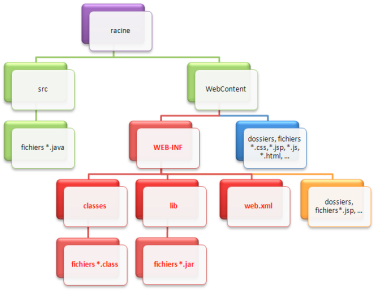
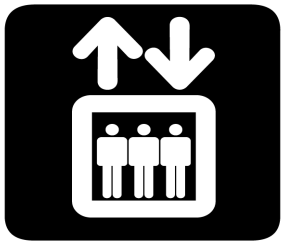
Retour sur les notions déjà étudiées

- Analyse (diagramme de Cas d'Utilisation)

- Modélisation statique (diagramme de Classes)
  - Principe de forte cohésion et pattern méta-classe
  - Principe de couplage entre packages (faible dépendances)
  - Structuration hiérarchique et pattern composite



- Modélisation dynamique (diagrammes de classes/objets/collaboration)
  - Gestion d'événements et pattern observateur



Menu du jour :

- Modélisation dynamique (diagrammes de classes/objets/collaboration/Etats-Transitions)
  - Gestion d'événements et pattern observateur
  - Gestion des états d'un objet et **pattern état**

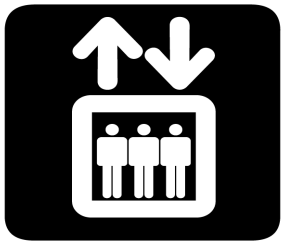


Diagramme d'états-transitions
&
Patron de conception « état »

Retour d'ascenseur

Description simplifiée des actions possibles pour un passager :

- Dans la cabine
  - Ouvrir/fermer les portes (2 boutons)
  - Choisir étage (boutons numérotés)
  - Appel d'urgence
- Dans le couloir
  - Appeler la cabine (1 bouton)

Retour d'ascenseur

Description **simplifiée** du fonctionnement de la cabine par rapport aux actions réalisées :

- L'ouverture et la fermeture des portes ne peuvent être réalisées que lorsque la cabine est à l'arrêt
- Si aucun choix d'étage n'est enregistré, une cabine à l'arrêt et portes fermées se met en attente (lumière éteinte) sinon elle se met en déplacement
- Une cabine en déplacement s'arrête dès qu'elle a atteint l'étage choisi
- Une cabine en attente se met en déplacement dès qu'elle est appelée ou qu'un choix d'étage est réalisé

Modélisation par un diagramme UML d'Etats-Transition

Diagramme dynamique permettant de décrire les changements d'états d'**un objet** en réponse aux interactions d'autres objets ou acteurs.

Etat : un état est défini par la valeur (instantanée) des attributs de l'objet et de ses liens avec d'autres objets.

Transition : une transition représente le passage instantané d'un état vers un autre état. Elle est déclenchée par un événement (appel opération)

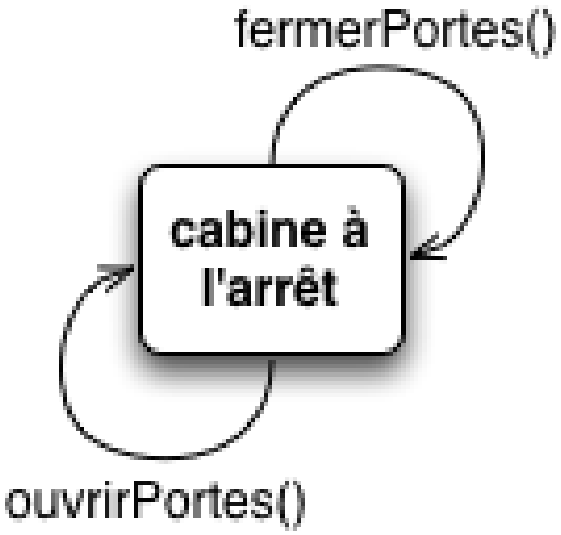
Question

Quels sont les différents états possibles d'une cabine d'ascenseur ?

- L'ouverture et la fermeture des portes ne peuvent être réalisées que lorsque la cabine est **à l'arrêt**
- Si aucun choix d'étage n'est enregistré, une cabine à l'arrêt et portes fermées se met **en attente** (lumière éteinte) sinon elle se met en déplacement
- Une cabine **en déplacement** s'arrête dès qu'elle a atteint l'étage choisi
- Une cabine en attente se met en déplacement dès qu'elle est appelée ou qu'un choix d'étage est réalisé

A modéliser

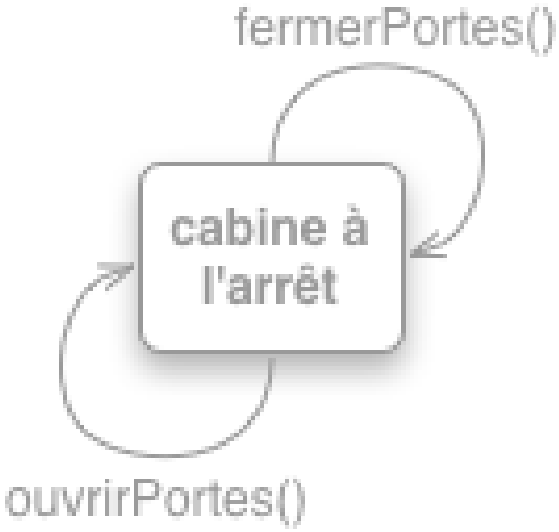
*L'ouverture et la fermeture des portes ne peuvent être réalisées que lorsque la cabine est **à l'arrêt***

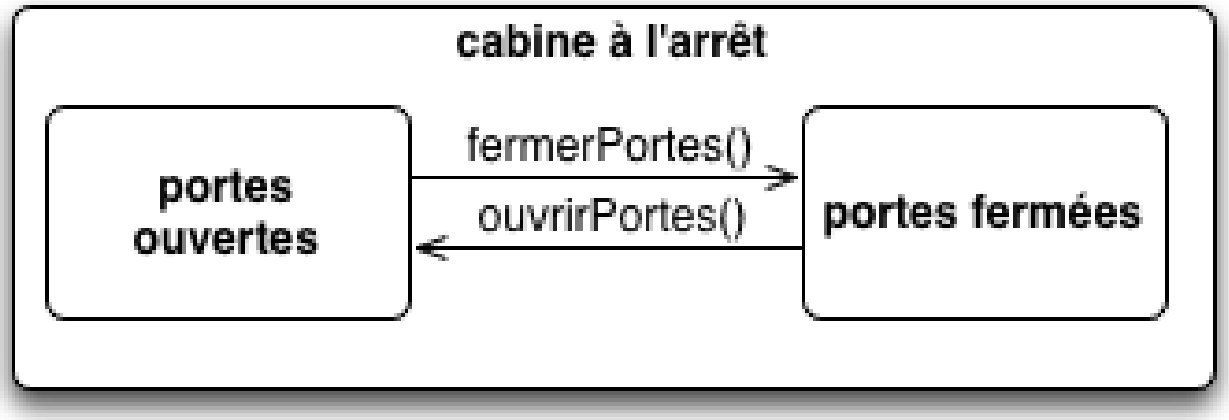


Un état peut être décrit lui-même par un diagramme d'états-transitions. On parle d'**état composé**.

A modéliser

*L'ouverture et la fermeture des portes ne peuvent être réalisées que lorsque la cabine est **à l'arrêt***

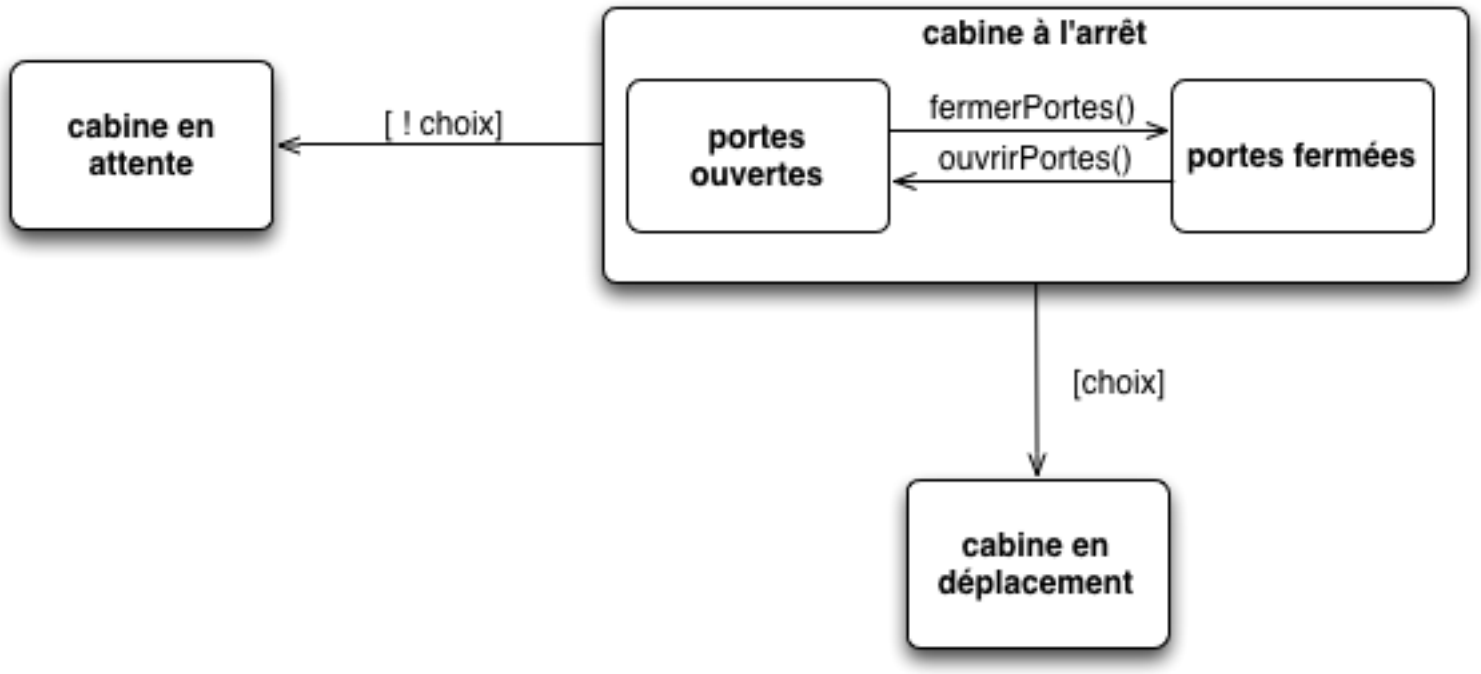




Un état peut être décrit lui-même par un diagramme d'états-transitions. On parle d'**état composé**.

