Modeling the Car Sequencing Problem

Gerrit Renker

The Robert Gordon University

Aberdeen

Requirements

- cars can be configured with various *options*
  - indicated by the *car type*
  - for each car type, there is a *demand* of cars.
- *capacity constraints:*
  - at most M out of a sequence of N cars may have option i

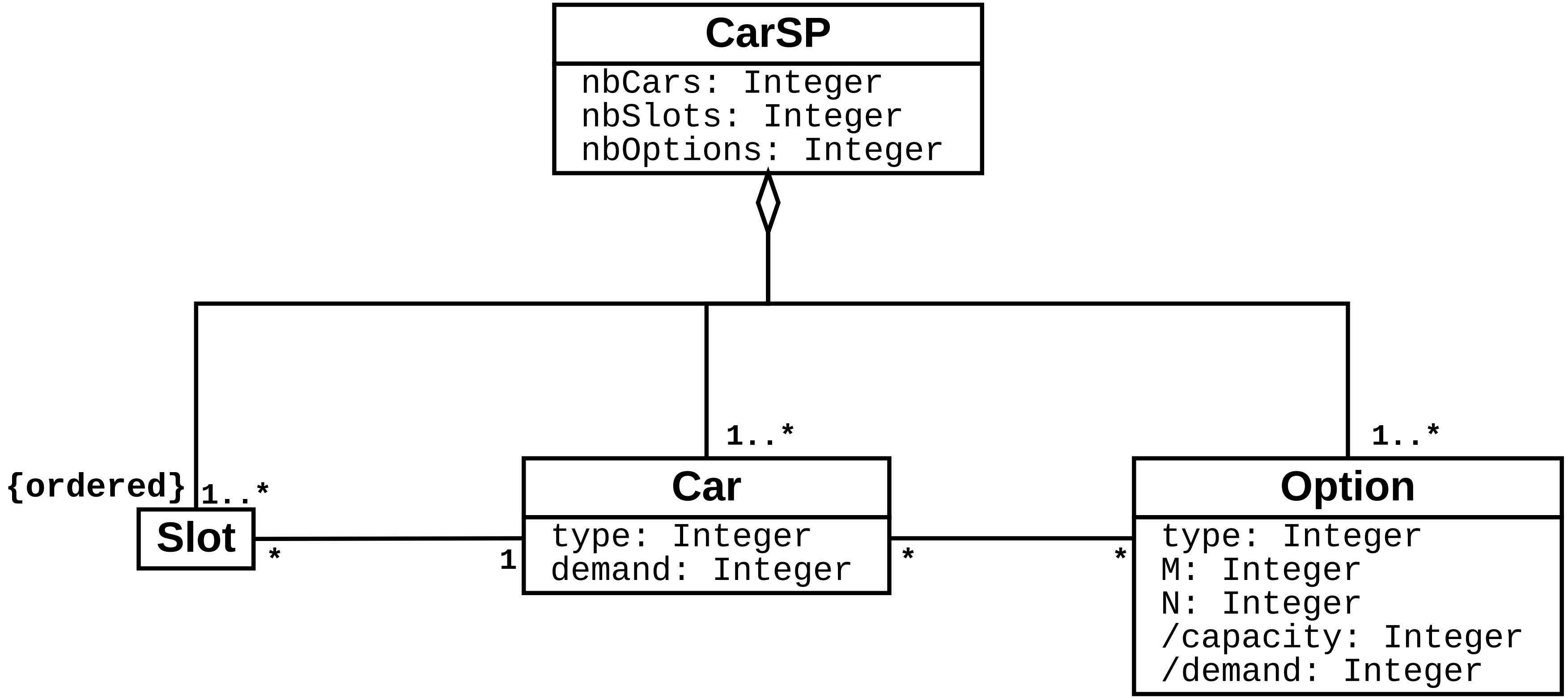
goal:
arrange the *sequence of all requested cars*
such that none of the capacity constraints is violated

