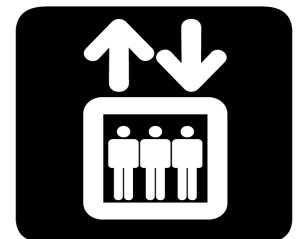
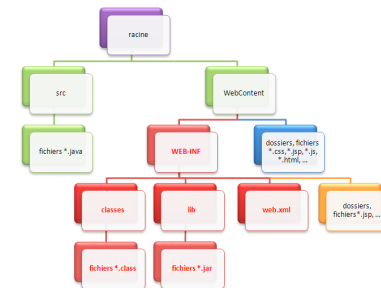


Retour sur les notions déjà étudiées

- 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**

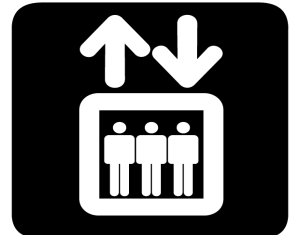


Diagramme d'états-transitions & Patron de conception « état »

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** 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.

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.



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

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



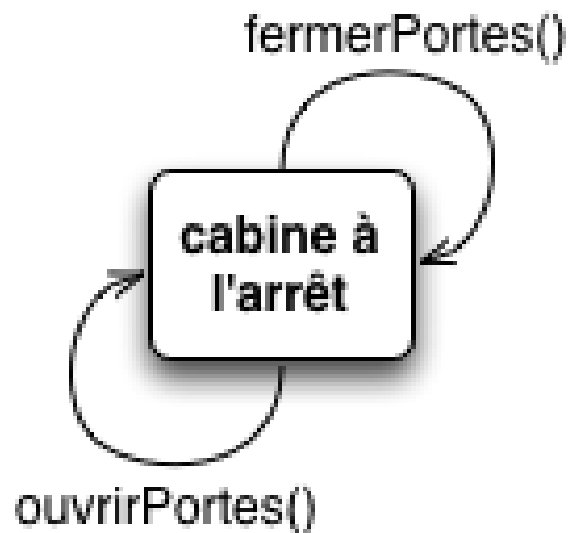
Question

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

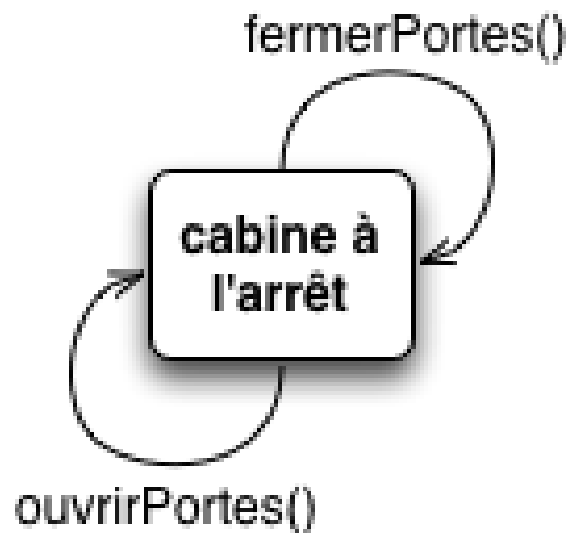
A modéliser

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



A modéliser

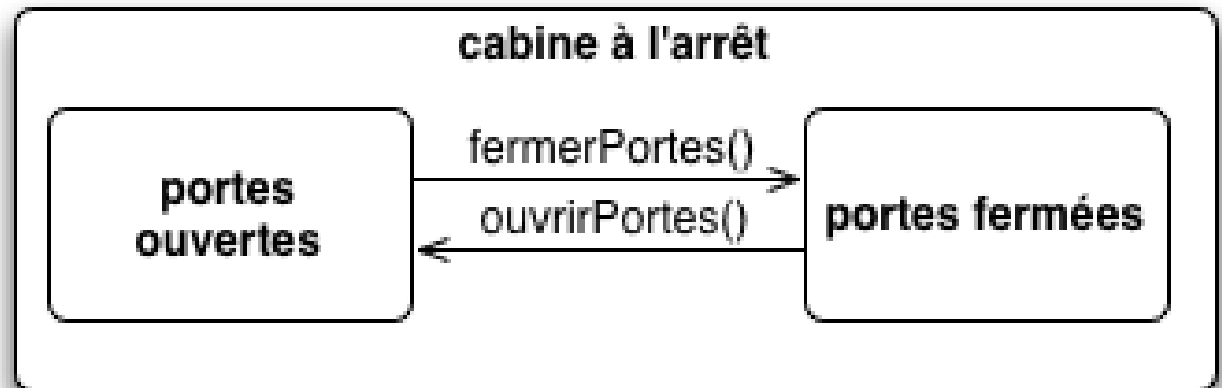
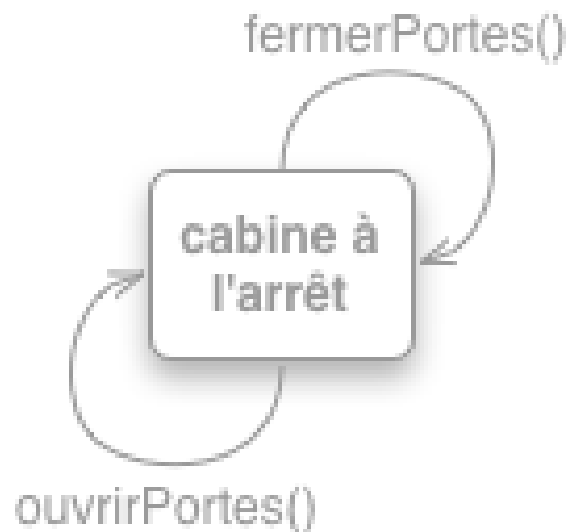
L'ouverture et la fermeture des portes ne peuvent être réalisées que lorsque la cabine est à l'arrêt



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

A modéliser

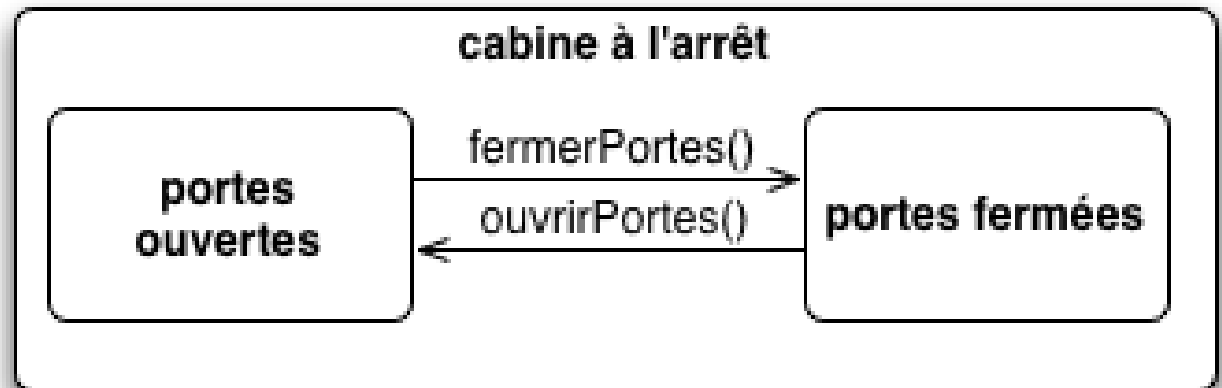
L'ouverture et la fermeture des portes ne peuvent être réalisées que lorsque la cabine est à l'arrêt