A modéliser

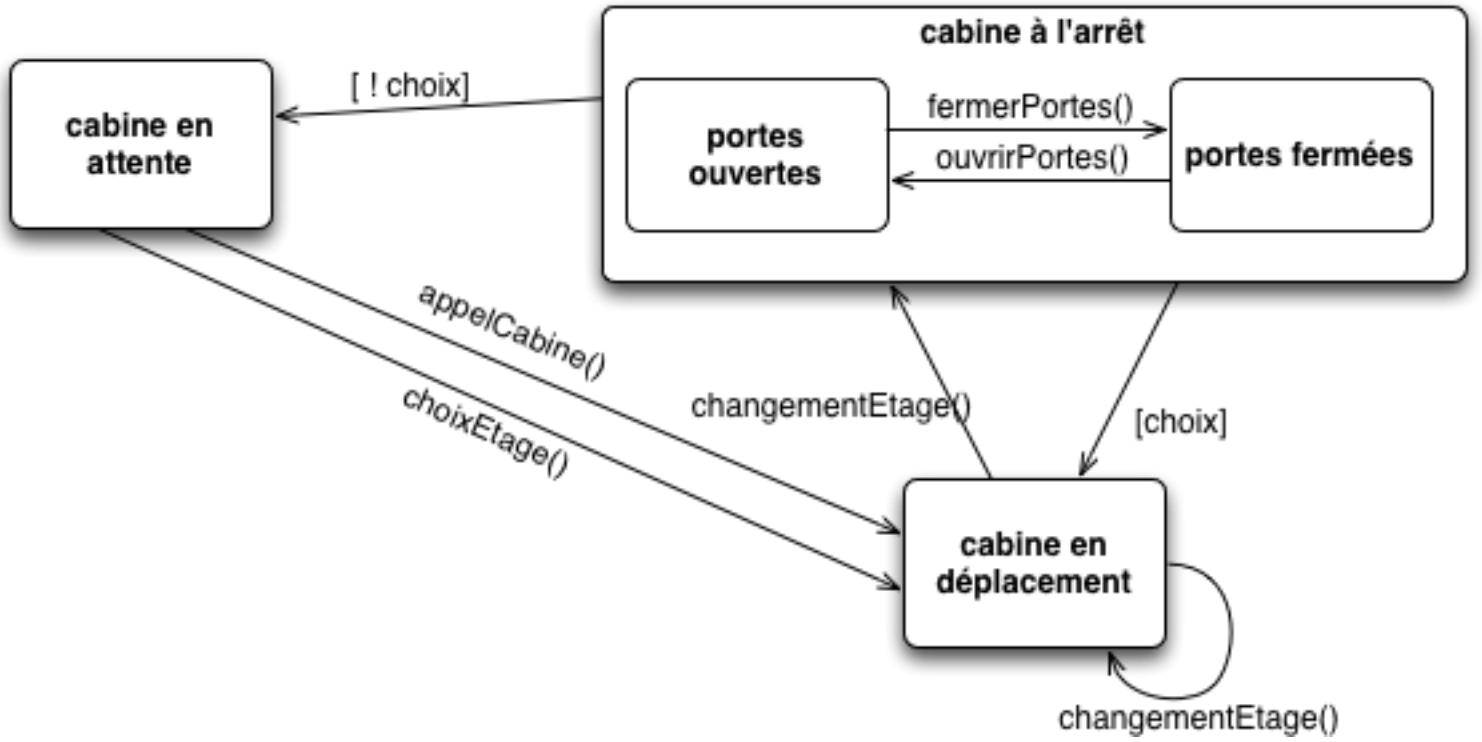
*Si aucun choix d'étage n'est enregistré, une cabine à l'arrêt et portes fermées se met **en attente** (lumière éteinte) sinon elle se met **en déplacement***



Une transition peut être **conditionnée** et/ou **automatique** (sans appel d'opération)

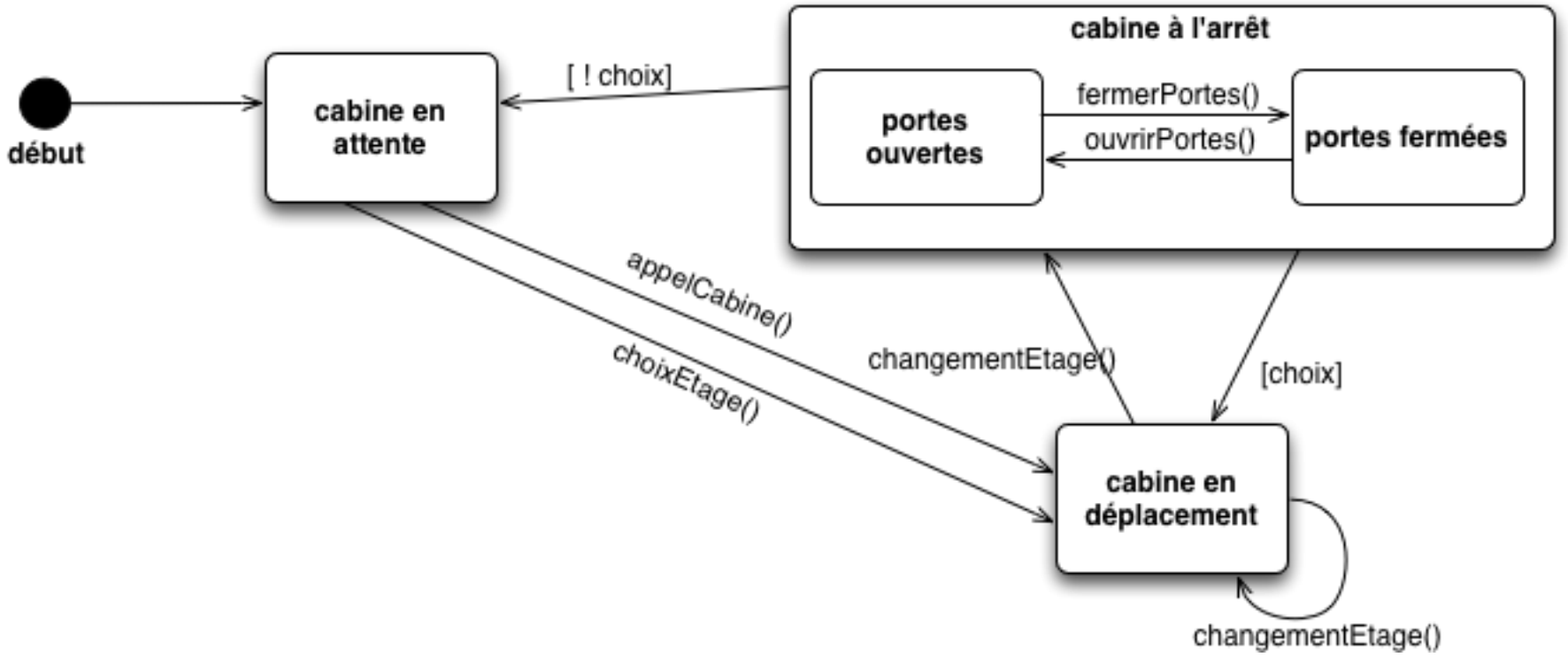
A modéliser

Une cabine en attente se met en déplacement dès qu'elle est appelée ou qu'un choix d'étage est réalisé



A modéliser (compléments)

A la réinitialisation du système « ascenseur », la cabine est par défaut en attente.



L'**état initial** indique l'état de l'objet lors de sa création.

A modéliser (compléments)

Les portes de la cabine sont nécessairement fermées pour pouvoir entrer et sortir de l'état d'arrêt.