Instance Data for the Car Sequencing Problem

Option		Capacity	Car Type		
Name	Type	M/N	1	2	3
Sunroof	1	2/3	1	1	0
Radio	2	3/4	1	0	1
Air Cond.	3	2/3	0	1	1
Number of Cars			10	20	20

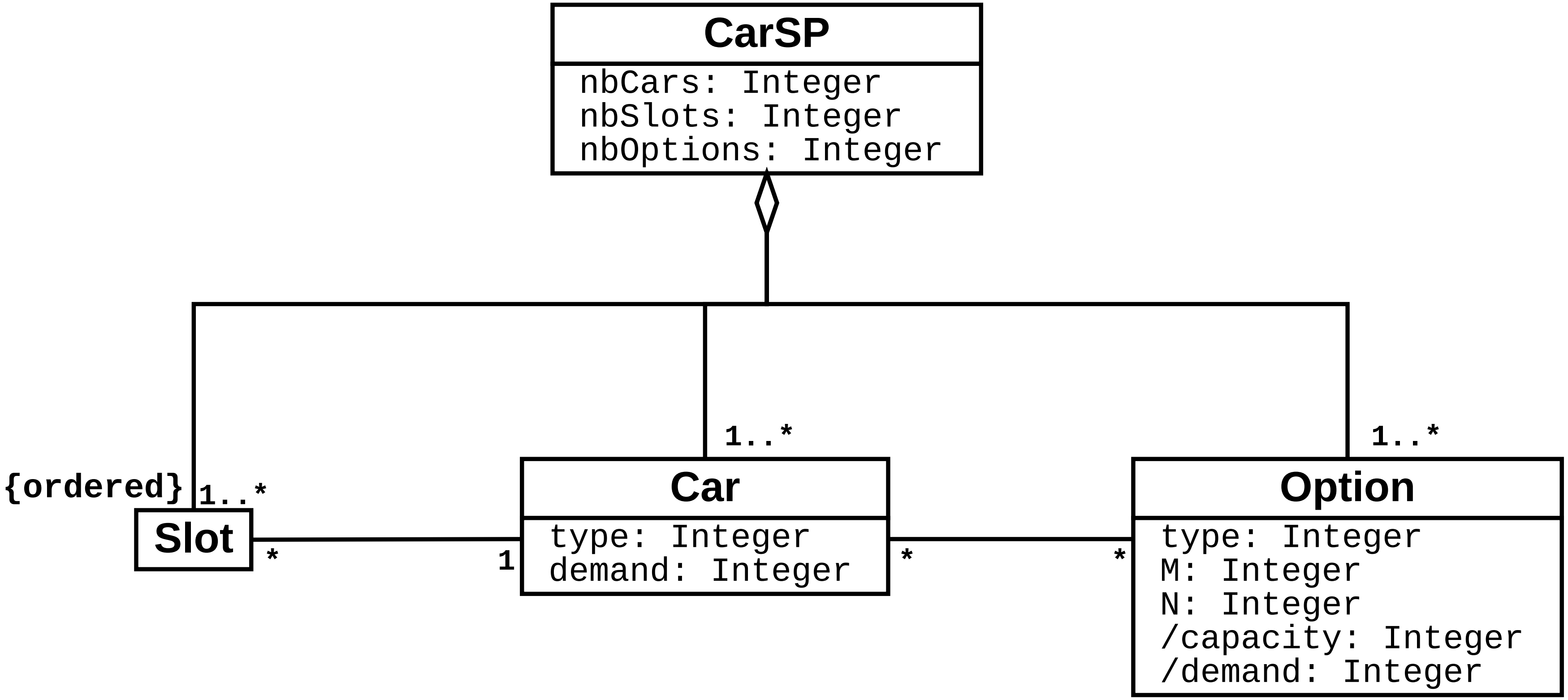
- *total of 50 cars:*
  - 30 cars with sunroofs
  - 30 cars with radios fitted
  - 40 cars with air conditioning