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

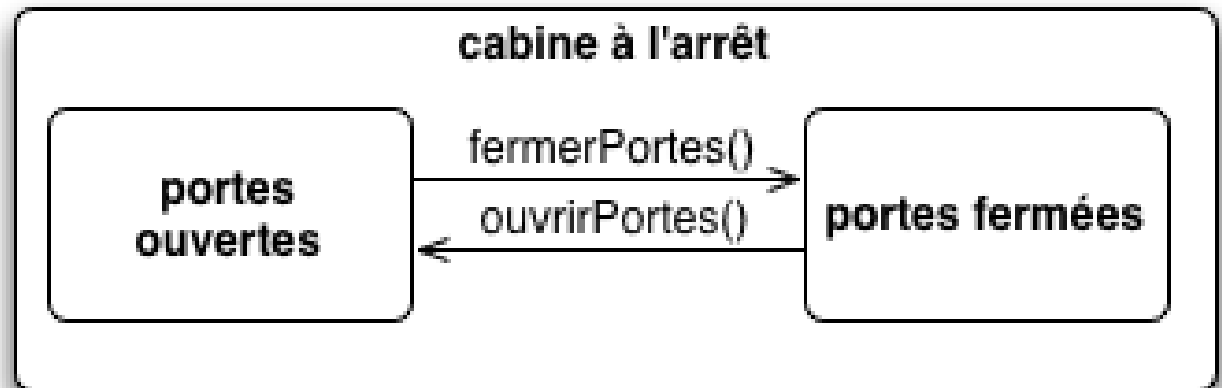
A modéliser

*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*



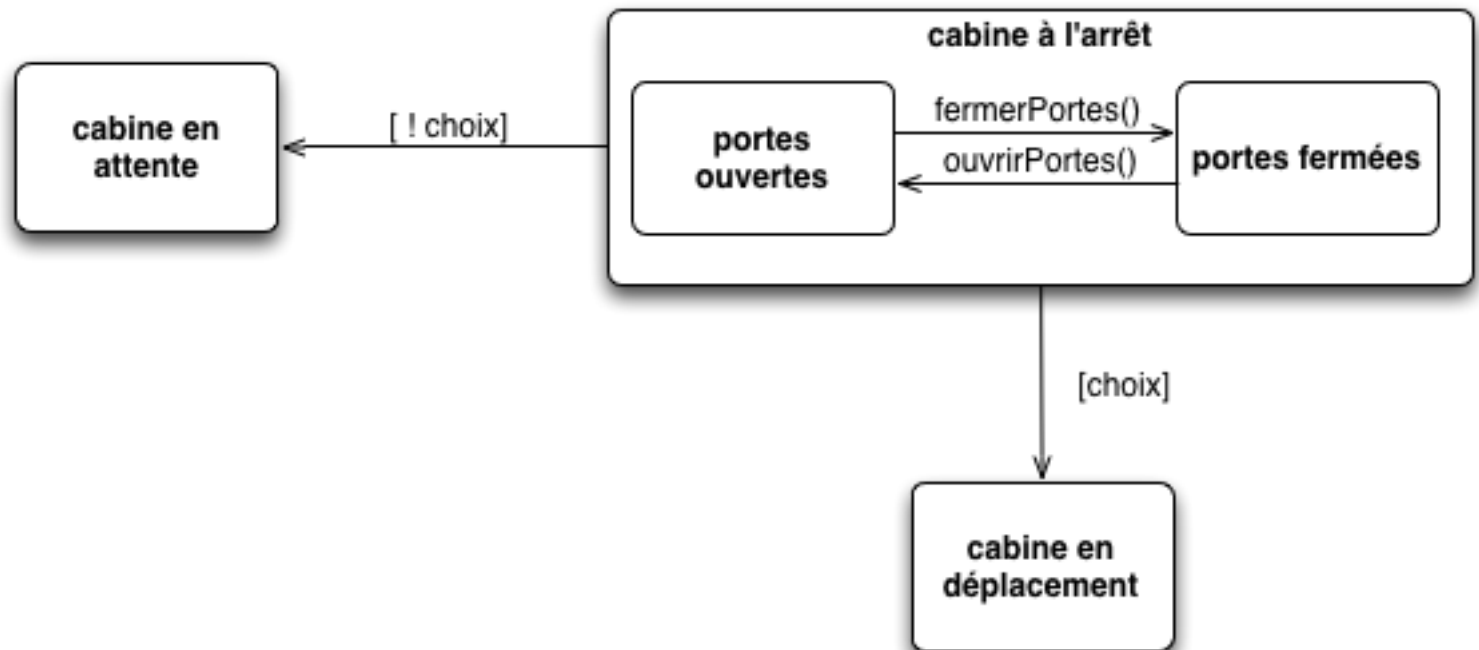
A modéliser

*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***



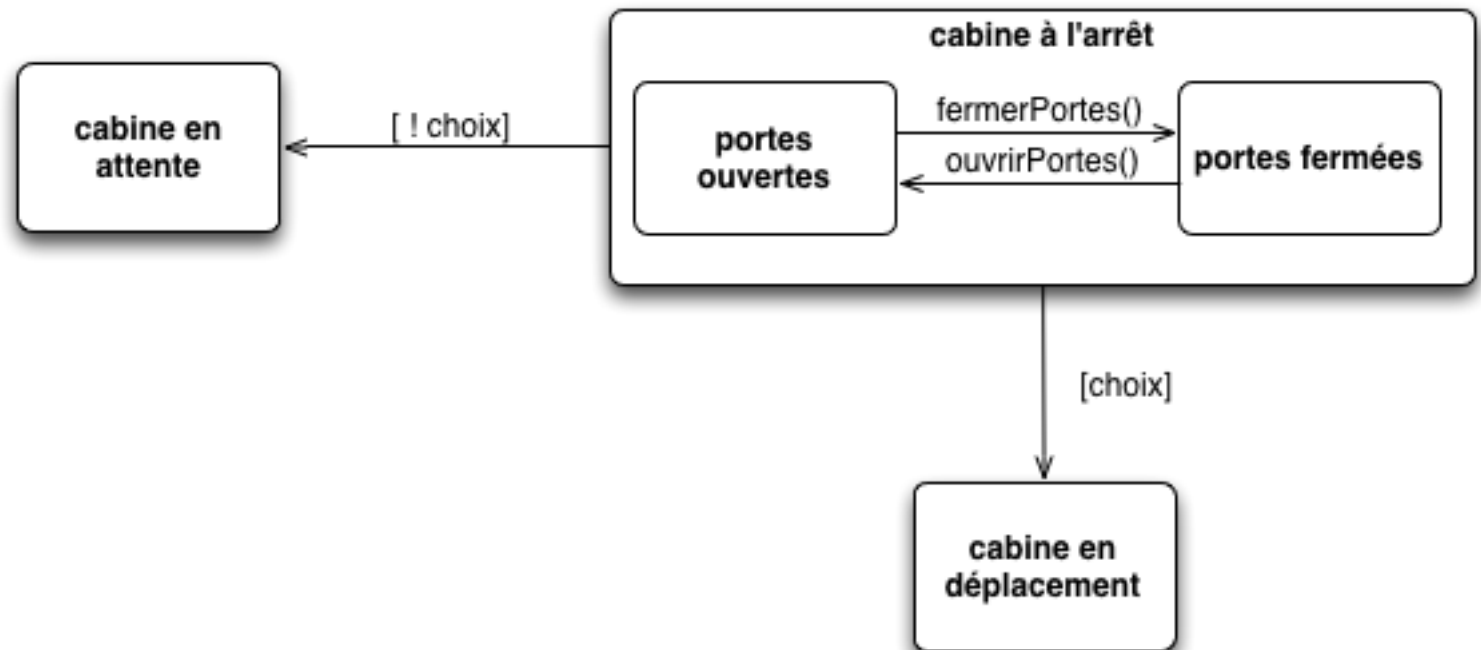
A modéliser

*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***



A modéliser

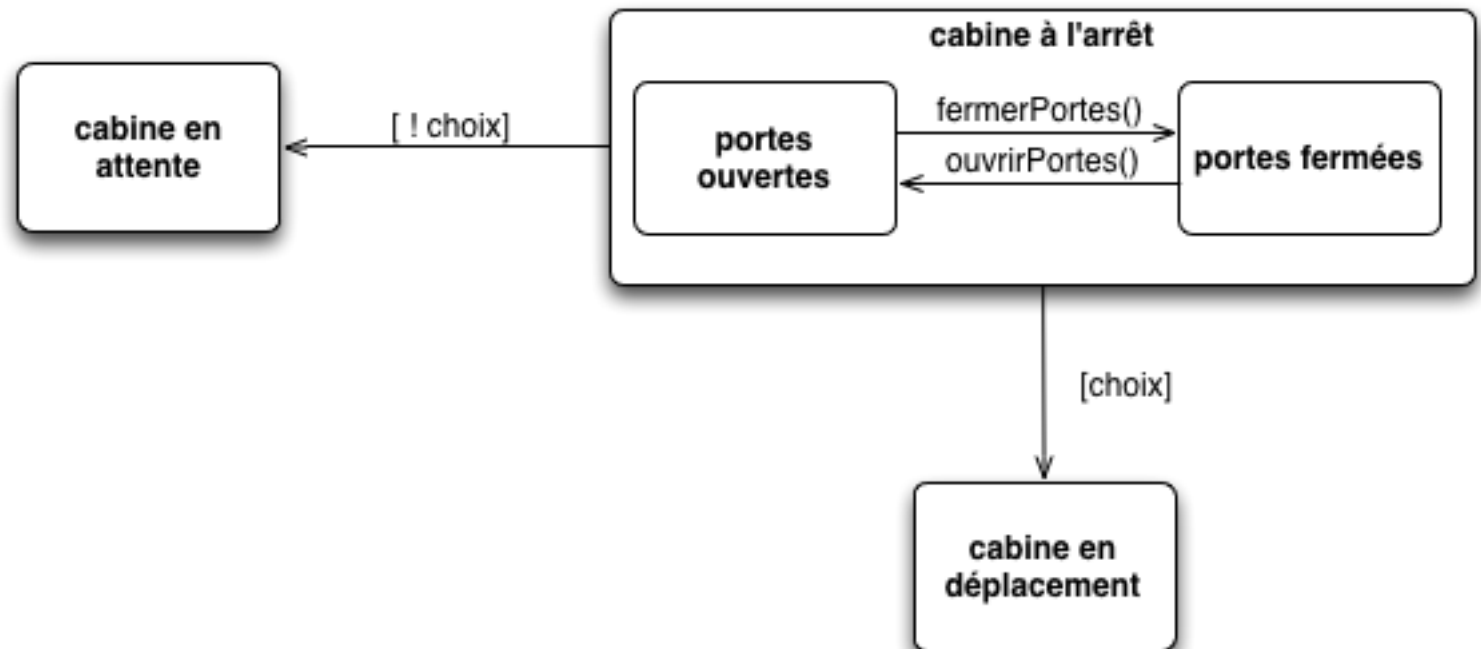
*Si aucun choix d'étage n'est enregistré, une cabine à l'arrêt et portes fermées se met **en attente** (lumière éteinte) sinon elle se met **en déplacement***