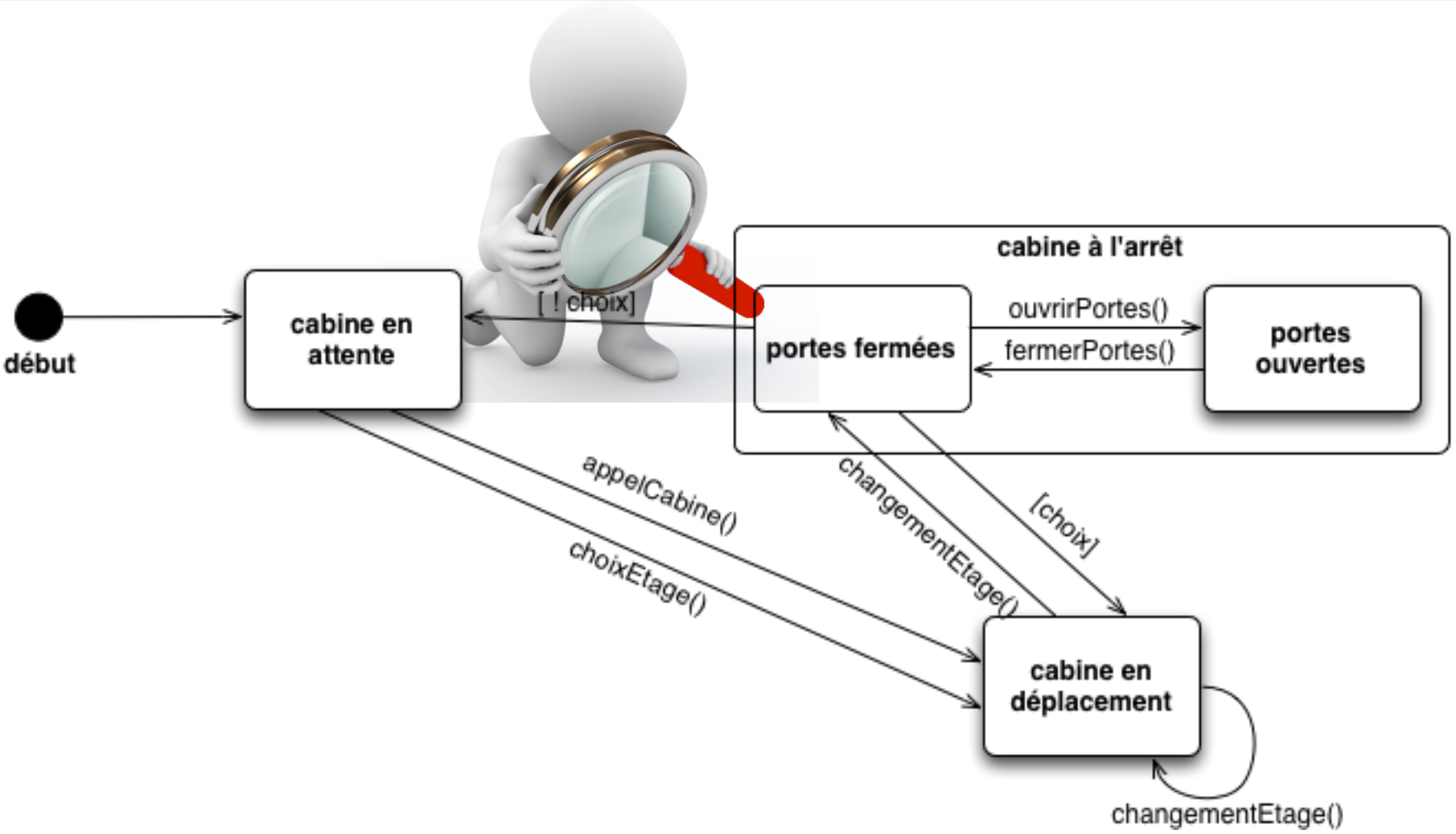


Diagramme UML d'Etats-Transition

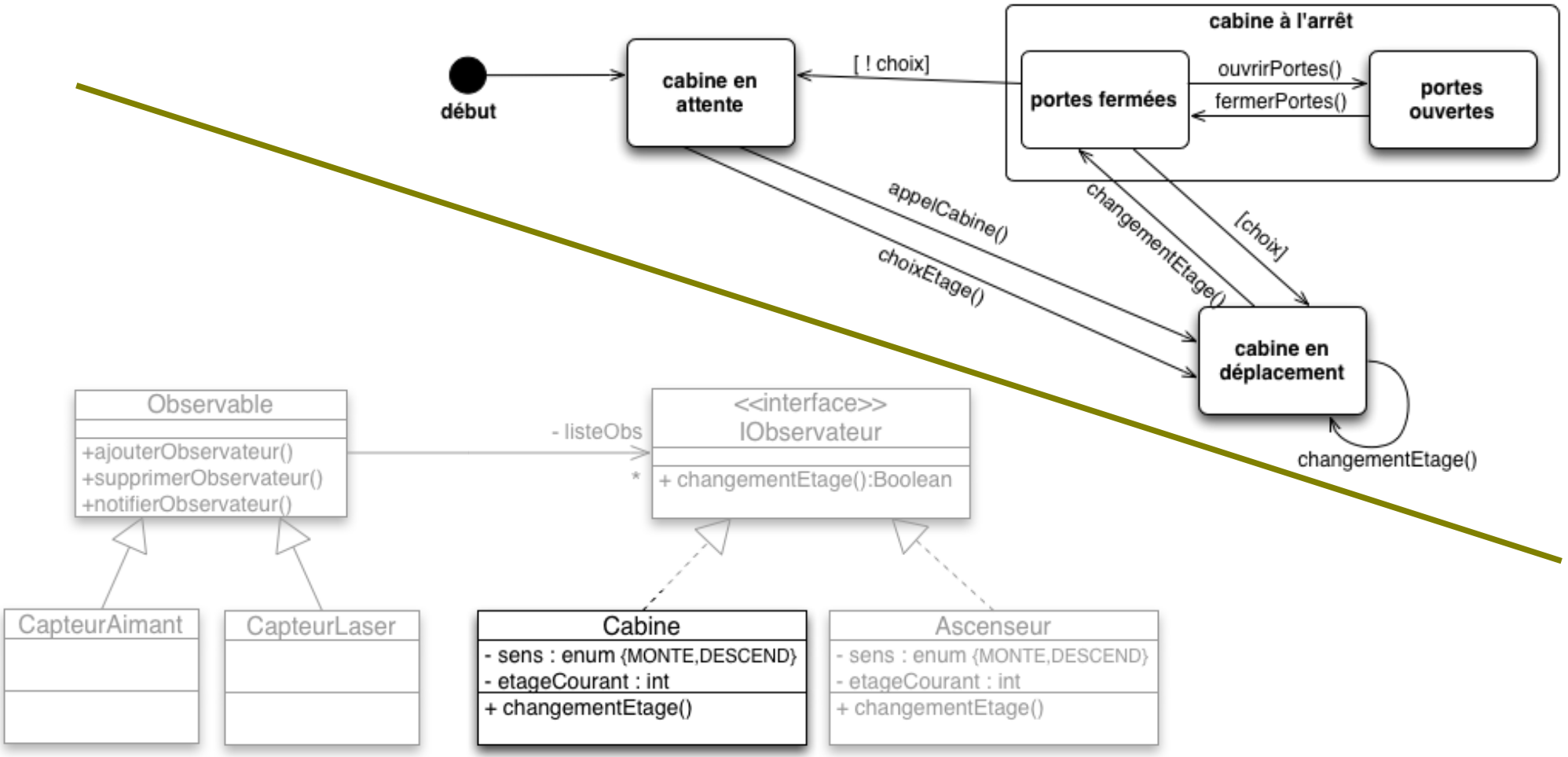
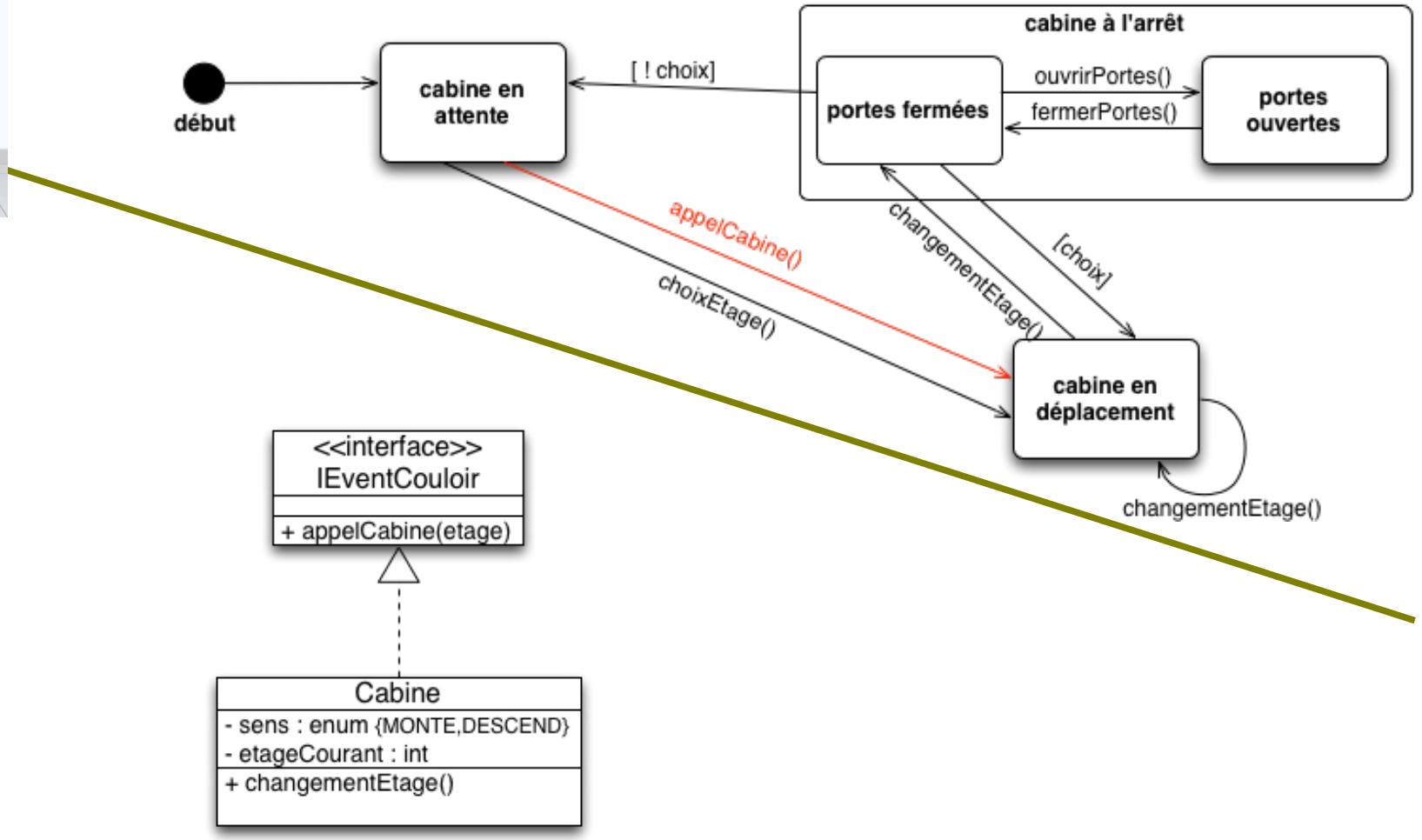
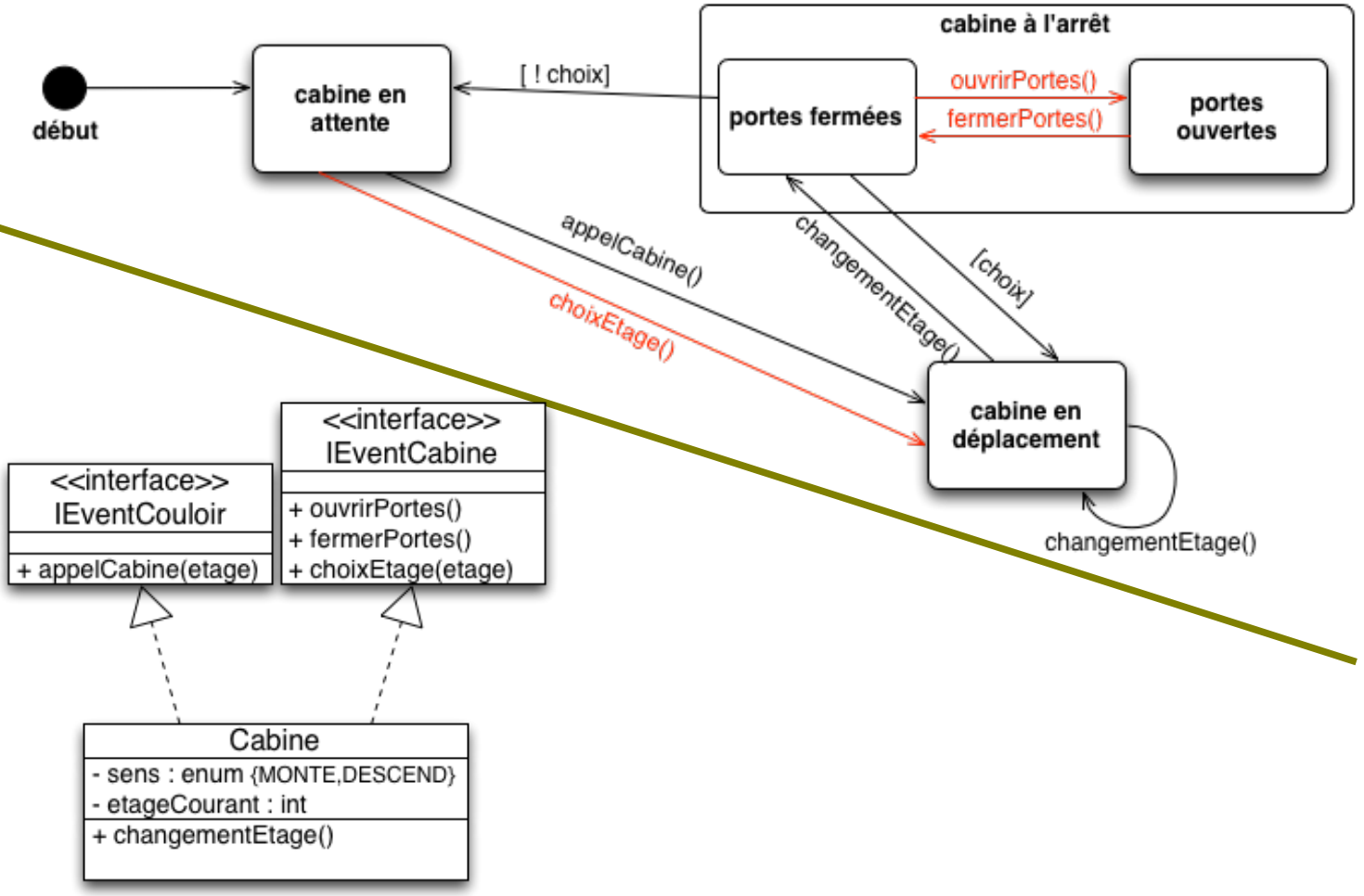