Structural Model



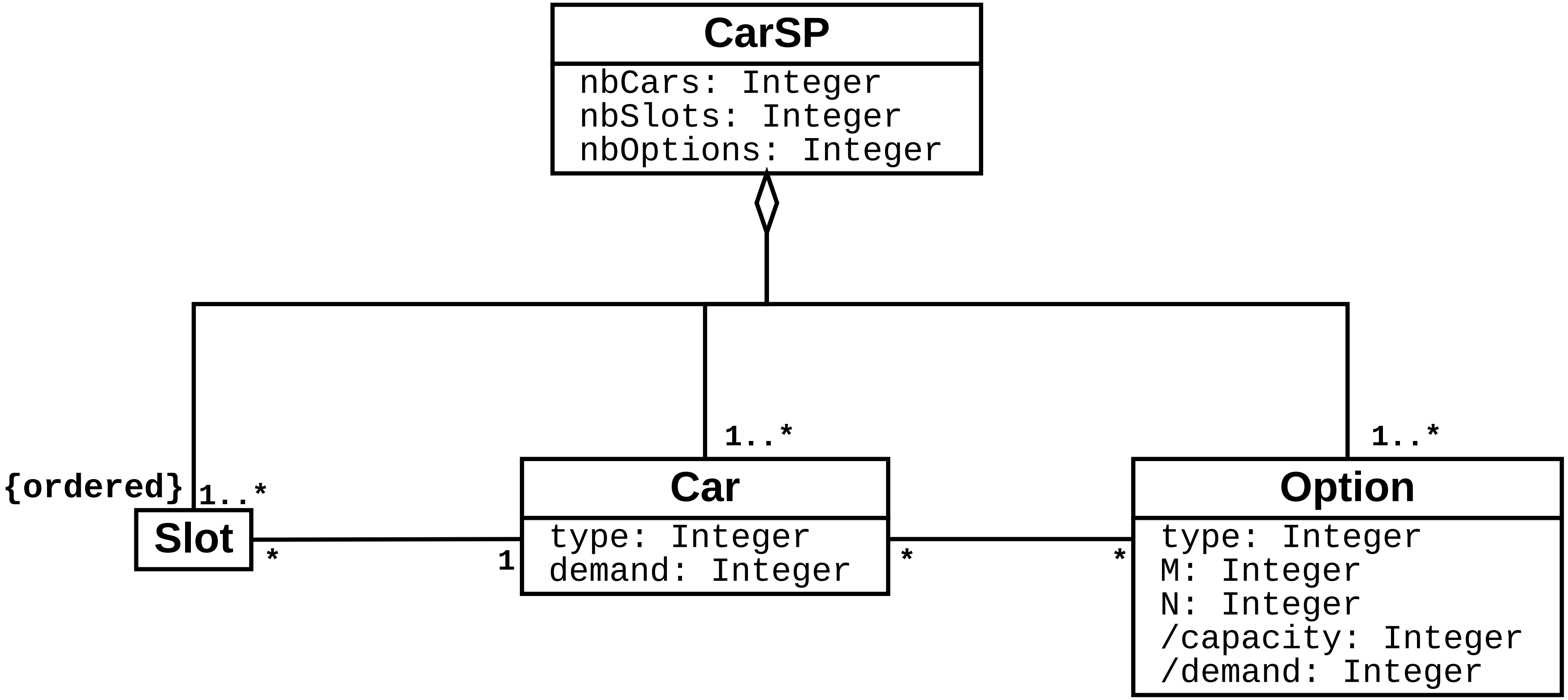
- `Car` represents possible car types
  - characterized by `type` and `demand` attributes
- `Slot:` the sequence of `Car` types