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

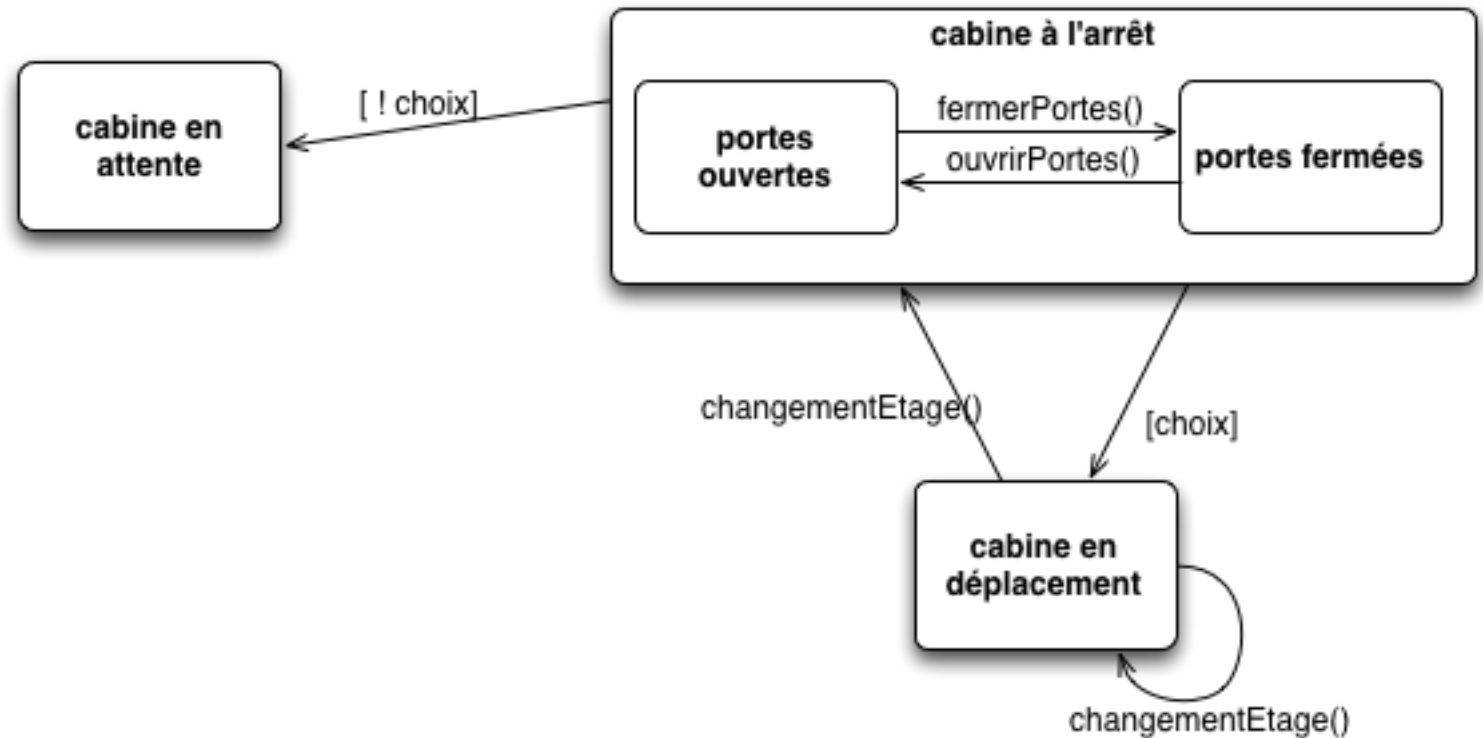
A modéliser

Une cabine en déplacement s'arrête dès qu'elle a atteint l'étage choisi



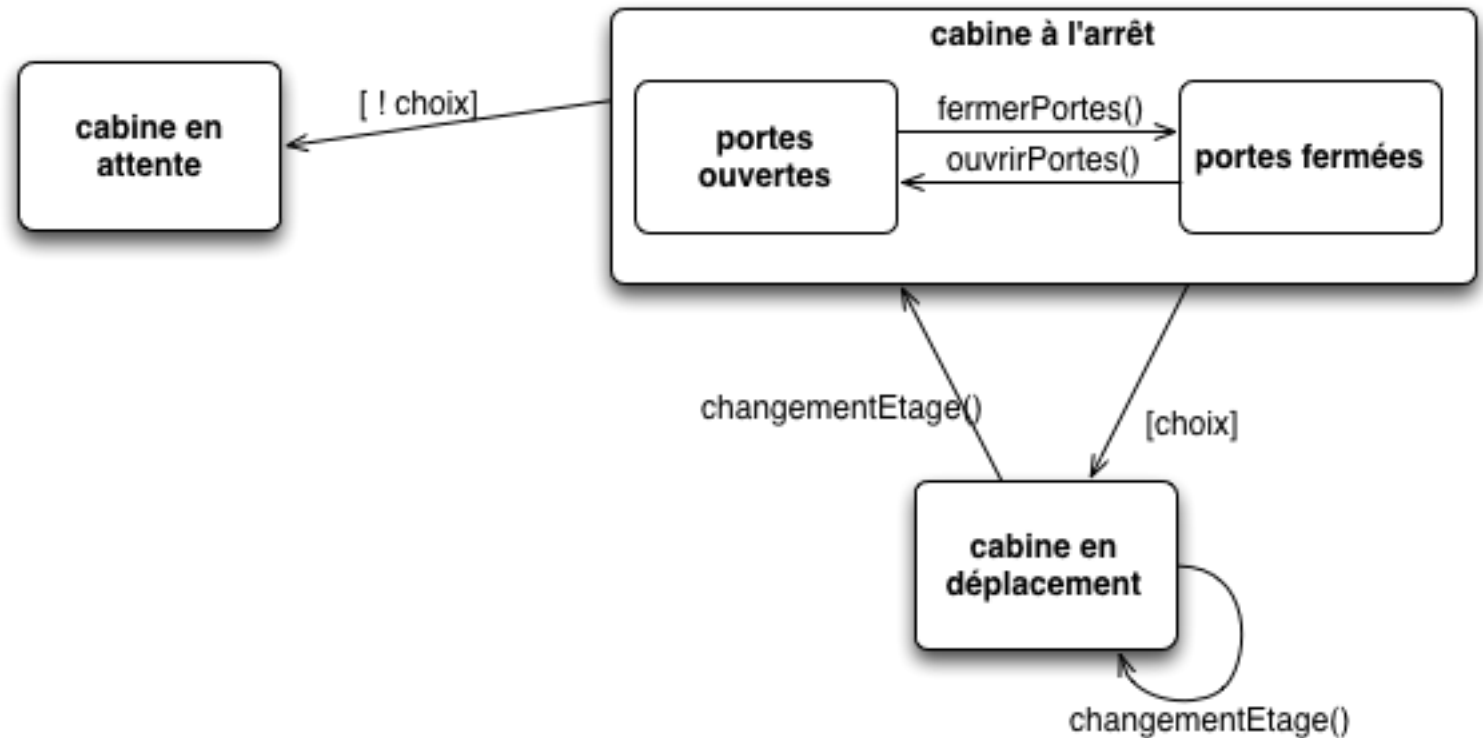
A modéliser

Une cabine en déplacement s'arrête dès qu'elle a atteint l'étage choisi



