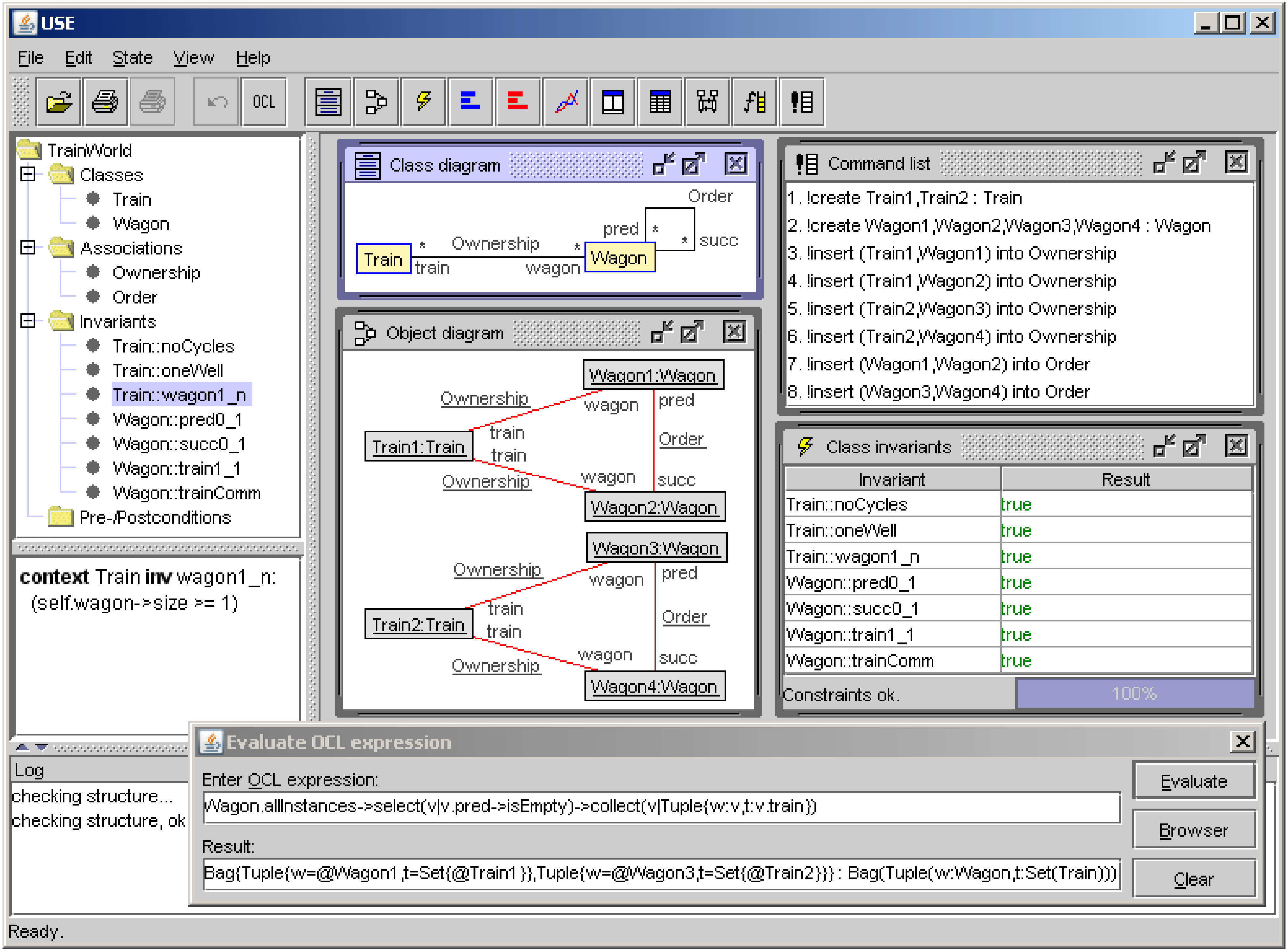
Evaluate OCL expression

Enter OCL expression:

```
Wagon.allInstances->select(v|v.pred->isEmpty)->collect(v|Tuple{w:v,t:v.train})
```

Result:

```
Bag{Tuple{w=@Wagon1,t=Set{@Train1}},Tuple{w=@Wagon3,t=Set{@Train2}}} : Bag(Tuple(w:Wagon,t:Set(Train)))
```



TrainWorld
- Classes
  - Train
  - Wagon
- Associations
  - Ownership
  - Order
- Invariants
  - Train::noCycles
  - Train::oneWell
  - Train::wagon1_n
  - Wagon::pred0_1
  - Wagon::succ0_1
  - Wagon::train1_1
  - Wagon::trainComm
- Pre-/Postconditions

```
context Train inv wagon1_n:
  (self.wagon->size >= 1)
```

```
1. !create Train1,Train2 : Train
2. !create Wagon1,Wagon2,Wagon3,Wagon4 : Wagon
3. !insert (Train1,Wagon1) into Ownership
4. !insert (Train1,Wagon2) into Ownership
5. !insert (Train2,Wagon3) into Ownership
6. !insert (Train2,Wagon4) into Ownership
7. !insert (Wagon1,Wagon2) into Order
8. !insert (Wagon3,Wagon4) into Order
```

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

Constraints ok. 100%

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

Enter OCL expression:

```
Wagon.allInstances->select(v|v.pred->isEmpty)->collect(v|Tuple{w:v,t:v.train})
```

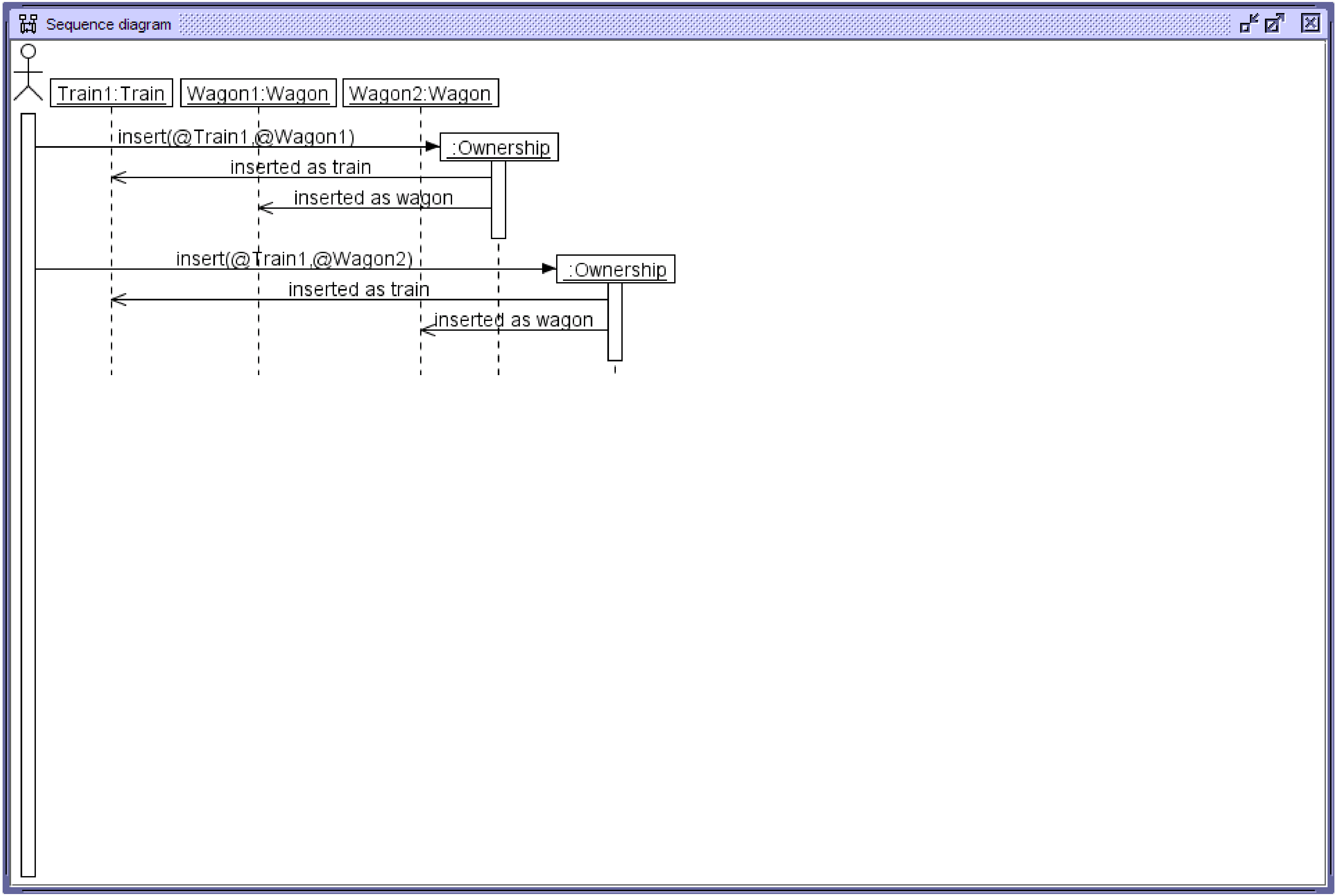
Result:

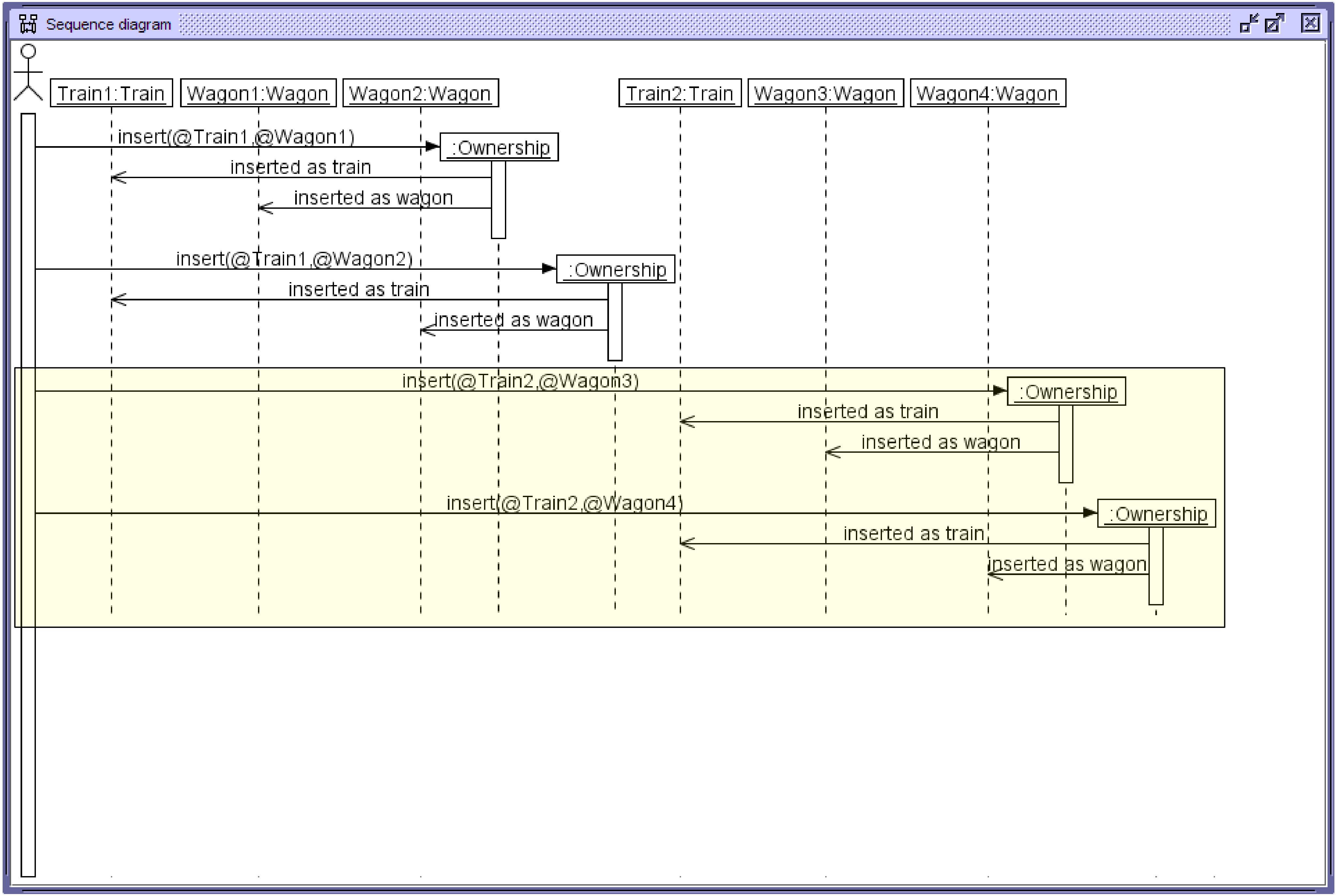
```
Bag{Tuple{w=@Wagon1,t=Set{@Train1}},Tuple{w=@Wagon3,t=Set{@Train2}}} : Bag(Tuple(w:Wagon,t:Set(Train)))
```