Multiplicity Constraints



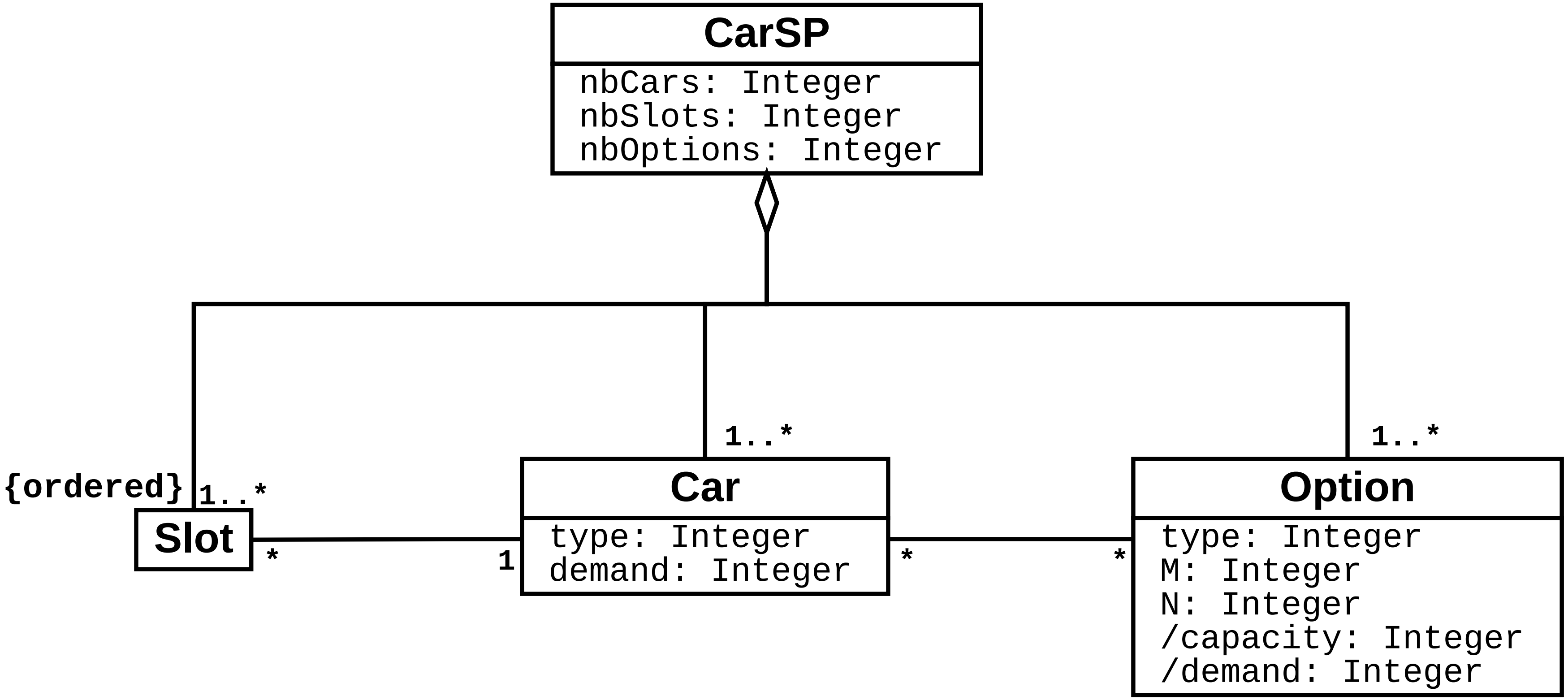

```
context CarSP inv:
  nbCars = car->size() and
  nbSlots = slot->size() and
  nbOptions = option->size()
```