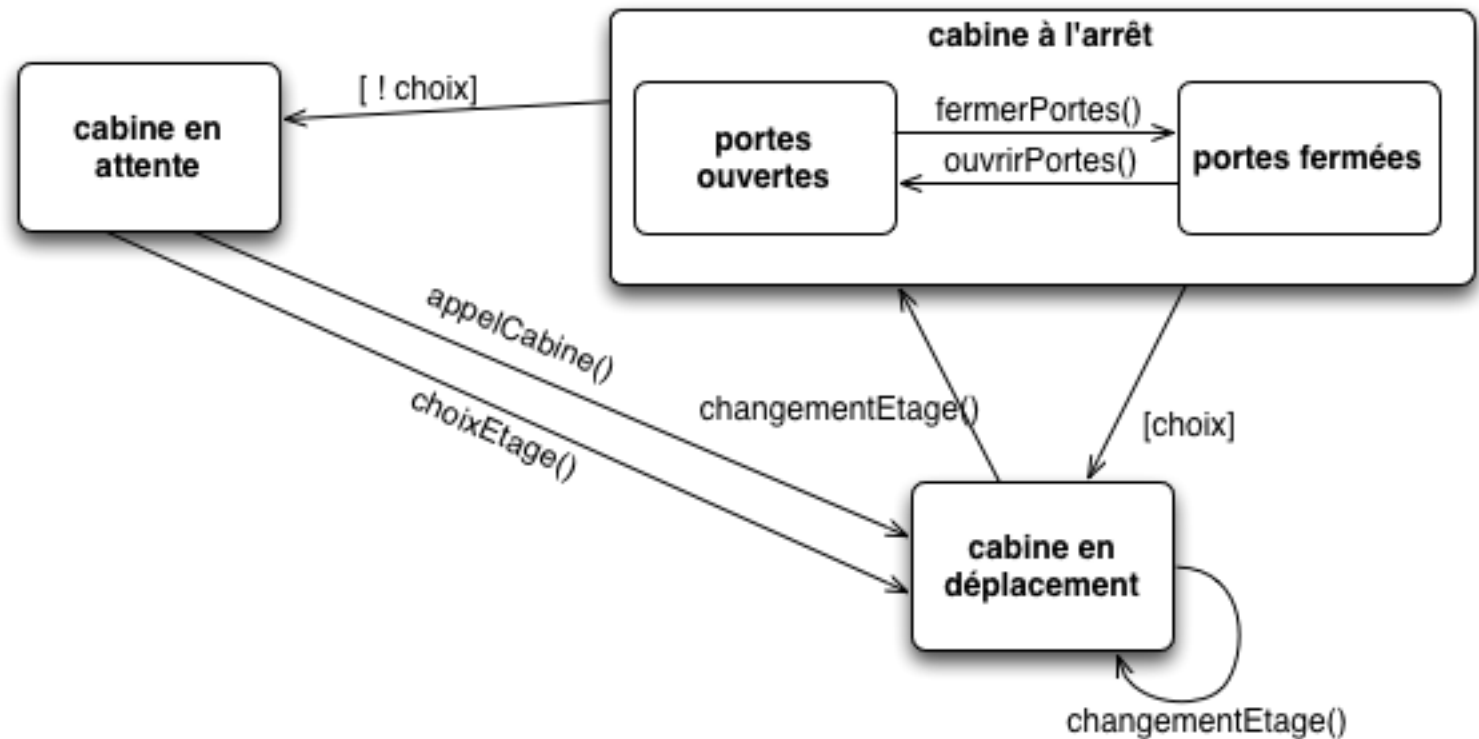
A modéliser

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



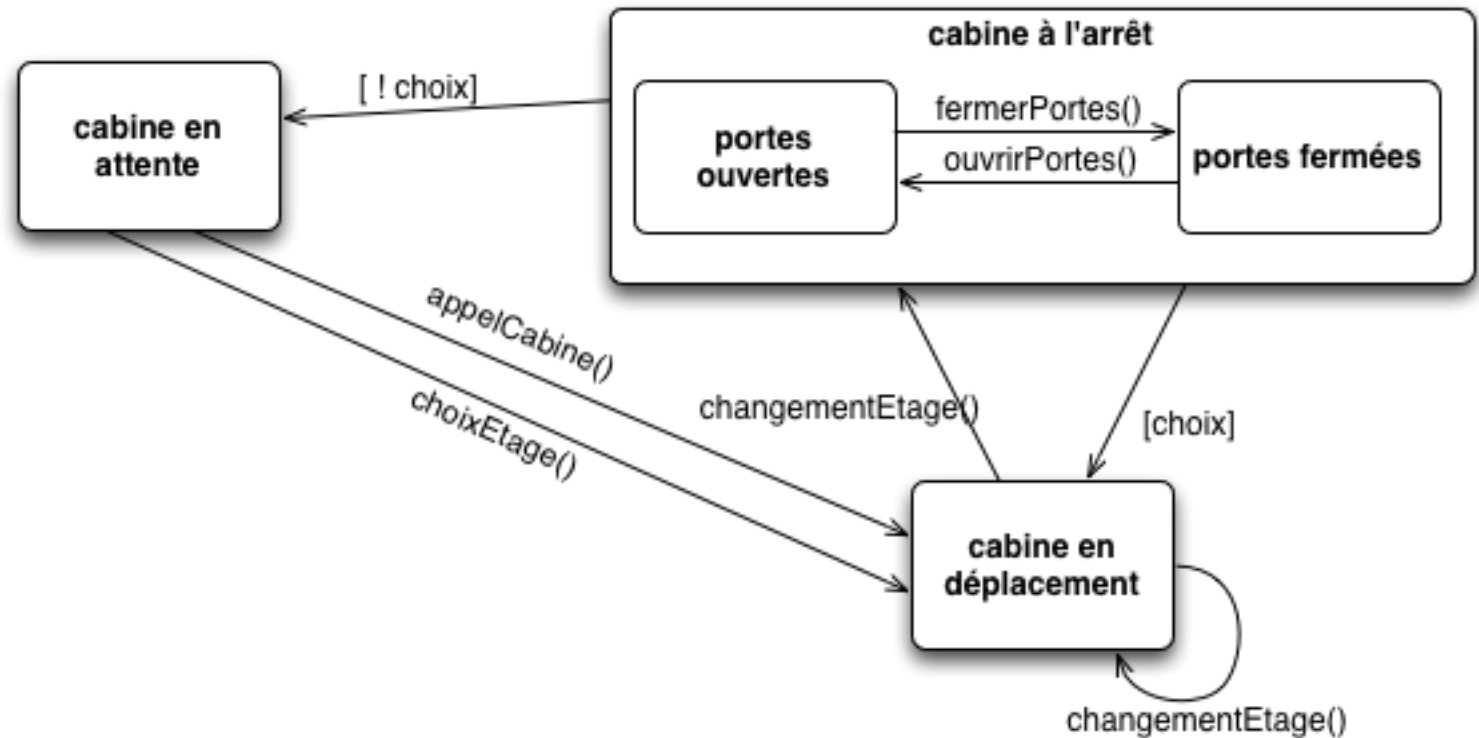
A modéliser

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



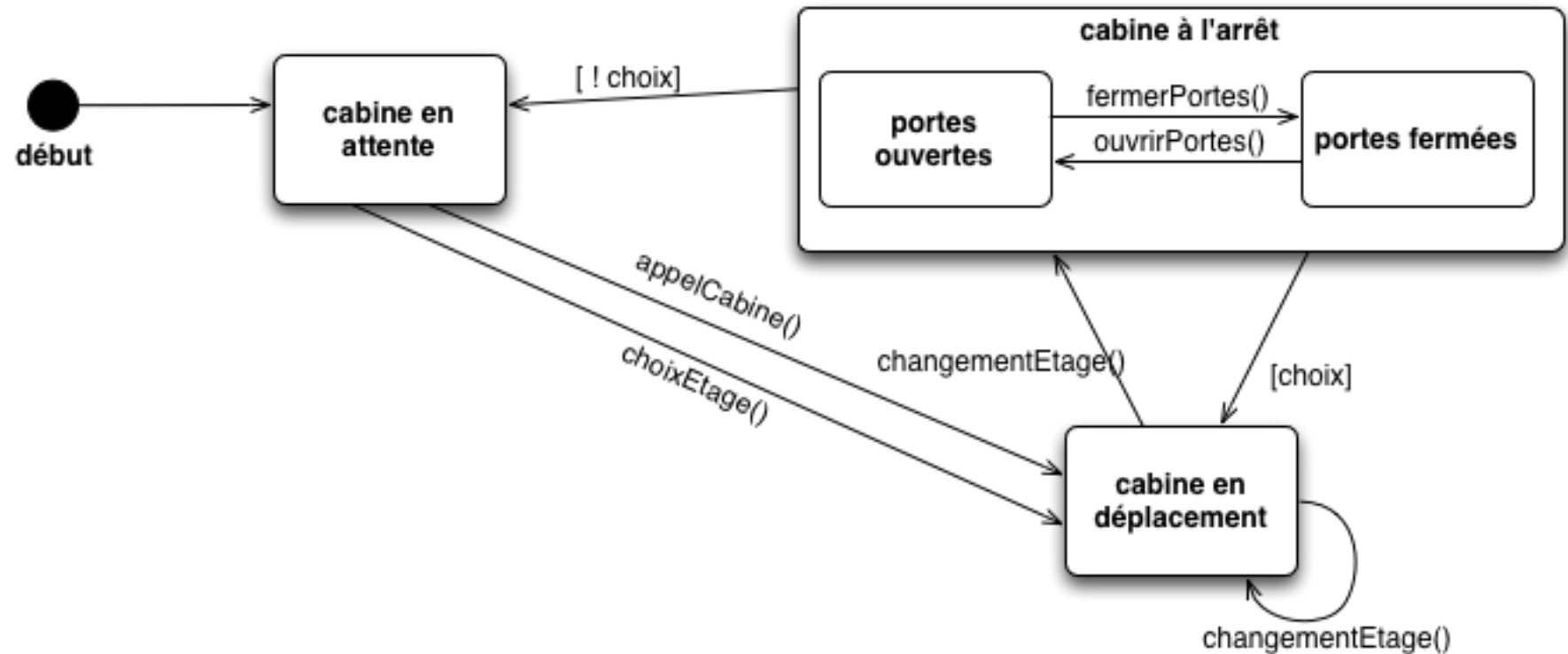
A modéliser (compléments)