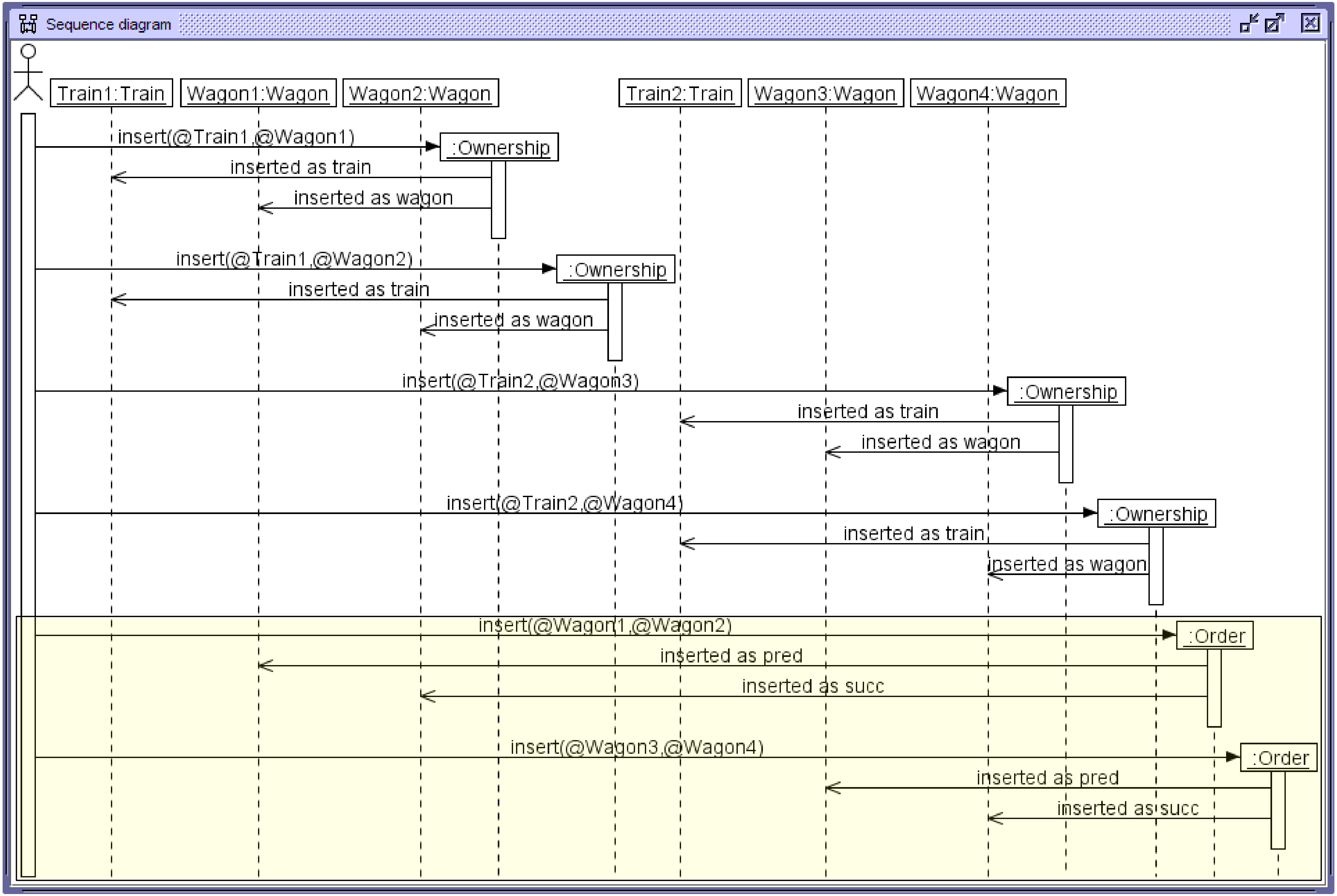
**Interaction with USE**
- Graphical User Interface (GUI)
- Command Line Interface (CLI)

**Views in USE**
- Project browser overview
- Project browser detail
- Class diagram
- Object diagram
- Log view for model-inherent constraints
- Command list
- Class invariant evaluation
- OCL expression evaluation

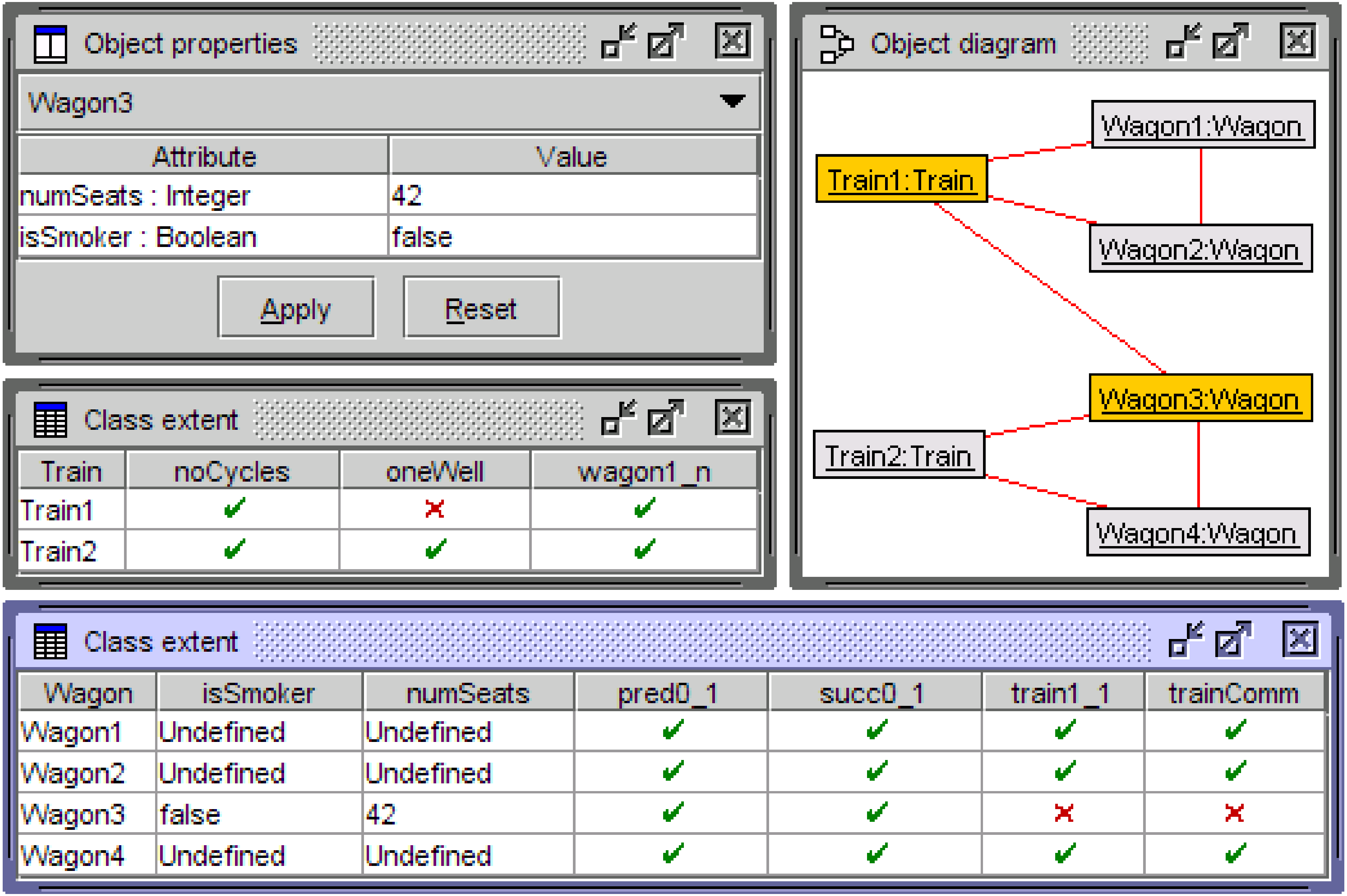
- Sequence diagram
- Object properties
- Class extent
- Evaluation browser


Sequence diagram
Train1:Train
Wagon1:Wagon
Wagon2:Wagon
insert(@Train1,@Wagon1)
:Ownership
inserted as train
inserted as wagon
insert(@Train1,@Wagon2)
:Ownership
inserted as train
inserted as wagon


Sequence diagram
Train1:Train
Wagon1:Wagon
Wagon2:Wagon
Train2:Train
Wagon3:Wagon
Wagon4:Wagon
insert(@Train1,@Wagon1)
:Ownership
inserted as train
inserted as wagon
insert(@Train1,@Wagon2)
:Ownership
inserted as train
inserted as wagon
insert(@Train2,@Wagon3)
:Ownership
inserted as train
inserted as wagon
insert(@Train2,@Wagon4)
:Ownership
inserted as train
inserted as wagon


