Un lieu de transport est identifié par trois lettres uniques à chaque lieu.

```
context LieuTransport inv:
  self.sigle->size() = 3 and
  LieuTransport.allInstances()->forAll(l | l.sigle <> self.sigle)
```

Une compagnie de transport est identifiée par moins de six caractères uniques à chaque compagnie.

```
context Compagnie inv:
  self.companyID->size() < 6 and
  Compagnie.allInstances()->forAll(c | c.companyID <> self.companyID)
```

Un voyage est identifié par un ID qui commence par deux lettres suivi d'une série de chiffres. La partie alphabétique de l'ID d'un voyage est unique à chaque compagnie et la partie numérique est unique à chaque voyage au sein de la même compagnie.

```
context Voyage inv:
  (self.voyageID.startsWith(self.Compagnie.companyID.substring(1, 2))) and
  (self.voyageID.endsWith(self.numVoyage.toString())) and
  Voyage.allInstances()->forAll(v | v.voyageID <> self.voyageID)
```

Les aéroports (les gares) de départ et d'arrivée d'un voyage doivent être différents.

```
context Aeroport inv:
  self.depart <> self.destination
```

Les gares de départ et d'arrivée d'un voyage doivent être différents.

```
context Gare inv :
   self.depart <> self.destination
```

Tous les sièges d'une même section ont le même prix.

```
context SectionAvionTrain inv:
  self.sieges->forAll(s | s.prix = self.prix)
```

Un itinéraire ne peut pas durer plus de 21 jours.

```
context Itineraire inv:
  (self.arrivee.toDay() - self.depart.toDay ()) <= 21
```

Le port de départ et d'arrivée doit être le même.

context Port **inv:**
  self.lieuDepart.sigle **= self**.lieuDestination.sigle

Un paquebot peut être assigné à plusieurs itinéraires tant qu'ils ne se chevauchent pas.

context Itineraire **inv:**
  Itineraire.**allInstances()**
    ->**select(**i **|** i.paquebot **= self**.paquebot **and** i **<> self)**
    ->**forAll(**i **| self.**arrivee **<=** i.depart **or** i.arrivee **<= self**.depart**)**

Toutes les cabines d'une même section ont le même prix.

context SectionPaquebot **inv**:
  self.cabines->**forAll(**s **|** s.prix **= self**.prix**)**

Le client peut réserver un siège disponible dans un vol (trajet) donné.

context Reservation
  pre: self.siege.disponibilite **= true**
  post: self.siege.disponibilite **= false**

Un siège réservé devient assigné à un passager une fois payé : le siège est donc confirmé

context ControllerReservation**::**payerSiege(numReservation : string, client : Client, carte : CarteCredit)
  let res **:** Reservation **=**
    Reservation**.allInstances()**->**any(**r **|** r.numReservation **=** numReservation**) in**
  pre : res.**notEmpty() and** res.etat **=** Etat**::**Reserve **and** res.siege.disponibilite **= false**
  post : res.etat **=** Etat**::**Confirme

Le client peut réserver une cabine disponible pour un itinéraire donné.

context Reservation
  pre: self.cabine.disponibilite **= true**
  post: self.cabine.disponibilite **= false**

Le choix de la priorité est seulement pour l'avion.

```
context Siege inv:
  self.priorite.notEmpty() implies
    self.Section.oclIsTypeOf(SectionAvionTrain)
```

Le lieu de transport de départ doit être le même que celui d'arrivée.

```
context Voyage inv:
  self.depart.oclType() = self.arrivee.oclType()
```

Un vol n'a pas d'escales.

```
context Voyage inv:
  self.Voyage.oclIsTypeOf(Vol) implies
    self.escales->isEmpty()
```

Les sections prédéfinis pour les trajets sont seulement Premiere et Economique.

```
context SectionAvionTrain inv:
  self.Voyage.oclIsKindOf(Trajet) implies
  (self.classe = ClasseAT::Premiere or self.classe = ClasseAT::Economique)
```

La disposition des sièges du train est Etroit.

```
context SectionAvionTrain inv:
  self.Voyage.oclIsKindOf(Trajet) implies
    self.disposition = Disposition::Etroit
```