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



A modéliser (compléments)

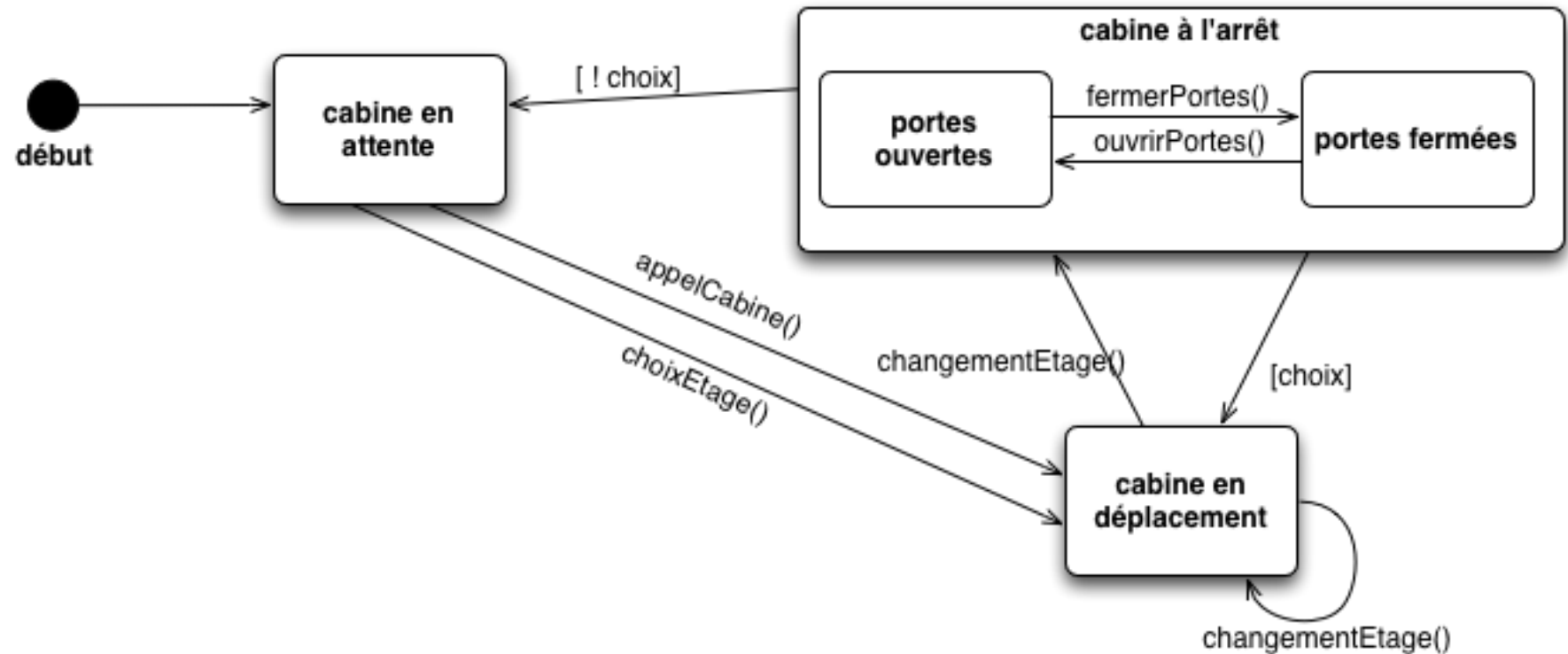
A la réinitialisation du système « ascenseur », la cabine est par défaut en attente.



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.



A modéliser (compléments)