Sequence diagram
Train1:Train
Wagon1:Wagon
Wagon2:Wagon
Train2:Train
Wagon3:Wagon
Wagon4:Wagon
insert(@Train1,@Wagon1)
:Ownership
inserted as train
inserted as wagon
insert(@Train1,@Wagon2)
:Ownership
inserted as train
inserted as wagon
insert(@Train2,@Wagon3)
:Ownership
inserted as train
inserted as wagon
insert(@Train2,@Wagon4)
:Ownership
inserted as train
inserted as wagon
insert(@Wagon1,@Wagon2)
:Order
inserted as pred
inserted as succ
insert(@Wagon3,@Wagon4)
:Order
inserted as pred
inserted as succ

```
use> !insert (Train1,Wagon3) into Ownership
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Train::noCycles': OK.
     checking invariant (2) `Train::oneWell': FAILED.
       -> false : Boolean
     checking invariant (3) `Train::wagon1_n': OK.
     checking invariant (4) `Wagon::pred0_1': OK.
     checking invariant (5) `Wagon::succ0_1': OK.
     checking invariant (6) `Wagon::train1_1': FAILED.
       -> false : Boolean
     checking invariant (7) `Wagon::trainComm': FAILED.
       -> false : Boolean
     checked 7 invariants in 0.031s, 3 failures.
```

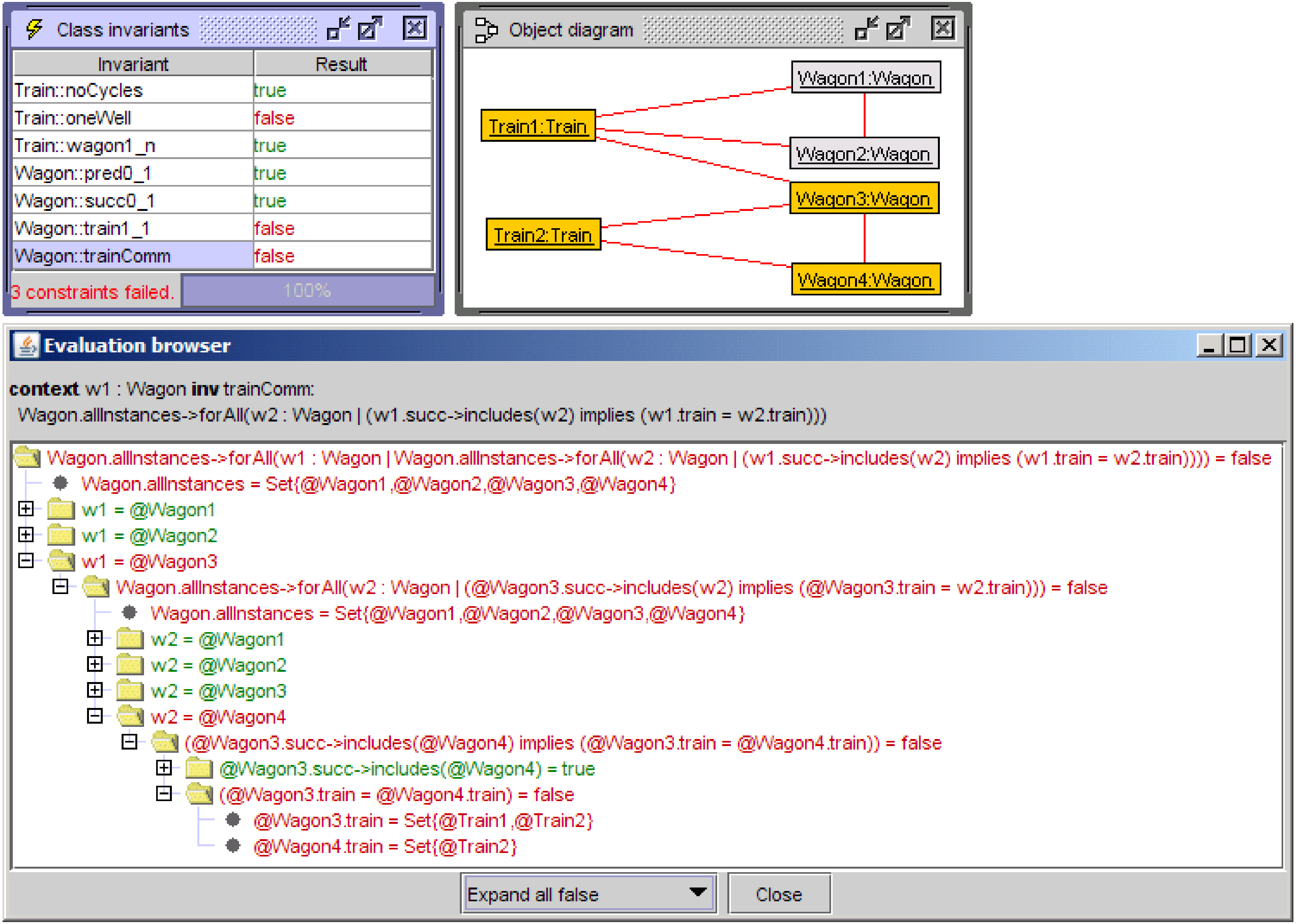


Wagon3

| Attribute | Value |
|---|---|
| numSeats : Integer | 42 |
| isSmoker : Boolean | false |

| Train | noCycles | oneWell | wagon1_n |
|---|---|---|---|
| Train1 | ✓ | ✗ | ✓ |
| Train2 | ✓ | ✓ | ✓ |

| Wagon | isSmoker | numSeats | pred0_1 | succ0_1 | train1_1 | trainComm |
|---|---|---|---|---|---|---|
| Wagon1 | Undefined | Undefined | ✓ | ✓ | ✓ | ✓ |
| Wagon2 | Undefined | Undefined | ✓ | ✓ | ✓ | ✓ |
| Wagon3 | false | 42 | ✓ | ✓ | ✗ | ✗ |
| Wagon4 | Undefined | Undefined | ✓ | ✓ | ✓ | ✓ |



| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | false |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | false |
| Wagon::trainComm | false |

3 constraints failed.

**context** w1 : Wagon **inv** trainComm:
Wagon.allInstances->forAll(w2 : Wagon | (w1.succ->includes(w2) implies (w1.train = w2.train)))

```
Wagon.allInstances->forAll(w1 : Wagon | Wagon.allInstances->forAll(w2 : Wagon | (w1.succ->includes(w2) implies (w1.train = w2.train)))) = false
   ● Wagon.allInstances = Set{@Wagon1,@Wagon2,@Wagon3,@Wagon4}
 ⊞ w1 = @Wagon1
 ⊞ w1 = @Wagon2
 ⊟ w1 = @Wagon3
     ⊟ Wagon.allInstances->forAll(w2 : Wagon | (@Wagon3.succ->includes(w2) implies (@Wagon3.train = w2.train))) = false
          ● Wagon.allInstances = Set{@Wagon1,@Wagon2,@Wagon3,@Wagon4}
        ⊞ w2 = @Wagon1
        ⊞ w2 = @Wagon2
        ⊞ w2 = @Wagon3
        ⊟ w2 = @Wagon4
            ⊟ (@Wagon3.succ->includes(@Wagon4) implies (@Wagon3.train = @Wagon4.train)) = false
                ⊞ @Wagon3.succ->includes(@Wagon4) = true
                ⊟ (@Wagon3.train = @Wagon4.train) = false
                     ● @Wagon3.train = Set{@Train1,@Train2}
                     ● @Wagon4.train = Set{@Train2}
```