Diagramme UML de classes (temporaire)

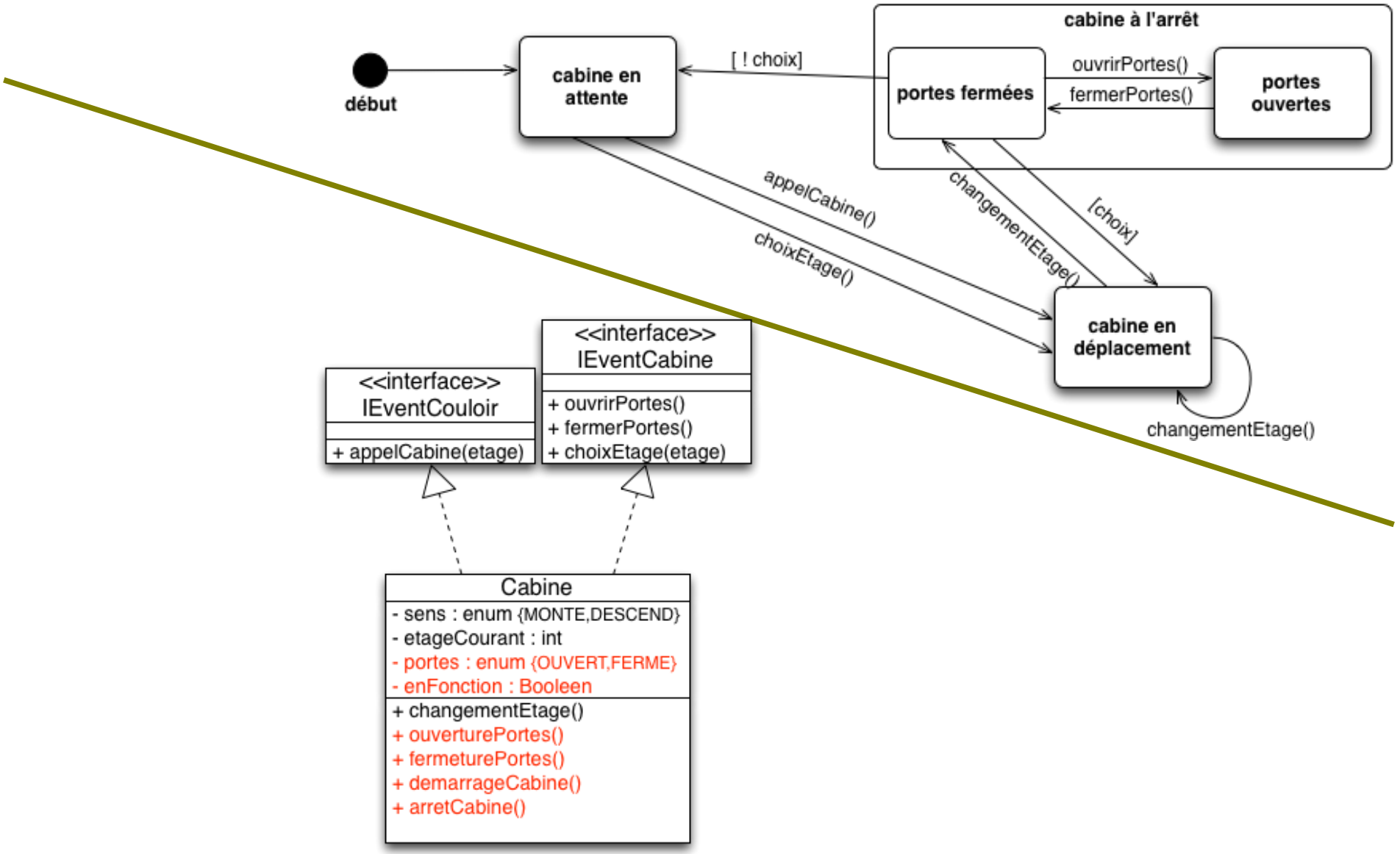
Appel de la cabine du couloir



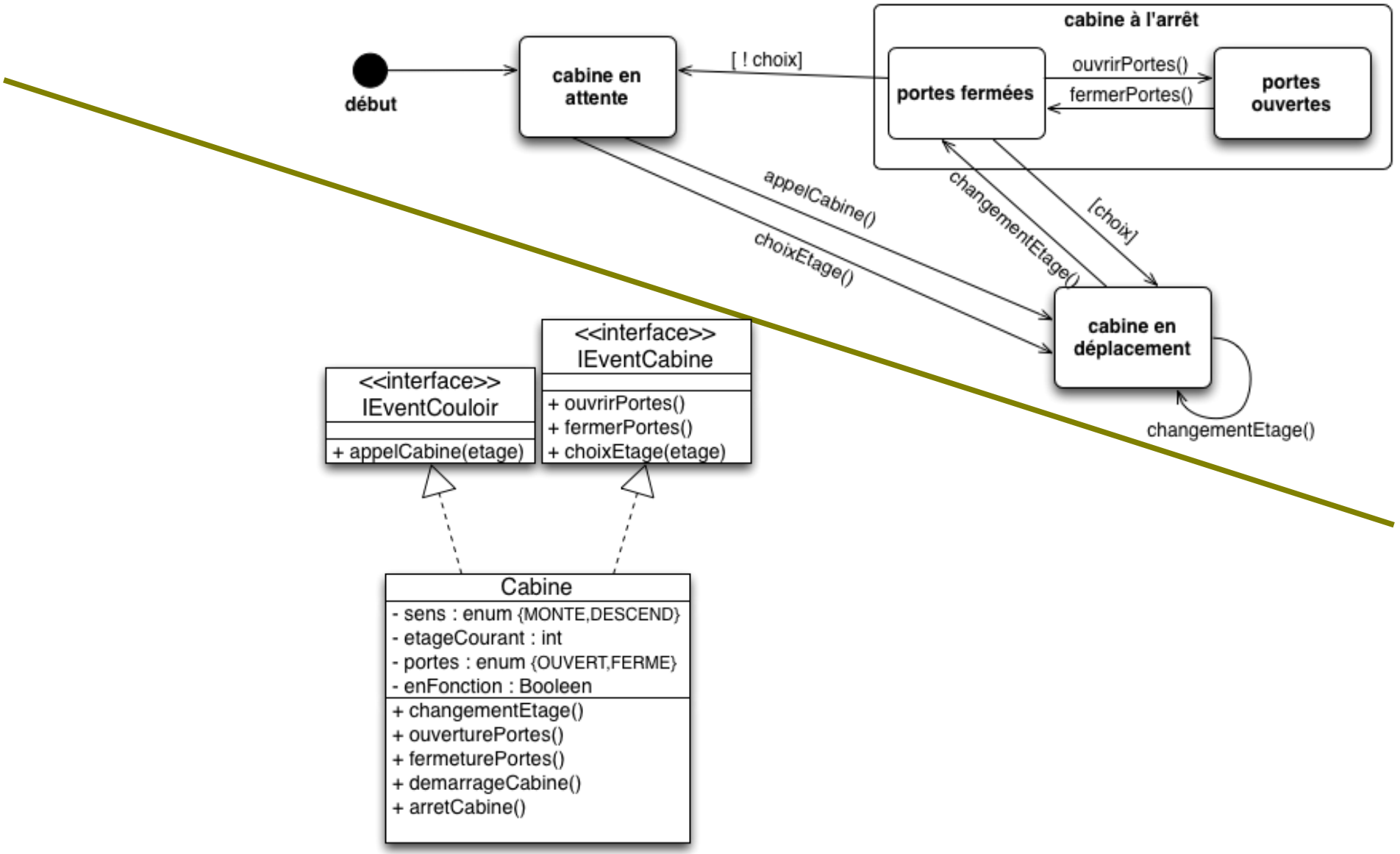
Commandes de la cabine : ouvrir/fermer portes, choisir étage



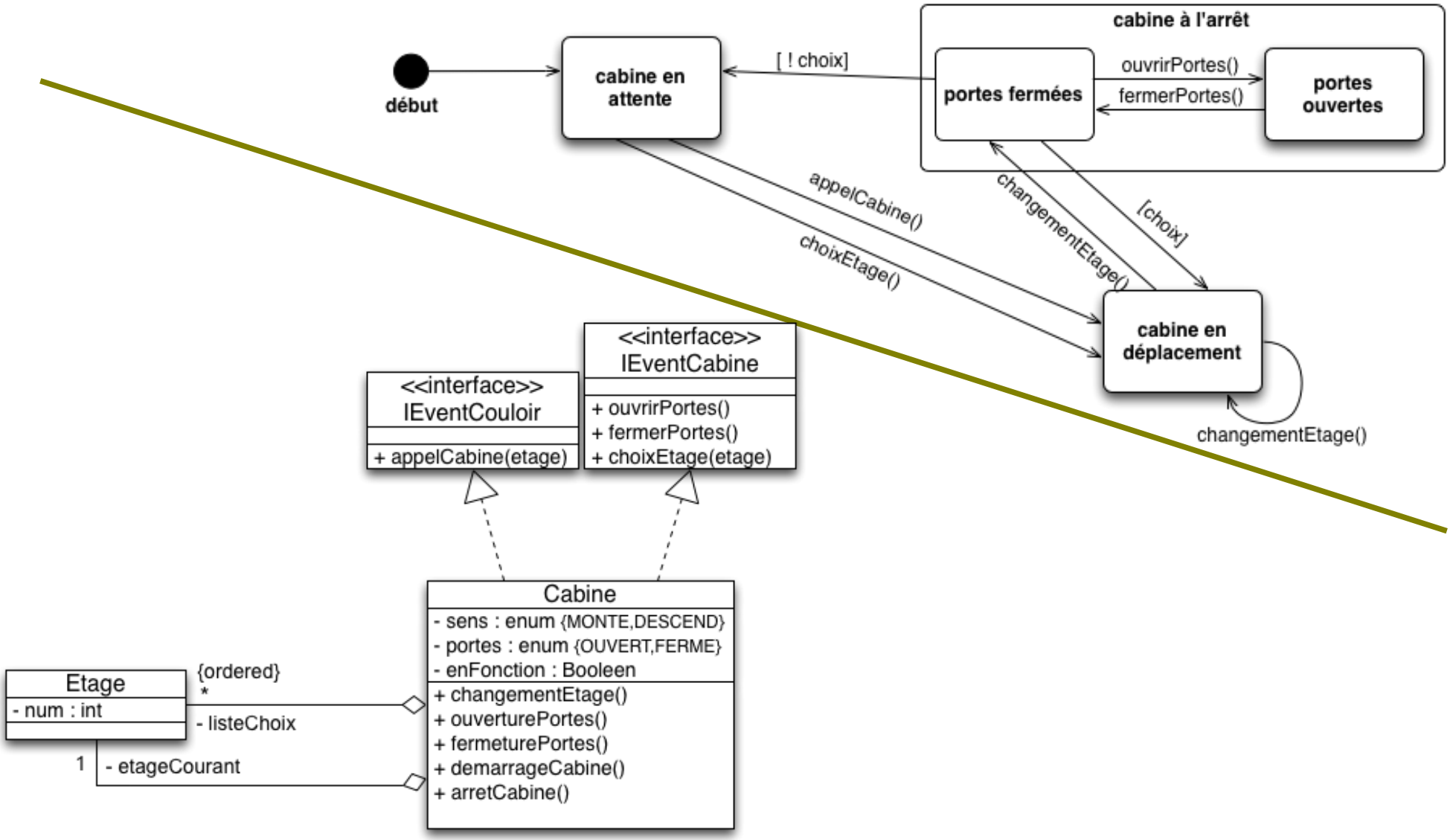
Evénements dont la cabine est responsable : ouverture/fermeture des portes, démarrage, arrêt, ...