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

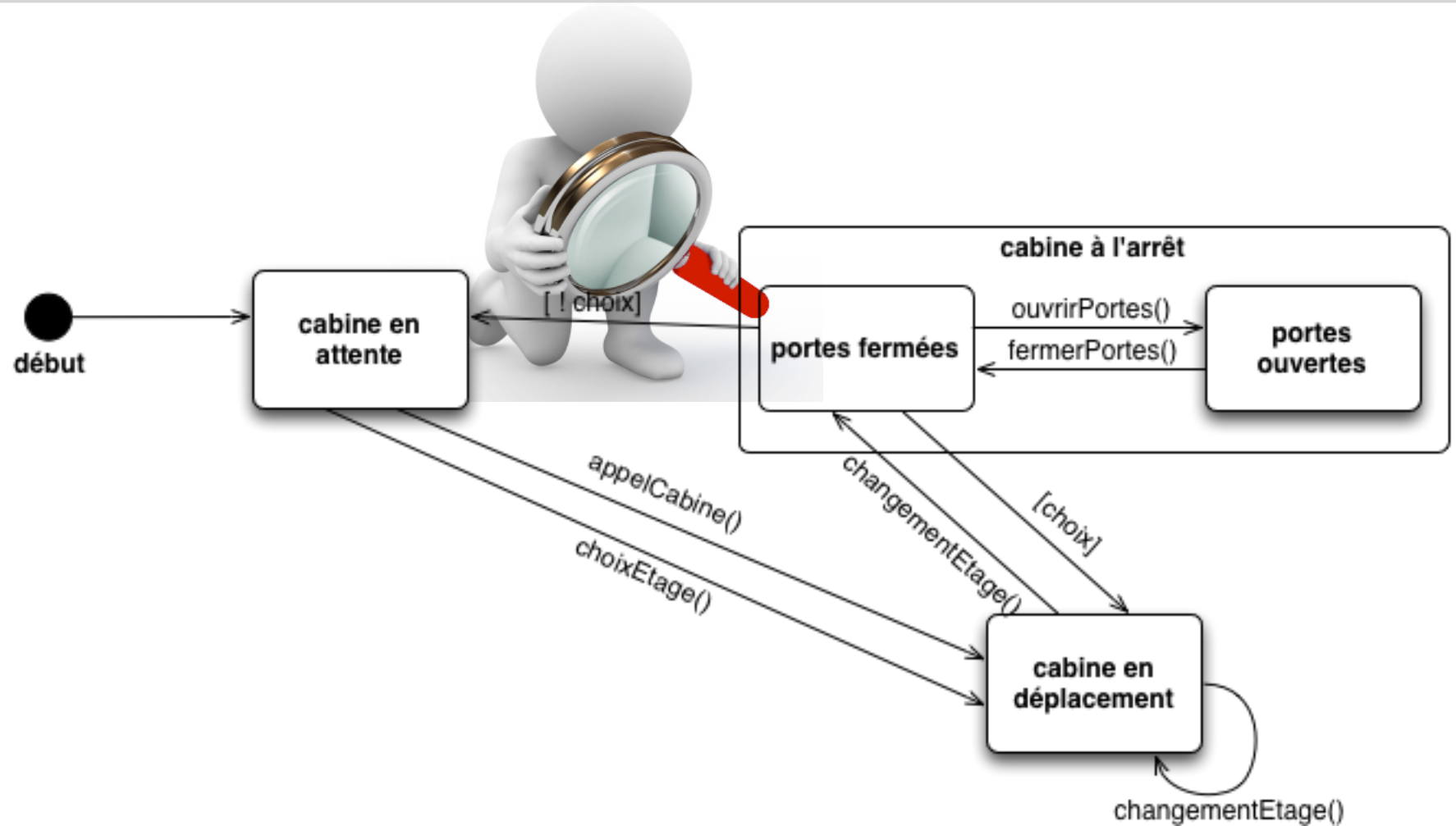


Diagramme UML d'Etats-Transition

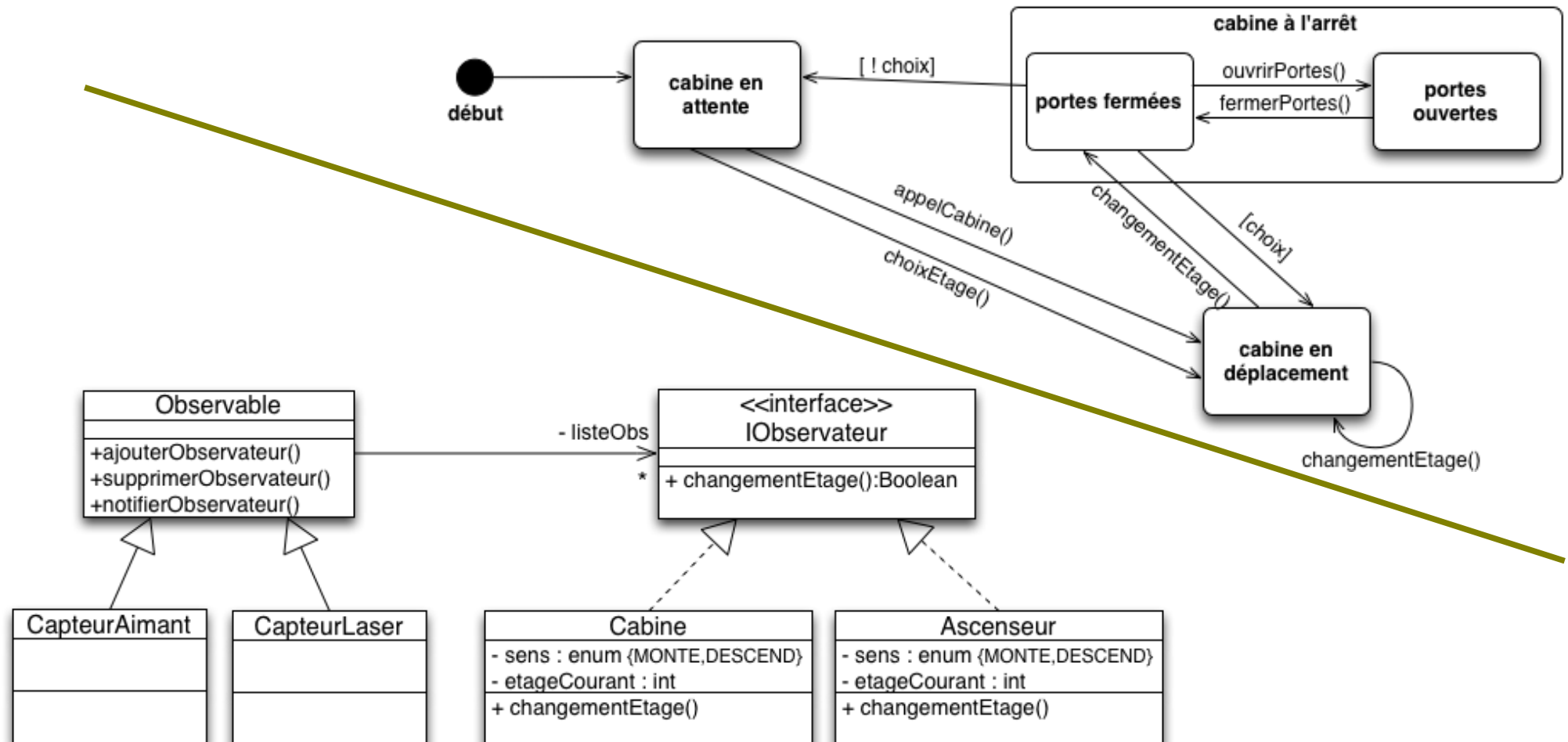


Diagramme UML de classes (temporaire)

Diagramme UML d'Etats-Transition

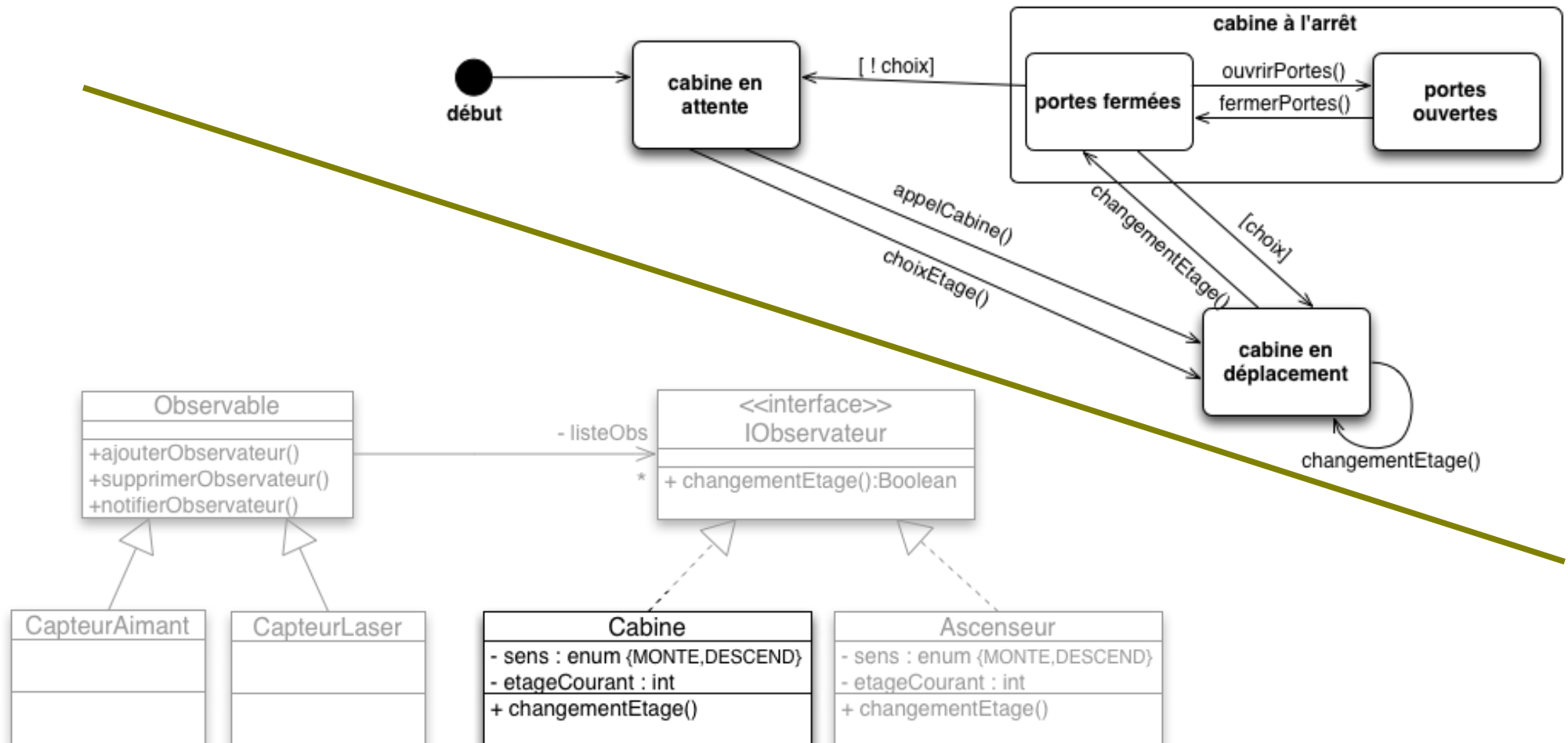
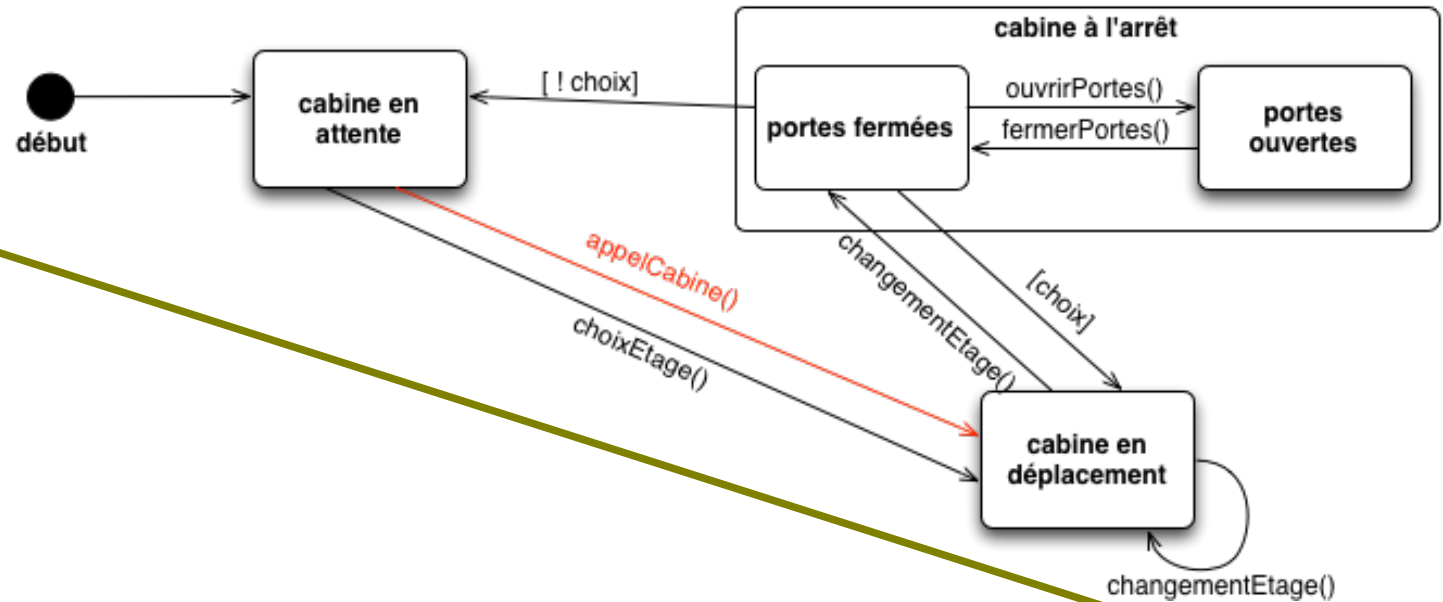