Special (efficient) ASSL procedure for building system states

```
procedure genTrainsWagonsOwnershipOrder
 (countTrains:Integer,countWagons:Integer,
  countOwnership:Integer,countOrder:Integer)
var theTrains:Sequence(Train), aTrain:Train,
    theWagons:Sequence(Wagon),
    aWagon:Wagon, aWagon2:Wagon;
begin
theTrains:=CreateN(Train,[countTrains]);
theWagons:=CreateN(Wagon,[countWagons]);
for i:Integer in [Sequence{1..countOwnership}]
  begin
  aTrain:=Try([theTrains]);
  aWagon:=Try([theWagons->reject(w|w.train->includes(aTrain))]);
  Insert(Ownership,[aTrain],[aWagon]);
  end;
for i:Integer in [Sequence{1..countOrder}]
  begin
  aWagon:=Try([theWagons]);
  aWagon2:=Try([theWagons->reject(w|w.pred->includes(aWagon))]);
  Insert(Order,[aWagon],[aWagon2]);
  end;
end;
```

## CONSISTENCY: Using ASSL procedure and directing results with invariants

```
context t1:Train inv trainSizeBalanced: Train.allInstances->forAll(t2|
  t1<>t2 implies (t1.wagon->size-t2.wagon->size).abs<=1)
```

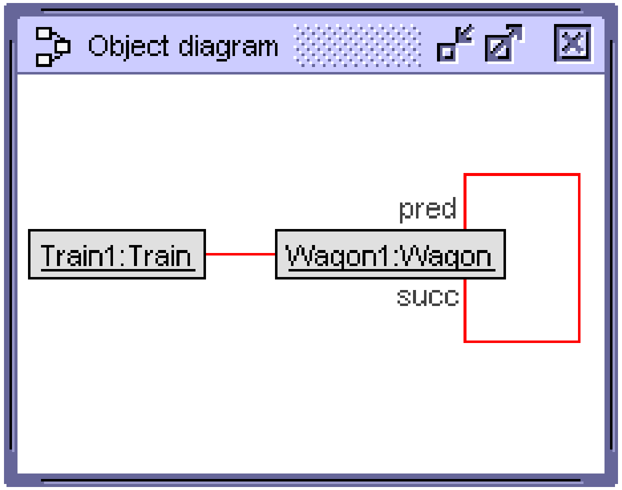
```
use> open train_wagon.use

use> gen load trainSizeBalanced.invs
     Added invariants: Train::trainSizeBalanced

use> gen start train_wagon.assl genTrainsWagonsOwnershipOrder(2,4,4,2)
use> gen result
     Random number generator was initialized with 6315.
     Checked 5786 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train1,Train2 : Train
     !create Wagon1,Wagon2,Wagon3,Wagon4 : Wagon
     !insert (Train1,Wagon1) into Ownership
     !insert (Train1,Wagon2) into Ownership
     !insert (Train2,Wagon3) into Ownership
     !insert (Train2,Wagon4) into Ownership
     !insert (Wagon1,Wagon2) into Order
     !insert (Wagon3,Wagon4) into Order
use> gen result accept
     Generated result (system state) accepted.
```

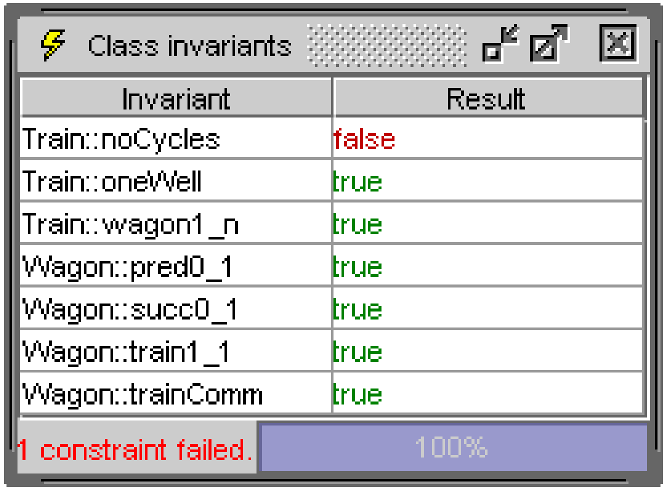
## General (inefficient) ASSL procedure

```
procedure genMaxCountTrainsMaxCountWagons
   (maxCountTrains:Integer,maxCountWagons:Integer)
var theWagons:Sequence(Wagon), theTrains:Sequence(Train),
    actualCountTrains:Integer, actualCountWagons:Integer;
begin actualCountTrains:=Try([Sequence{1..maxCountTrains}]);
actualCountWagons:=Try([Sequence{1..maxCountWagons}]);
theTrains:=CreateN(Train,[actualCountTrains]);
theWagons:=CreateN(Wagon,[actualCountWagons]);
Try(Ownership,[theTrains],[theWagons]);
Try(Order,[theWagons],[theWagons]); end;
```

INDEPENDENCE: Negate invariant whose independence is to be shown

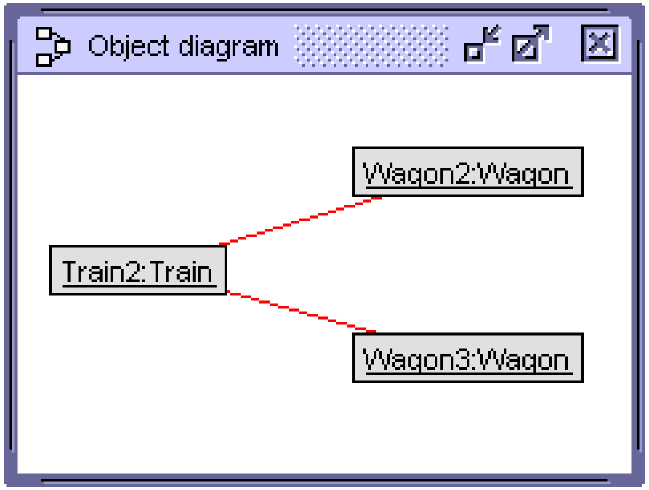
```
use> gen flags Train::noCycles +n
use> gen start train_wagon.assl
genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 9864.
     Checked 4 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train1 : Train
     !create Wagon1 : Wagon
     !insert (Train1,Wagon1) into Ownership
     !insert (Wagon1,Wagon1) into Order
use> gen result accept
     Generated result (system state) accepted.
```

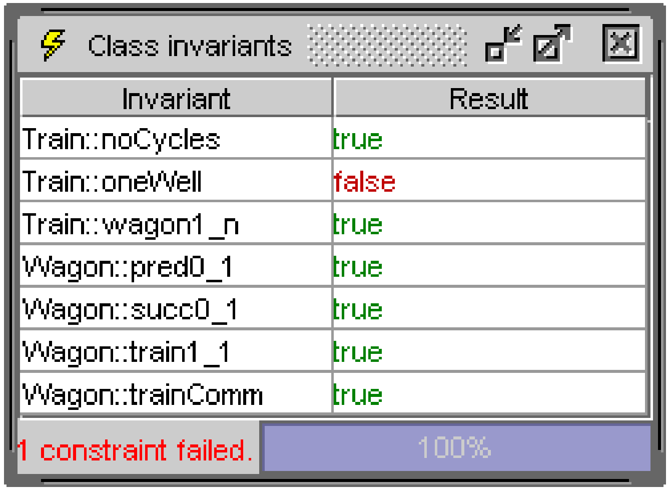




| Invariant | Result |
|---|---|
| Train::noCycles | false |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

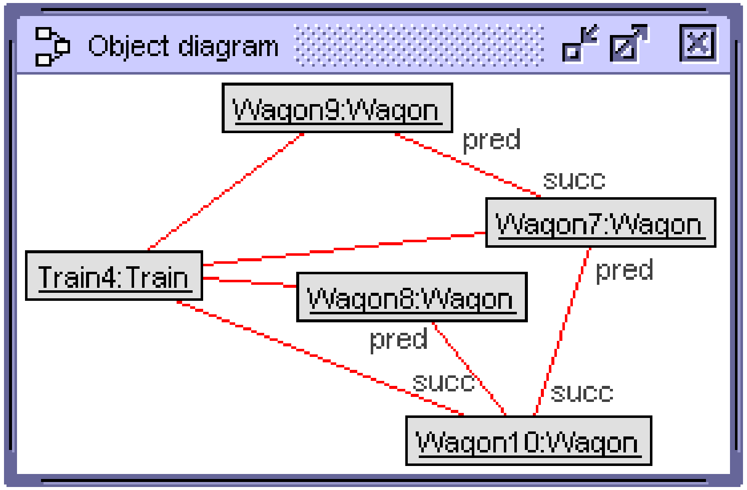
1 constraint failed.