La cabine doit aussi pouvoir gérer (la liste) des étages à desservir



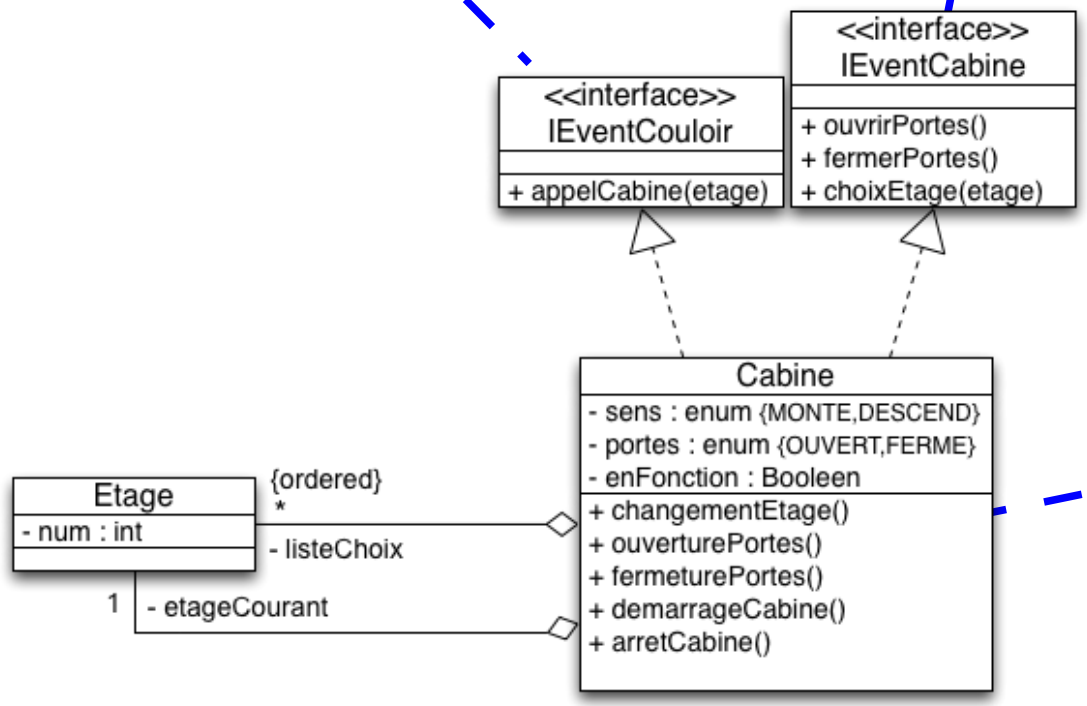
La cabine doit aussi pouvoir gérer (la liste) des étages à desservir



Un peu de code...

```
interface IEventCouloir{

        public void appelCabine(Etage e);
}
```

```
interface IEventCabine{

        public void ouvrirPortes();
        public void fermerPortes();
        public void choixEtage(Etage e);
}
```

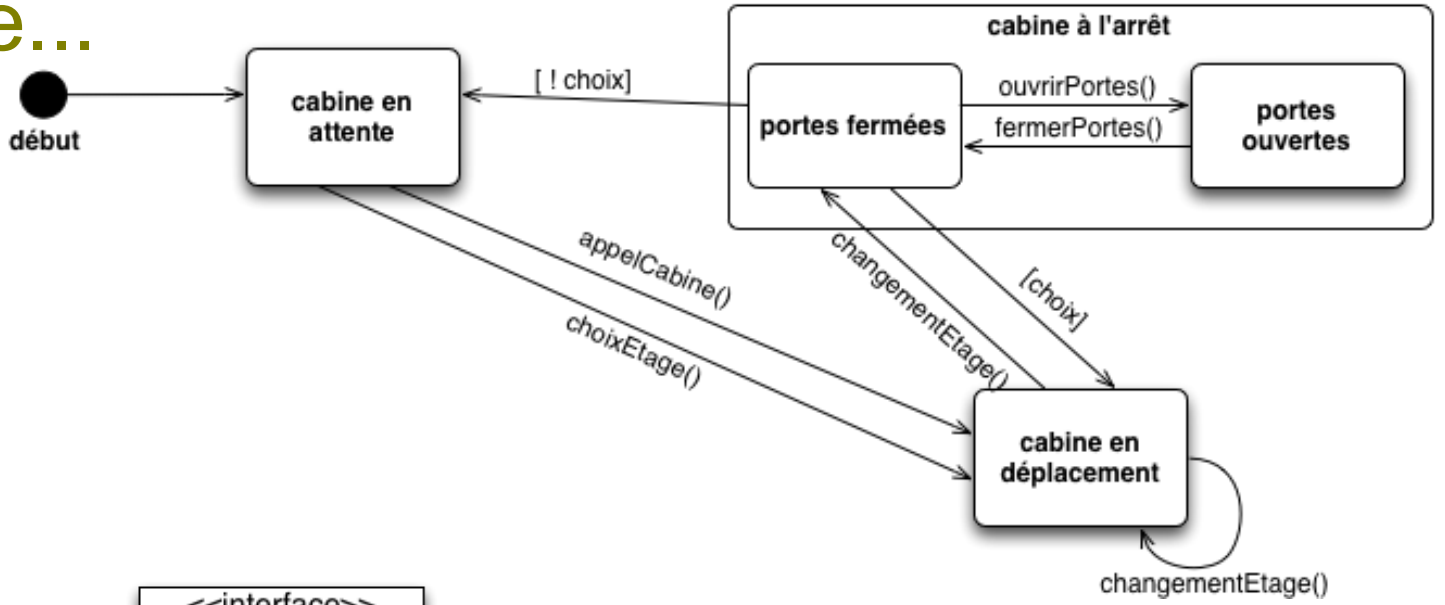


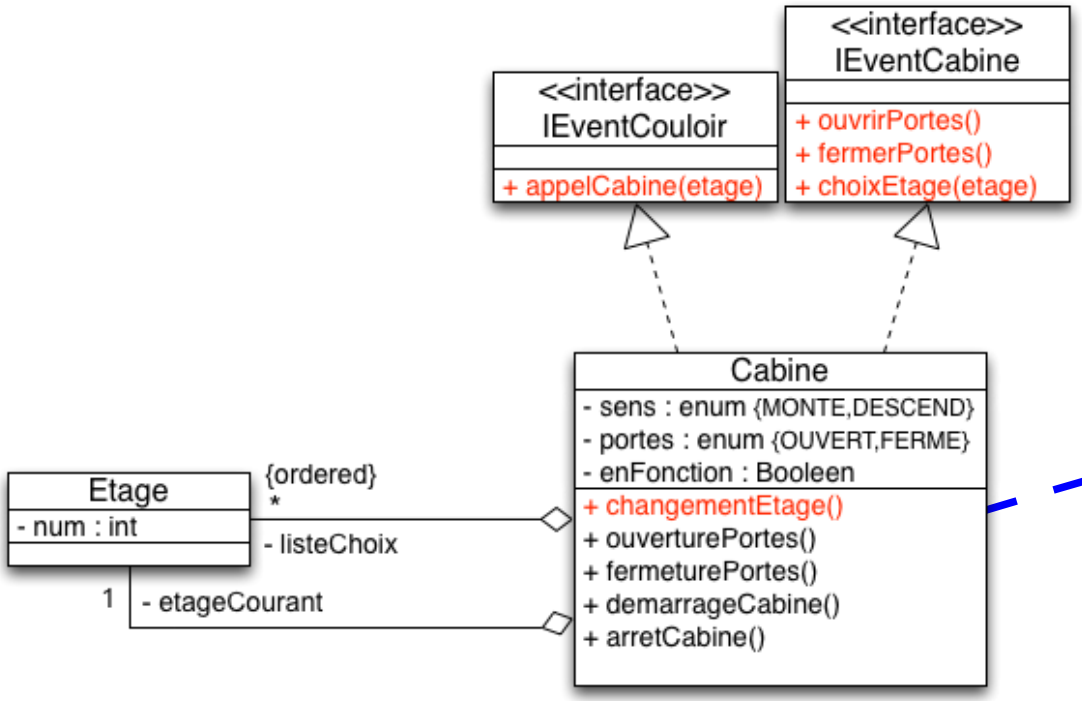

```
public class Cabine implements IeventCouloir,
IeventCabine{

        private enum sens {MONTE,DESCEND};
        private enum portes {OUVERT,FERME};
        private Boolean enFonction;
        private A                                  Choix;
        private E
```

Our job !

```
        public void appelCabine(Etage e){/*code*/};
        public void ouvrirPortes(){/*code*/};
        public void fermerPortes(){/*code*/};
        public void choixEtage(Etage e){/*code*/};
        public void changementEtage(){/*code*/};

        public void ouverturePortes(){/*code*/};
        public void fermeturePortes(){/*code*/};
        public void demarrageCabine(){/*code*/};
        public void arretCabine{/*code*/};
}
```

Un peu de code...