Diagramme UML de classes (temporaire)

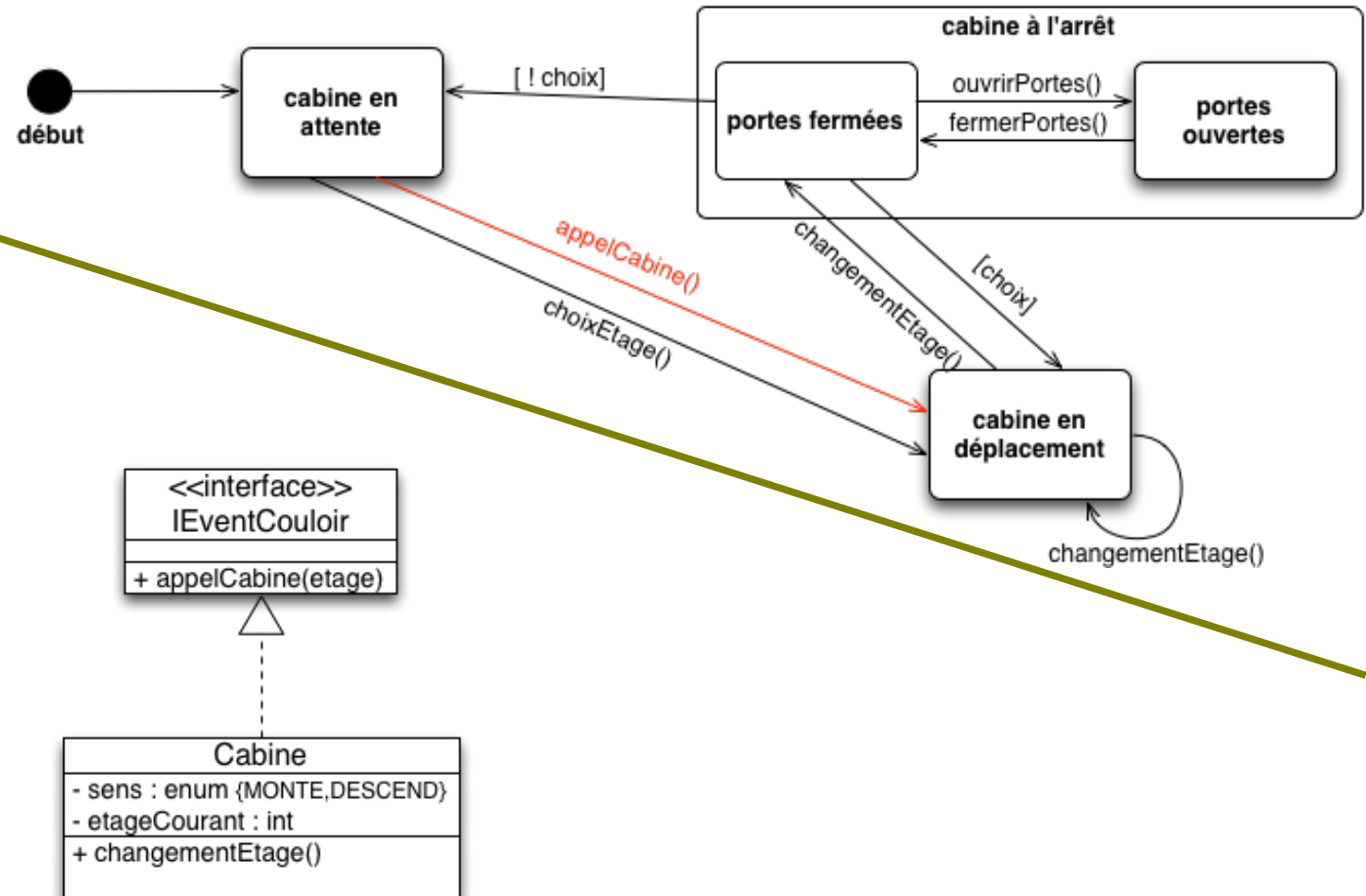


Appel de la cabine du couloir

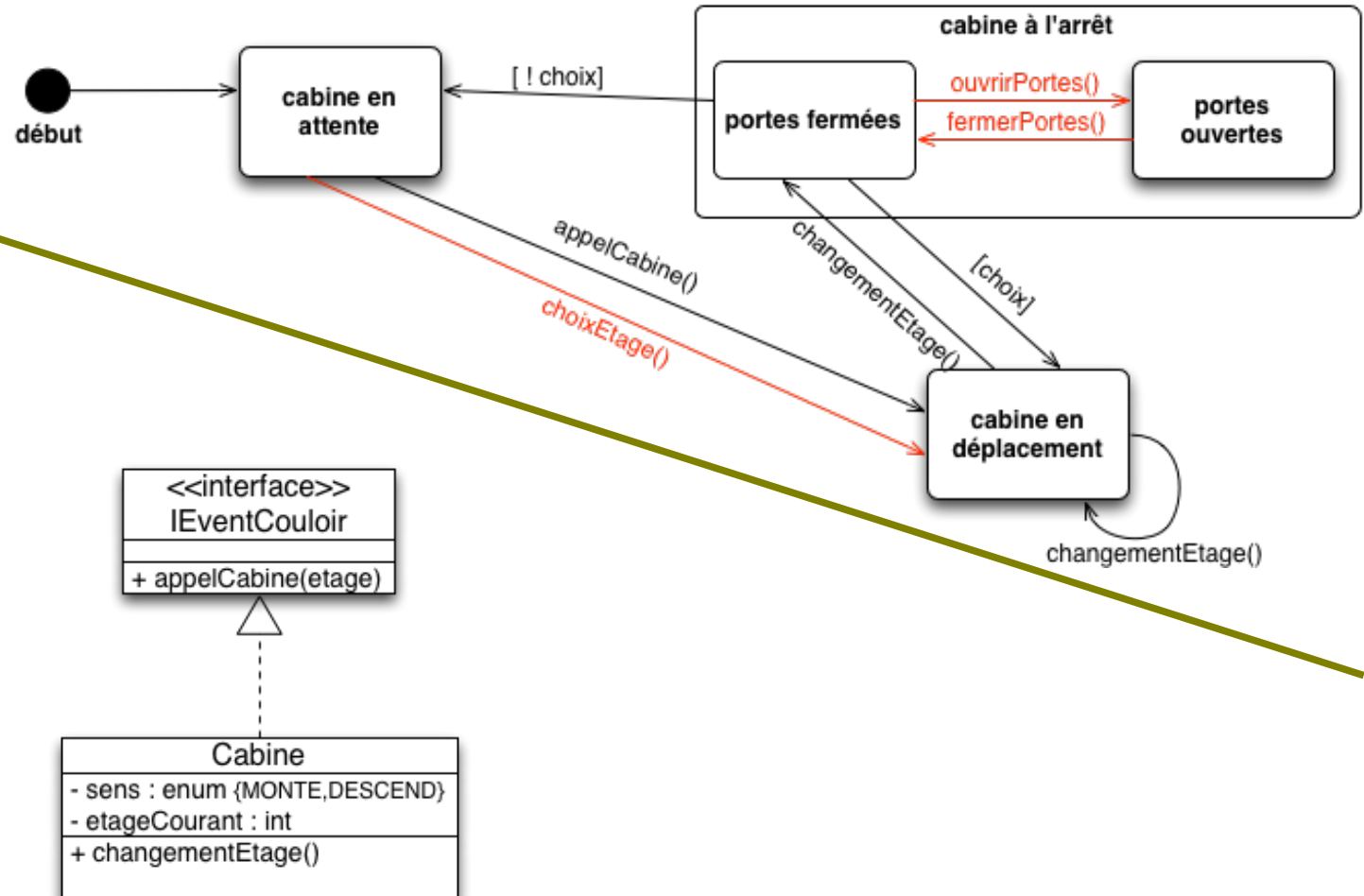


Cabine
- sens : enum {MONTE,DESCEND}
- etageCourant : int
+ changementEtage()