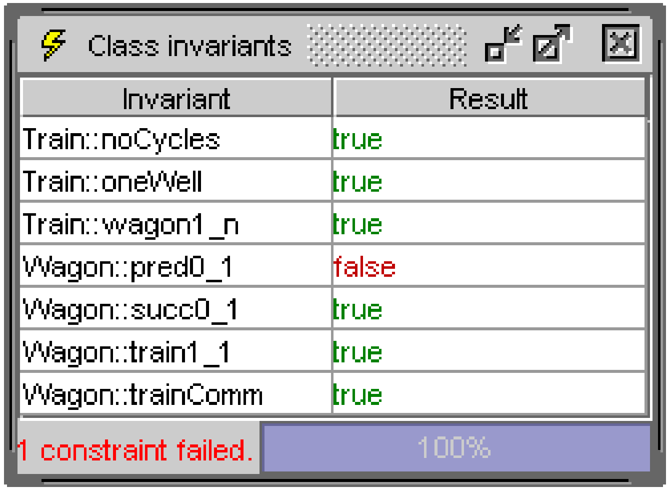




| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | false |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

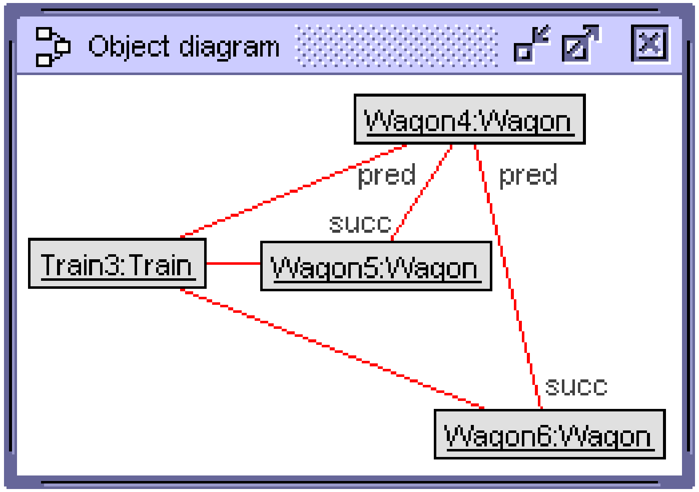
1 constraint failed.

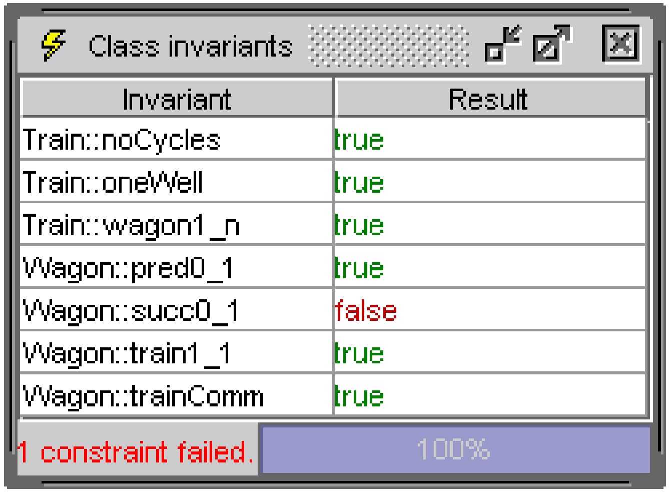




| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | false |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

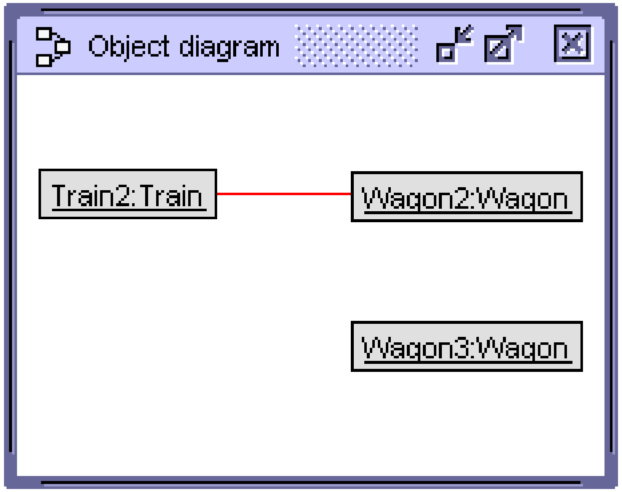
1 constraint failed.

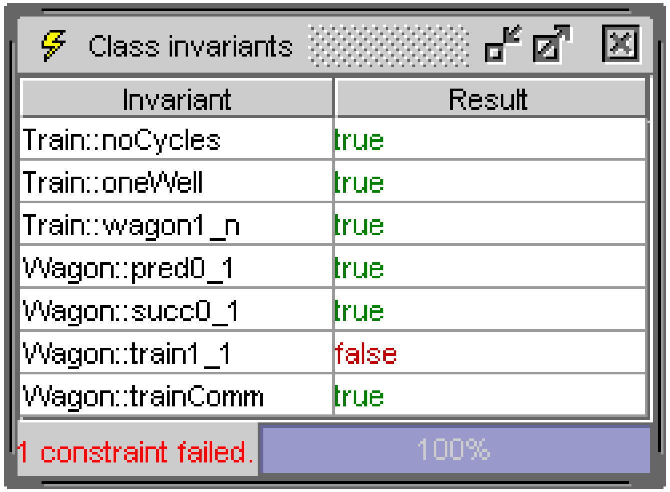



Class invariants

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | false |
| Wagon::train1_1 | true |
| Wagon::trainComm | true |

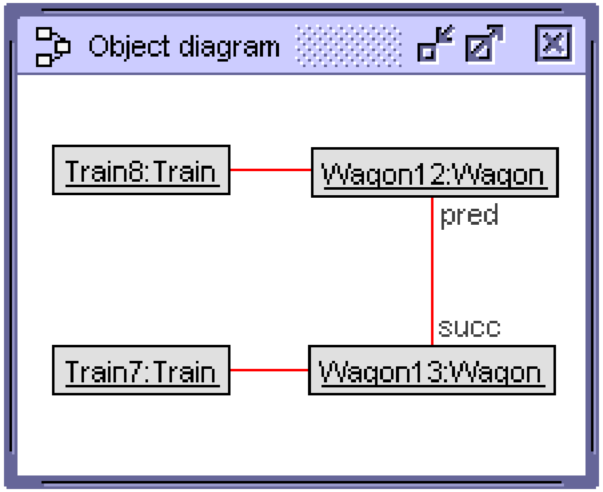
1 constraint failed. 100%

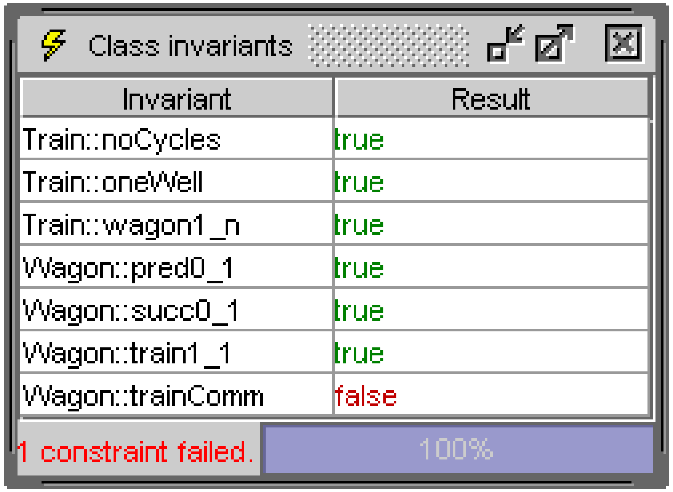



Class invariants

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | false |
| Wagon::trainComm | true |