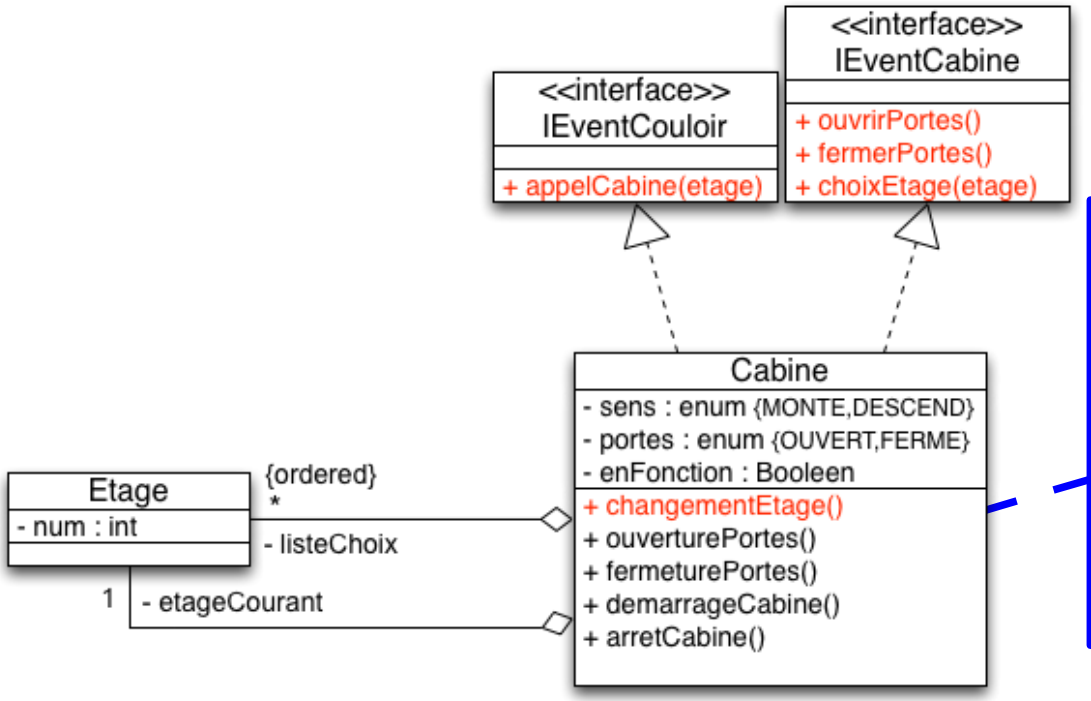

```
public class Cabine...{
        ...
        public void ouvrirPortes(){
          If(!enFonction && portes==FERME)
             this.ouverturePortes() ;
          } ;
        ...
}
```

Notons

Cette manière de procéder nécessite, pour chaque événement, de retrouver l'état de la cabine à partir de son contexte (propriétés) et d'appliquer la règle de gestion qui convient :

→ **difficile à lire et à maintenir**

Par exemple, l'ajout d'un état « en maintenance » nécessitera de redéfinir la plupart des opérations.

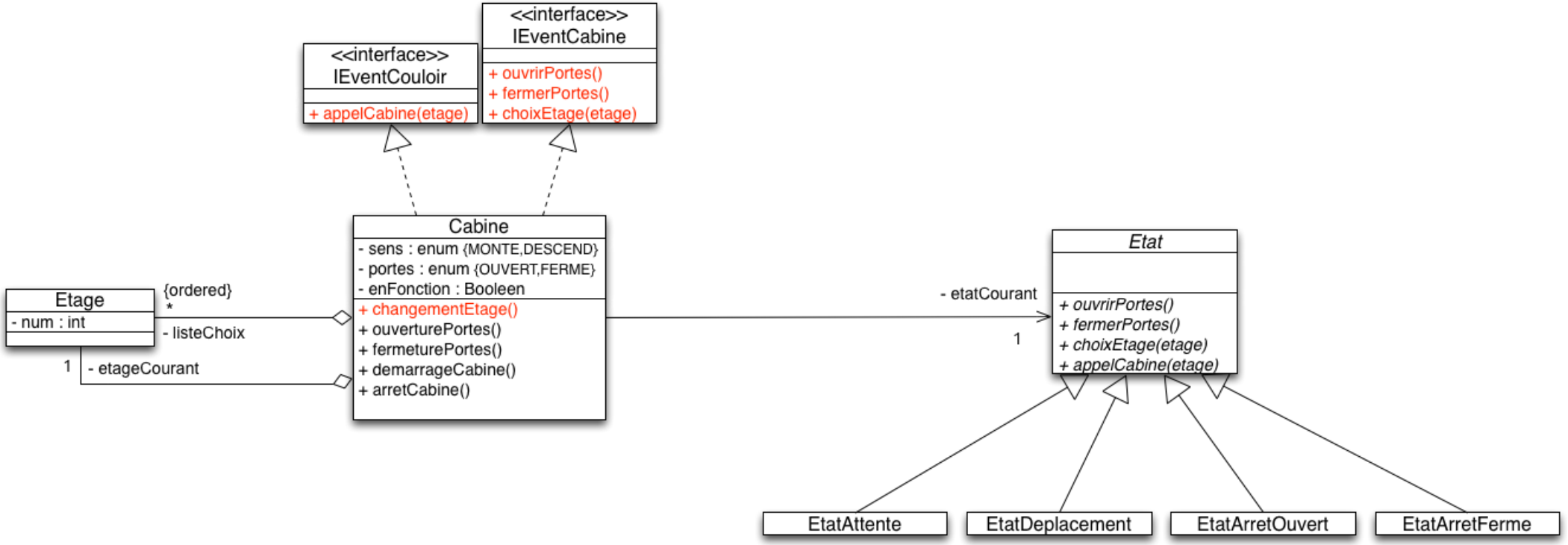



```
public class Cabine...{
        ...
        public void ouvrirPortes(){
          If(!enFonction && portes==FERME)
             this.ouverturePortes() ;
          } ;
        ...
}
```

Question

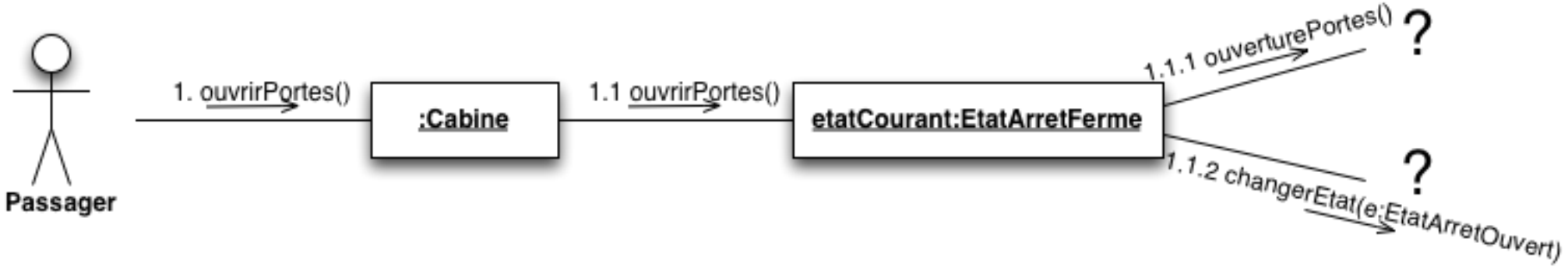
Comment organiser le code de manière à :

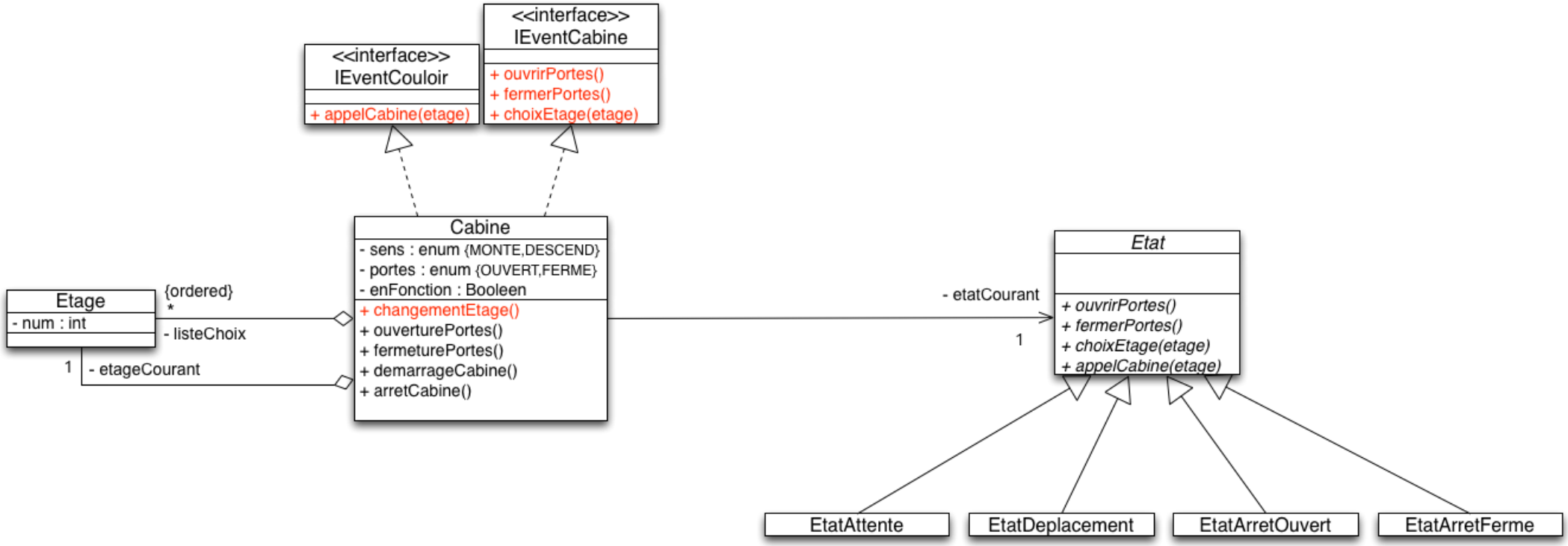
- expliciter les états possibles et l'**état courant de la cabine**
- lui faire adopter un comportement différent selon son état ?



Question

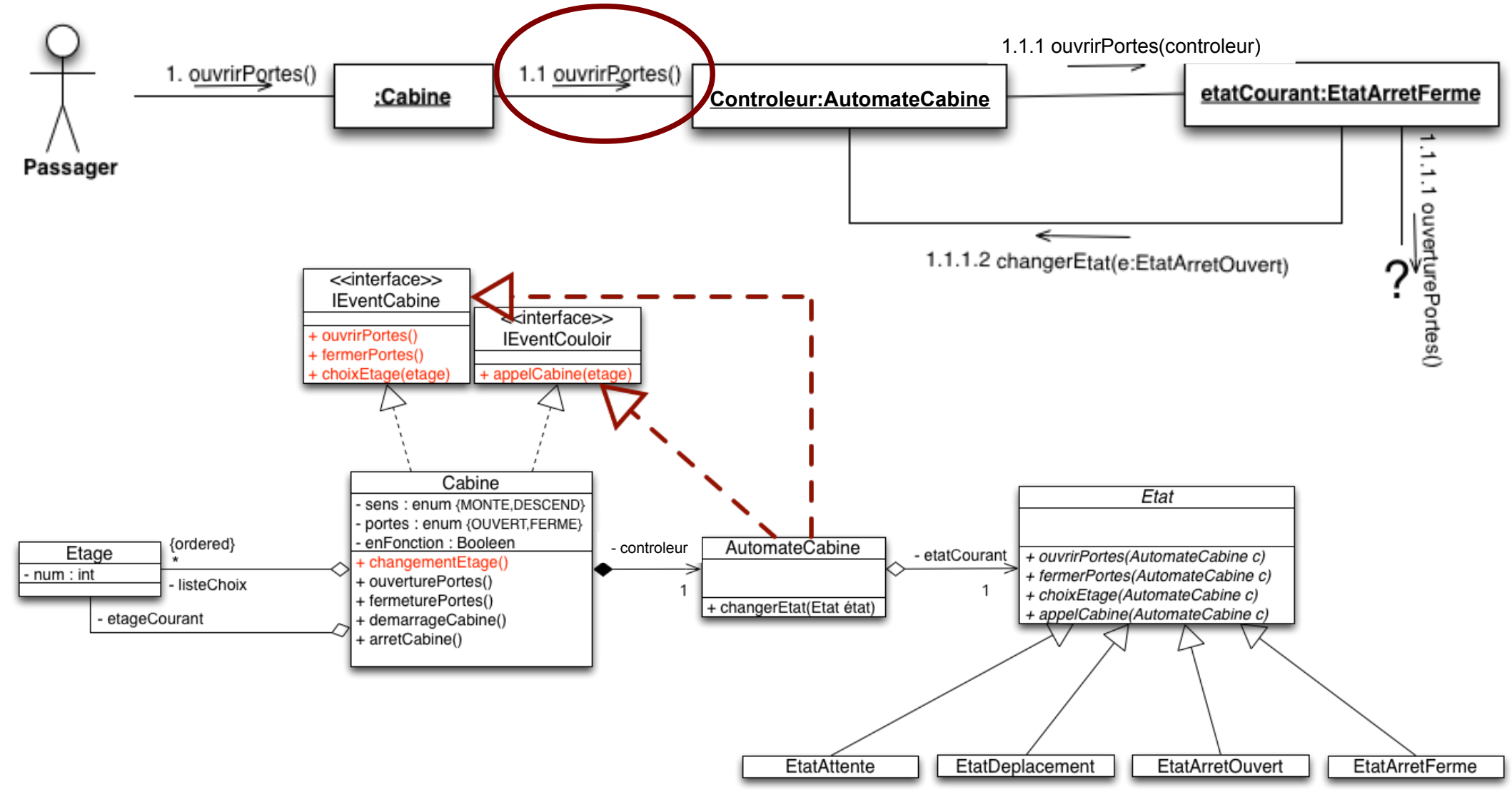
Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées





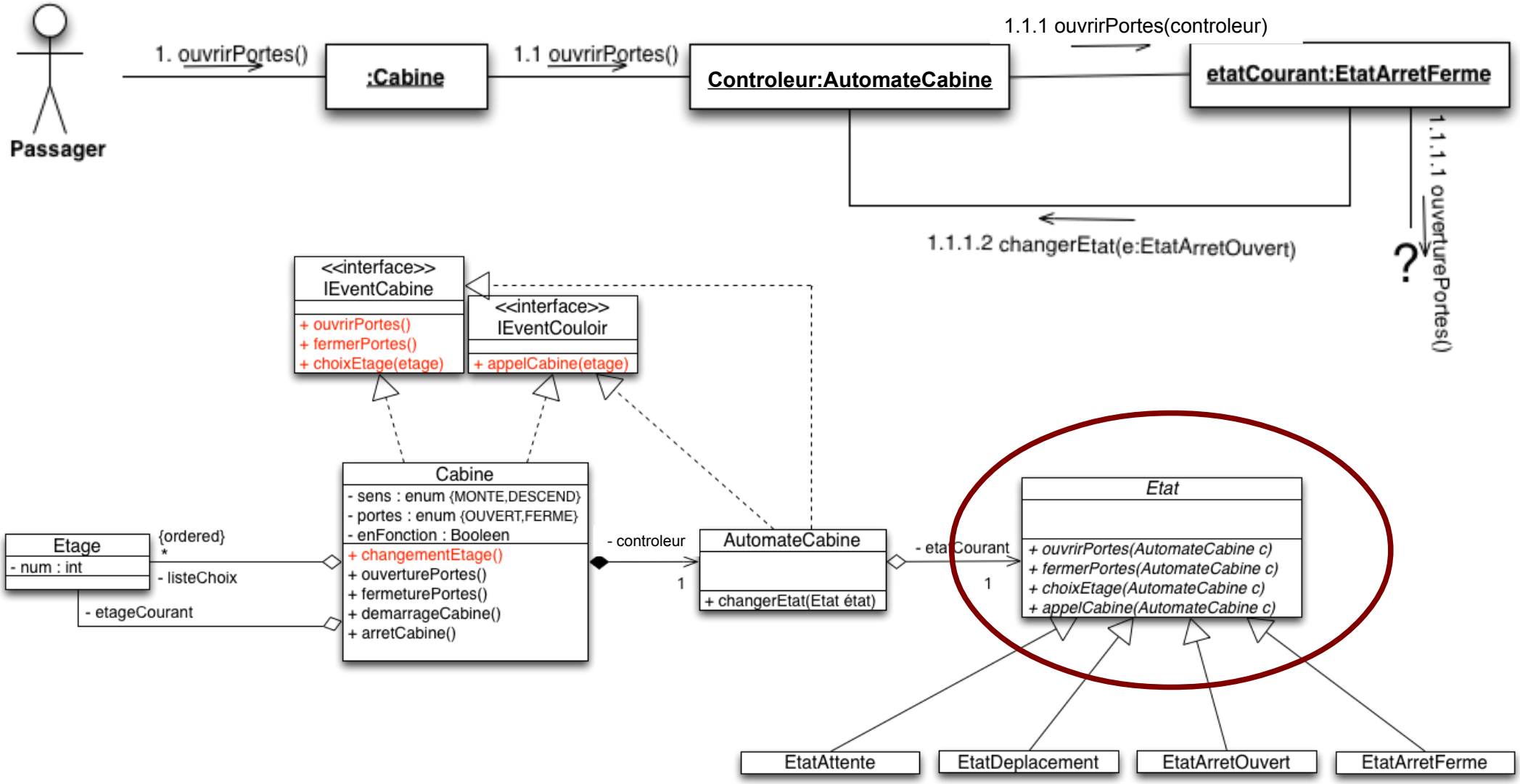
Question

Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées



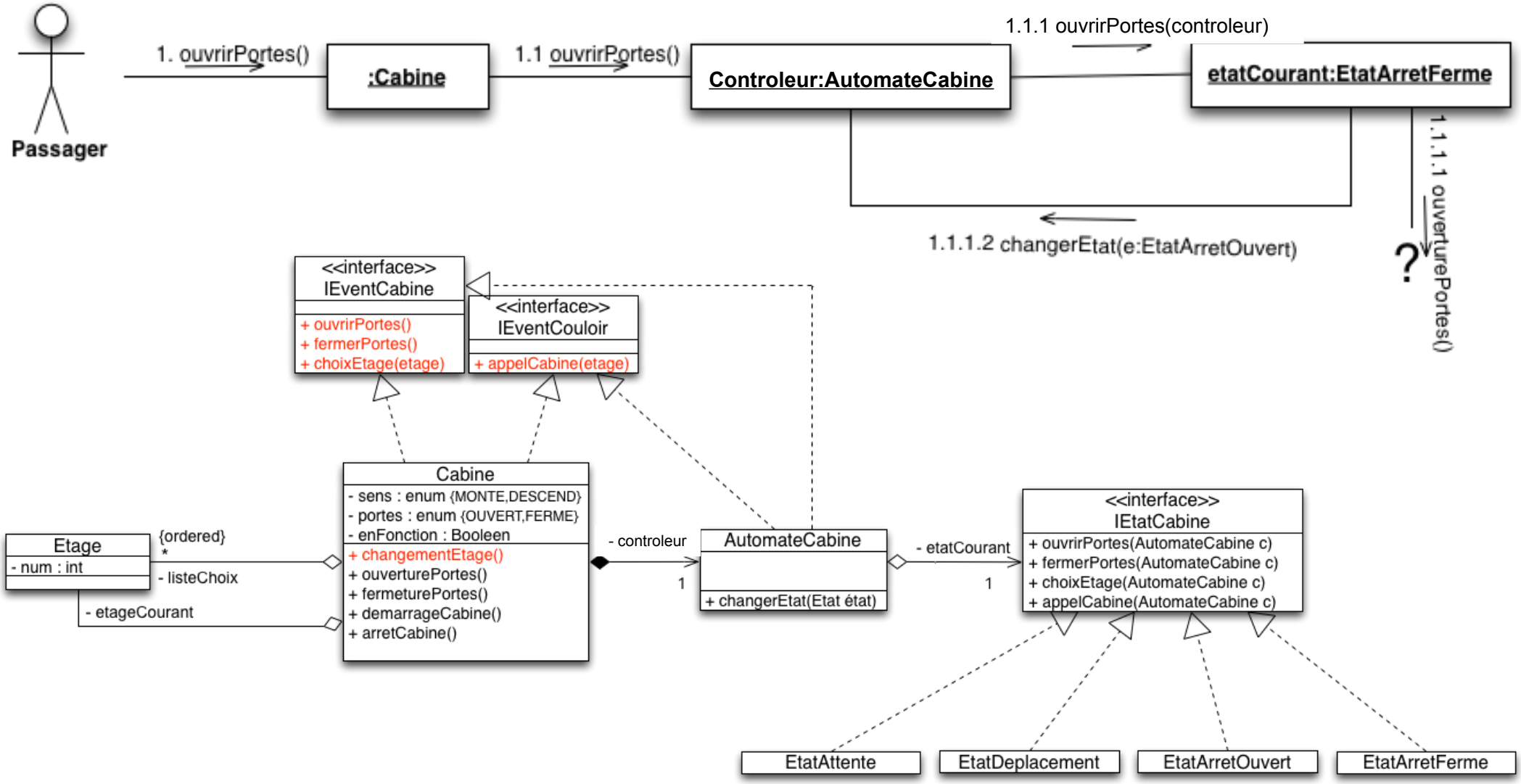
Question

Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées



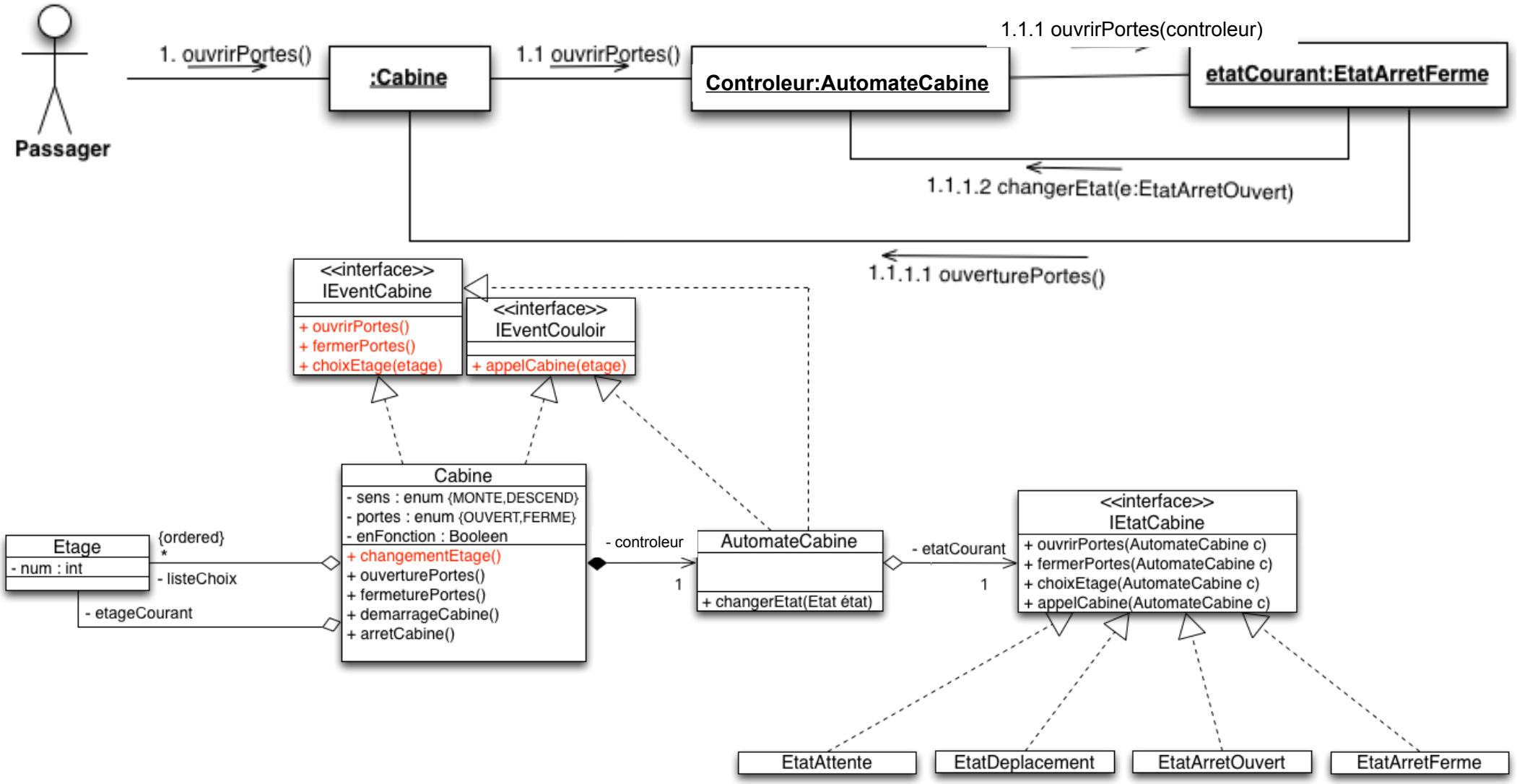
Question

Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées



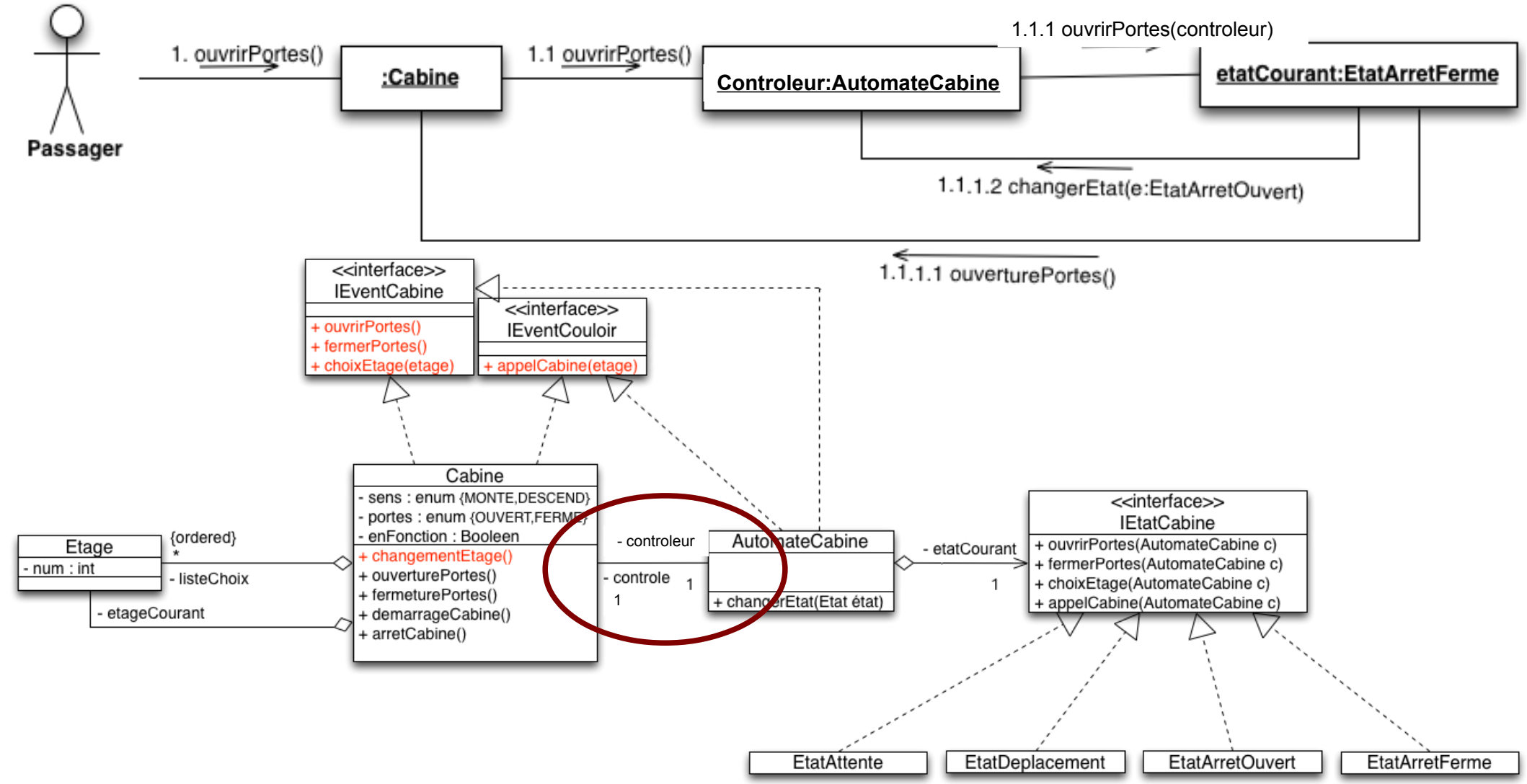
Question

Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées

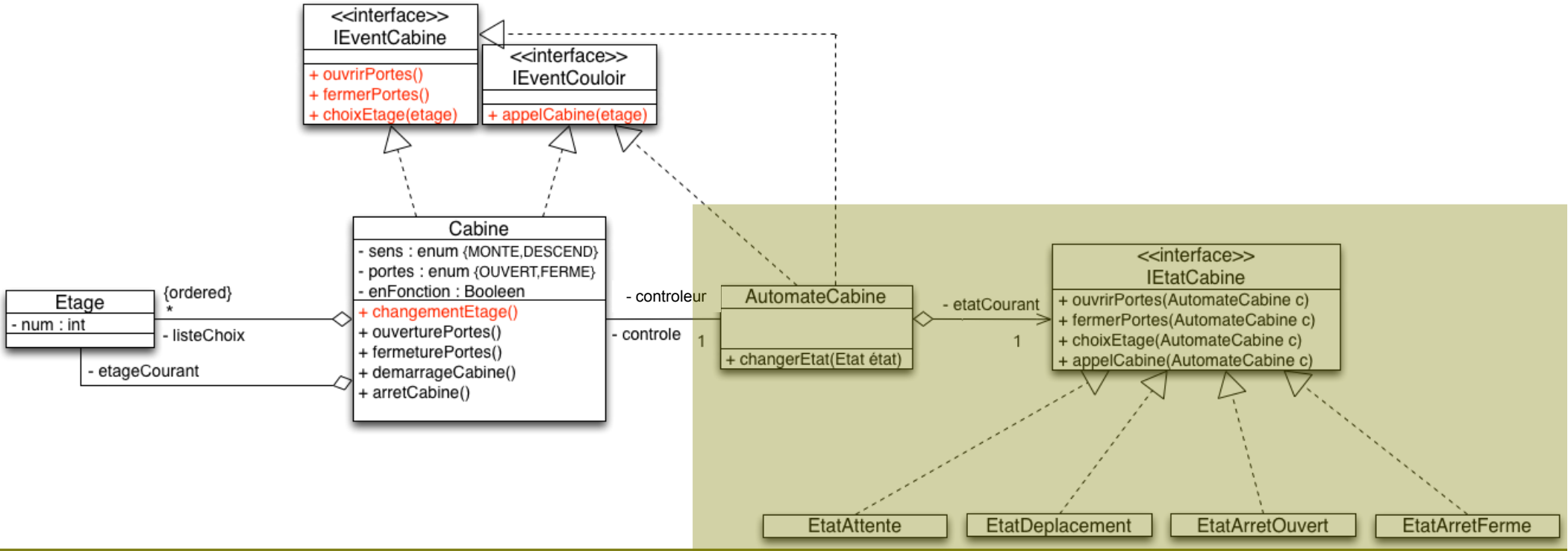


Question

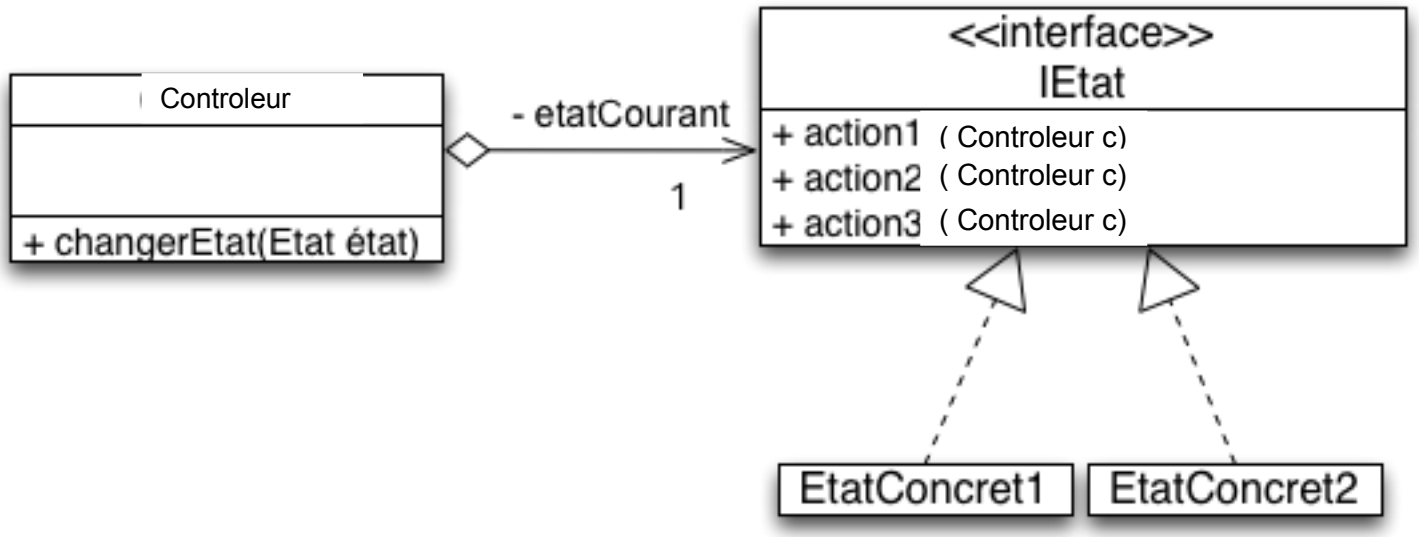
Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées



Design pattern « Etat » (state)



Design pattern « Etat » (state)



Questions ?