Domains as Unary Constraints



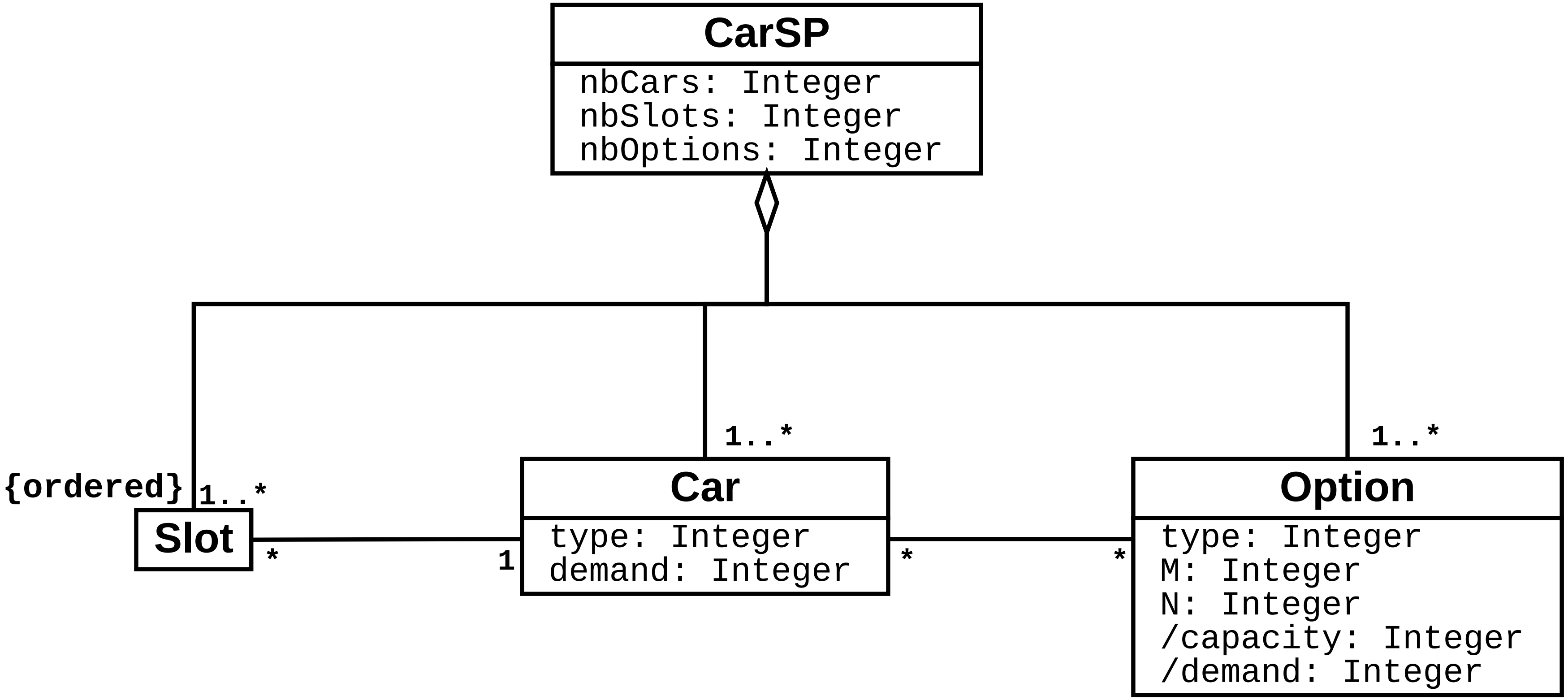

```
context Option inv:
  type > 0 and type <= 3


context CarSP inv:      -- uniqueness constraint
  option->isUnique(type)
```

Instantiation: tabled values for the domain of Car




```
context Car inv:
type = 1 implies (demand = 10 and option.type = Set{1,2})
  and
type = 2 implies (demand = 20 and option.type = Set{1,3})
  and
type = 3 implies (demand = 20 and option.type = Set{2,3})
```