1 constraint failed. 100%




Class invariants

| Invariant | Result |
|---|---|
| Train::noCycles | true |
| Train::oneWell | true |
| Train::wagon1_n | true |
| Wagon::pred0_1 | true |
| Wagon::succ0_1 | true |
| Wagon::train1_1 | true |
| Wagon::trainComm | false |

1 constraint failed. 100%

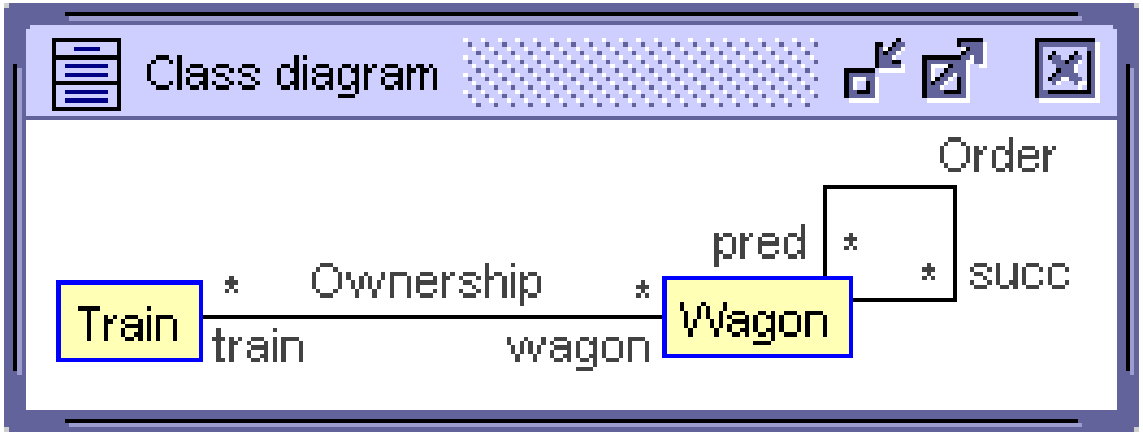
Indication for DEPENDENCE: Counter example not found

```
use> open train_wagon.use

use> gen flags Train::wagon1_n +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 5785.
     Checked 17862988 snapshots.
     Result: No valid state found.
```

Further reasoning shows: wagon1_n is implication of oneWell

## Example model: oneWell implies wagon1_n

```
context Train inv wagon1_n: self.wagon->size>=1
context Wagon inv train1_1: self.train->size=1
context Wagon inv succ0_1: self.succ->size<=1
context Wagon inv pred0_1: self.pred->size<=1

context Train inv oneWell:
 self.wagon->one(well| self.wagon->forAll(w|
  well.succPlus()->includesAll(w.succPlus())))
context Train inv noCycles:
 self.wagon->forAll(w|w.predPlus()->excludes(w))
context w1:Wagon inv trainComm:
 Wagon.allInstances->forAll(w2|
  w1.succ->includes(w2) implies w1.train=w2.train)
```



# CONSEQUENCES

```
context t1:Train inv distinctTrainsDistinctWagons:
  Train.allInstances->forAll(t2| t1<>t2 implies
    t1.allWagons()->intersection(t2.allWagons())->isEmpty())

use> open train_wagon.use
use> gen load distinctTrainsDistinctWagons.invs
     Added invariants: Train::distinctTrainsDistinctWagons
use> gen flags Train::distinctTrainsDistinctWagons +n
use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 9261.
     Checked 17862988 snapshots.
     Result: No valid state found.
```

This proves: Within the given finite search space (at most 2 trains and at most 4 wagons), distinctTrainsDistinctWagons is a consequence of the stated invariants

## Conclusion

- Consistency, independence and checking of UML and OCL models within the USE tool on the basis of test scenarios
- OCL is employed for formulating constraints, for reducing the test search space (in ASSL procedures), for formulating search space properties (by employing dynamically loaded invariants) and for focusing deductions (by switching off unneeded invariants)
- Approach based on interaction between building scenarios (through test cases) and studying system properties (through formulating properties and trying to giving proofs)

## Future work

- Reducing the search space
- ASSL search to be finished earlier in negative cases
- Show more information about the search space as well as valid and invalid invariants during the search
- User interface improvement
- Employ efficient SAT solver technology for checking properties like consistency or independence

## Thanks for your Attention!

**Appendix with USE project files**

- train_wagon.use
- train_wagon.assl
- train_wagon.invs
- *.pro files

```
train_wagon.use:

model TrainWorld

class Train
operations
allWagons():Set(Wagon)=
  self.wagon->union(self.wagon.predPlus()->asSet())->
    union(self.wagon.succPlus()->asSet())
end

------------------------------------------------------------------------

class Wagon
attributes
  numSeats: Integer
  isSmoker: Boolean
operations
predPlus():Set(Wagon)=
  self.predPlusOnSet(self.pred)
predPlusOnSet(s:Set(Wagon)):Set(Wagon)=
  let oneStep:Set(Wagon)=s.pred->asSet in
  if oneStep->exists(w|s->excludes(w))
    then predPlusOnSet(s->union(oneStep)) else s endif
succPlus():Set(Wagon)=
  self.succPlusOnSet(self.succ)
succPlusOnSet(s:Set(Wagon)):Set(Wagon)=
  let oneStep:Set(Wagon)=s.succ->asSet in
  if oneStep->exists(w|s->excludes(w))
    then succPlusOnSet(s->union(oneStep)) else s endif
end
```

```
association Ownership between
  Train[0..*] role train
  Wagon[0..*] role wagon
end

association Order between
  Wagon[0..*] role pred
  Wagon[0..*] role succ
end
```

```
constraints

-- dependent: implied by oneWell
context Train inv wagon1_n: self.wagon->size>=1

-- independent: gen_indep_train1_1.cmd
context Wagon inv train1_1: self.train->size=1

-- independent: gen_indep_succ0_1.cmd
context Wagon inv succ0_1: self.succ->size<=1

-- independent: gen_indep_pred0_1.cmd
context Wagon inv pred0_1: self.pred->size<=1

-- independent: gen_indep_oneWell.cmd
context Train inv oneWell:
  self.wagon->one(well| self.wagon->forAll(w|
    well.succPlus()->includesAll(w.succPlus())))

-- independent: gen_indep_noCycles.cmd
context Train inv noCycles:
  self.wagon->forAll(w|w.predPlus()->excludes(w))

-- independent: gen_indep_trainComm.cmd
context w1:Wagon inv trainComm:
  Wagon.allInstances->forAll(w2|
    w1.succ->includes(w2) implies w1.train=w2.train)
```

train_wagon.assl:

```
procedure genTrainsWagonsOwnershipOrder
  (countTrains:Integer,countWagons:Integer,
   countOwnership:Integer,countOrder:Integer)
-- countOwnership<=countTrains*countWagons
-- countOrder<=countWagons*countWagons
var theTrains:Sequence(Train), aTrain:Train,
    theWagons:Sequence(Wagon),
    aWagon:Wagon, aWagon2:Wagon;
begin
theTrains:=CreateN(Train,[countTrains]);
theWagons:=CreateN(Wagon,[countWagons]);
-- generate countOwnership links in Ownership
for i:Integer in [Sequence{1..countOwnership}]
  begin
  aTrain:=Try([theTrains]);
  aWagon:=Try([theWagons->reject(w|w.train->includes(aTrain))]);
  Insert(Ownership,[aTrain],[aWagon]);
  end;
-- generate countOrder links in Order
for i:Integer in [Sequence{1..countOrder}]
  begin
  aWagon:=Try([theWagons]);
  aWagon2:=Try([theWagons->reject(w|w.pred->includes(aWagon))]);
  Insert(Order,[aWagon],[aWagon2]);
  end;
end;
```

train_wagon.assl:

```
procedure genMaxCountTrainsMaxCountWagons
  (maxCountTrains:Integer,maxCountWagons:Integer)
var wagons:Sequence(Wagon), trains:Sequence(Train),
    actualCountTrains:Integer, actualCountWagons:Integer;
begin
actualCountTrains:=Try([Sequence{1..maxCountTrains}]);
actualCountWagons:=Try([Sequence{1..maxCountWagons}]);
trains:=CreateN(Train,[actualCountTrains]);
wagons:=CreateN(Wagon,[actualCountWagons]);
Try(Ownership,[trains],[wagons]);
Try(Order,[wagons],[wagons]);
end;
```

train_wagon.invs:

```
context t1:Train inv trainSizeBalanced: Train.allInstances->forAll(t2|
  t1<>t2 implies (t1.wagon->size-t2.wagon->size+1).abs<=1)
```

```
use> open train_wagon.use

use> gen load trainSizeBalanced.invs
     Added invariants: Train::trainSizeBalanced

use> gen start train_wagon.assl genTrainsWagonsOwnershipOrder(2,4,4,2)
use> gen result
     Random number generator was initialized with 6315.
     Checked 5786 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train1,Train2 : Train
     !create Wagon1,Wagon2,Wagon3,Wagon4 : Wagon
     !insert (Train1,Wagon1) into Ownership
     !insert (Train1,Wagon2) into Ownership
     !insert (Train2,Wagon3) into Ownership
     !insert (Train2,Wagon4) into Ownership
     !insert (Wagon1,Wagon2) into Order
     !insert (Wagon3,Wagon4) into Order
use> gen result accept
     Generated result (system state) accepted.
```

```
use> open train_wagon.use

use> gen flags Train::noCycles +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 9864.
     Checked 4 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train1 : Train
     !create Wagon1 : Wagon
     !insert (Train1,Wagon1) into Ownership
     !insert (Wagon1,Wagon1) into Order
use> gen result accept
     Generated result (system state) accepted.
```

```
use> open train_wagon.use

use> gen flags Wagon::pred0_1 +n
use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(1,4)
use> gen result
     Random number generator was initialized with 7489.
     Checked 987597 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train4 : Train
     !create Wagon7,Wagon8,Wagon9,Wagon10 : Wagon
     !insert (Train4,Wagon10) into Ownership
     !insert (Train4,Wagon9) into Ownership
     !insert (Train4,Wagon8) into Ownership
     !insert (Train4,Wagon7) into Ownership
     !insert (Wagon9,Wagon7) into Order
     !insert (Wagon8,Wagon10) into Order
     !insert (Wagon7,Wagon10) into Order
use> gen result accept
     Generated result (system state) accepted.
```

```
use> open train_wagon.use

use> gen flags Wagon::succ0_1 +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 4715.
     Checked 3659 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train3 : Train
     !create Wagon4,Wagon5,Wagon6 : Wagon
     !insert (Train3,Wagon6) into Ownership
     !insert (Train3,Wagon5) into Ownership
     !insert (Train3,Wagon4) into Ownership
     !insert (Wagon4,Wagon6) into Order
     !insert (Wagon4,Wagon5) into Order
use> gen result accept
     Generated result (system state) accepted.
```

```
use> open train_wagon.use

use> gen flags Wagon::train1_1 +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 3211.
     Checked 21 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train2 : Train
     !create Wagon2,Wagon3 : Wagon
     !insert (Train2,Wagon2) into Ownership
use> gen result accept
     Generated result (system state) accepted.
```

```
use> open train_wagon.use

use> gen flags Wagon::trainComm +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,4)
use> gen result
     Random number generator was initialized with 9852.
     Checked 1052847 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Train7,Train8 : Train
     !create Wagon12,Wagon13 : Wagon
     !insert (Train8,Wagon12) into Ownership
     !insert (Train7,Wagon13) into Ownership
     !insert (Wagon12,Wagon13) into Order
use> gen result accept
     Generated result (system state) accepted.
```

```
use> open train_wagon.use

use> gen flags Train::wagon1_n +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,3)
use> gen result
     Random number generator was initialized with 5785.
     Checked 37196 snapshots.
     Result: No valid state found.
```

```
use> open train_wagon.use

use> gen flags Train::wagon1_n  +n

use> gen flags Train::noCycles  +d
use> gen flags Wagon::pred0_1   +d
use> gen flags Wagon::succ0_1   +d
use> gen flags Wagon::train1_1  +d
use> gen flags Wagon::trainComm +d

use> gen flags Train::oneWell   -d

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,3)

use> gen result
     Random number generator was initialized with 4575.
     Checked 37196 snapshots.
     Result: No valid state found.

use> gen result inv
     Note: A disabled invariant has never been checked.
     An enabled and negated invariant is `valid'
     if it has been evaluated to false.

     checks    valid  invalid  Invariant
          0        0        0  model-inherent multiplicities
      33436     4852    28584  Train::wagon1_n (negated)
      10527     1915     8612  Train::oneWell
          0        0        0  Train::noCycles (disabled)
          0        0        0  Wagon::pred0_1 (disabled)
          0        0        0  Wagon::succ0_1 (disabled)
          0        0        0  Wagon::train1_1 (disabled)
          0        0        0  Wagon::trainComm (disabled)
```

```
use> open train_wagon.use

use> gen load distinctTrainsDistinctWagons.invs
     Added invariants: Train::distinctTrainsDistinctWagons

use> gen flags Train::distinctTrainsDistinctWagons +n

use> gen start train_wagon.assl genMaxCountTrainsMaxCountWagons(2,3)
use> gen result
     Random number generator was initialized with 9261.
     Checked 37196 snapshots.
     Result: No valid state found.

use> gen result inv
     Note: A disabled invariant has never been checked.
     An enabled and negated invariant is `valid'
     if it has been evaluated to false.

     checks valid invalid  Invariant
          0     0       0  model-inherent multiplicities
      27497  2341   25156  Train::noCycles
      16863  6859   10004  Train::distinctTrainsDistinctWagons (negated)
       2056   156    1900  Wagon::train1_1
        136     0     136  Wagon::trainComm
          1     1       0  Train::oneWell
          1     1       0  Train::wagon1_n
          1     1       0  Wagon::pred0_1
          1     1       0  Wagon::succ0_1
```

Thanks again!