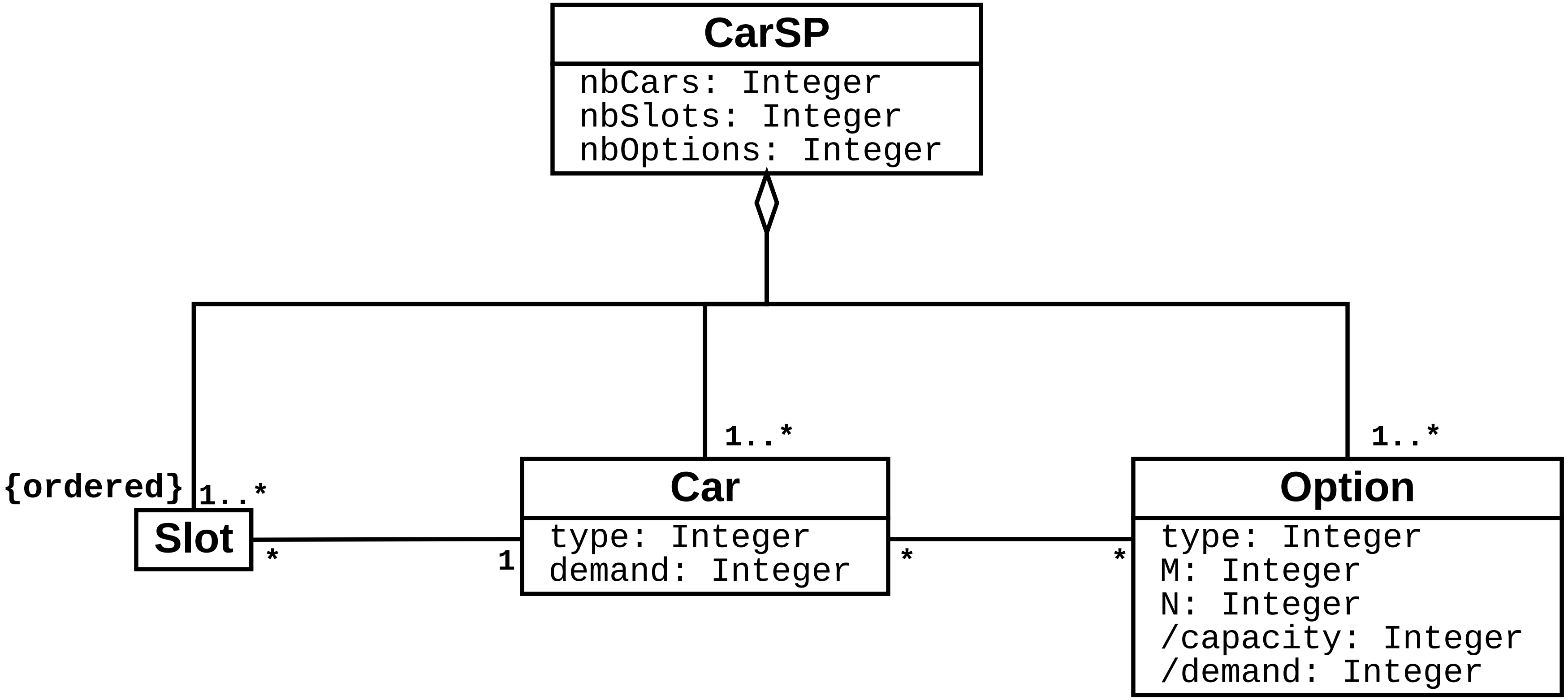
Computation of Derived Attributes



- can always be derived from other attributes

```
context Option inv:
  capacity = M/N and
  demand = car.demand->sum()
```

Capacity Constraints ("*at most*" - constraints)




```
context CarSP inv:
option->forAll(o|
 Sequence{1..(nbSlots - o.N + 1)}->forAll(i|
  slot->subSequence(i, i + o.N - 1).car.option->count(o)
                                                  <= o.M ))
```

End