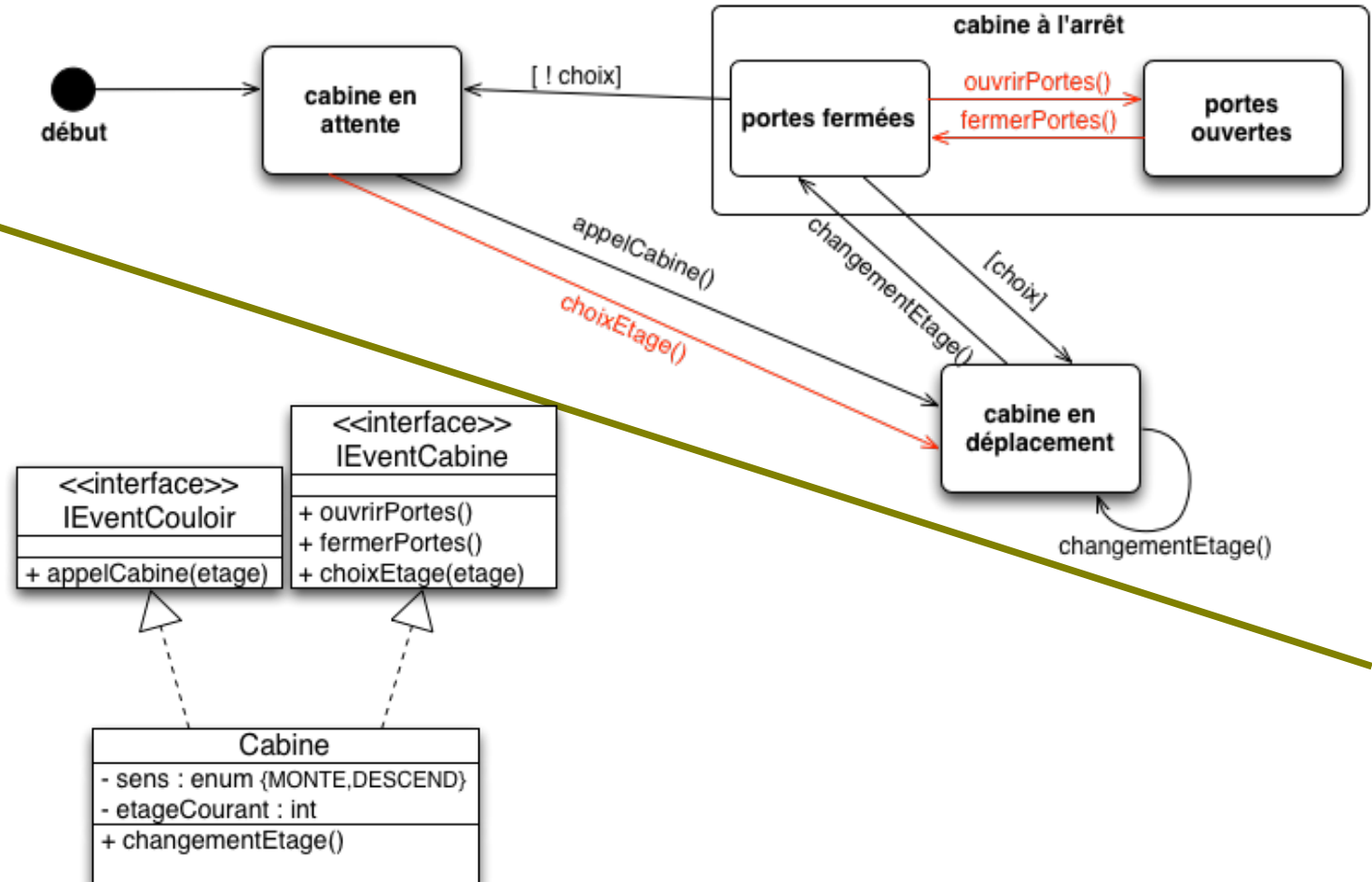
Appel de la cabine du couloir



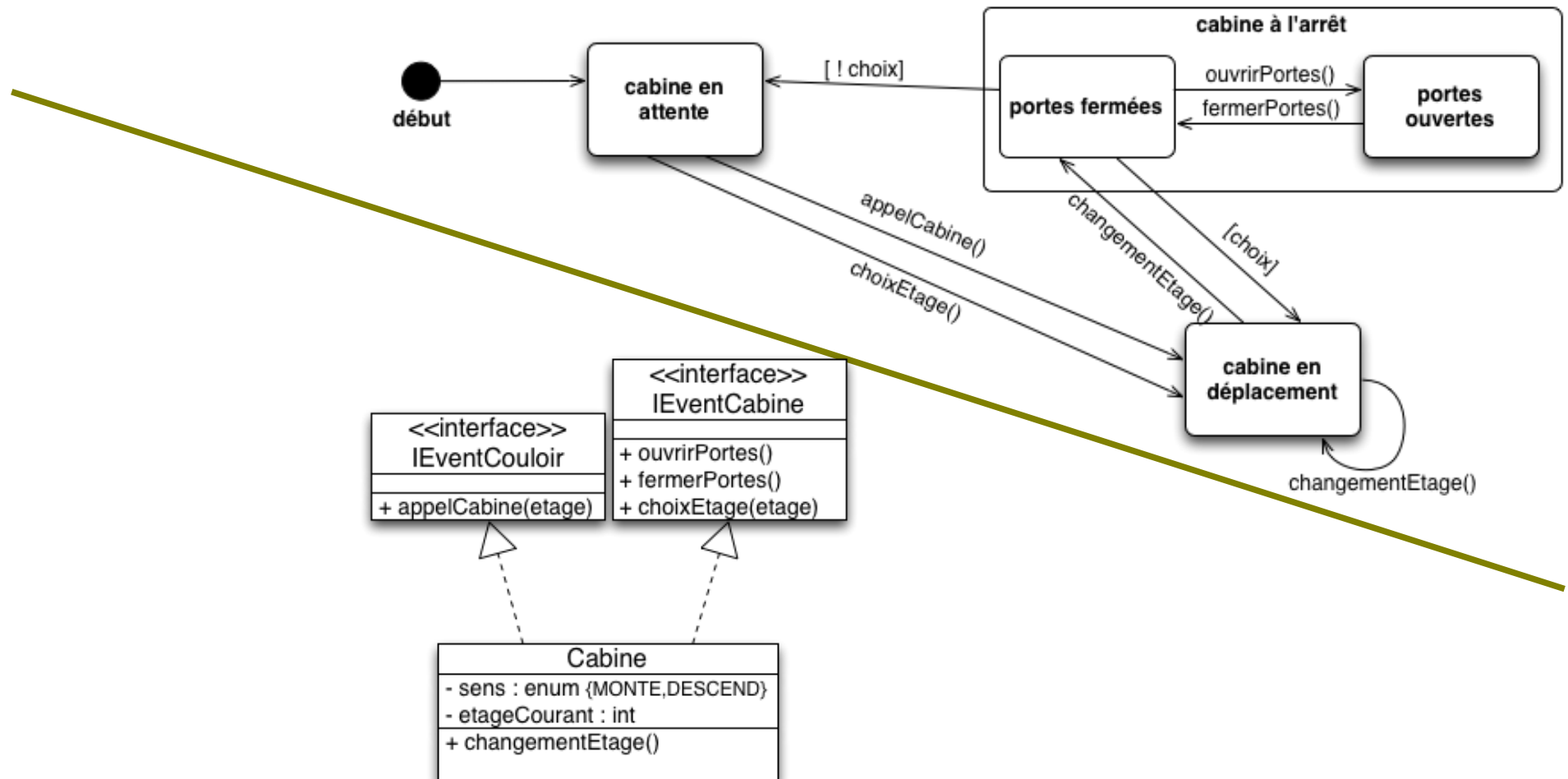
Commandes de la cabine : ouvrir/fermer portes, choisir étage



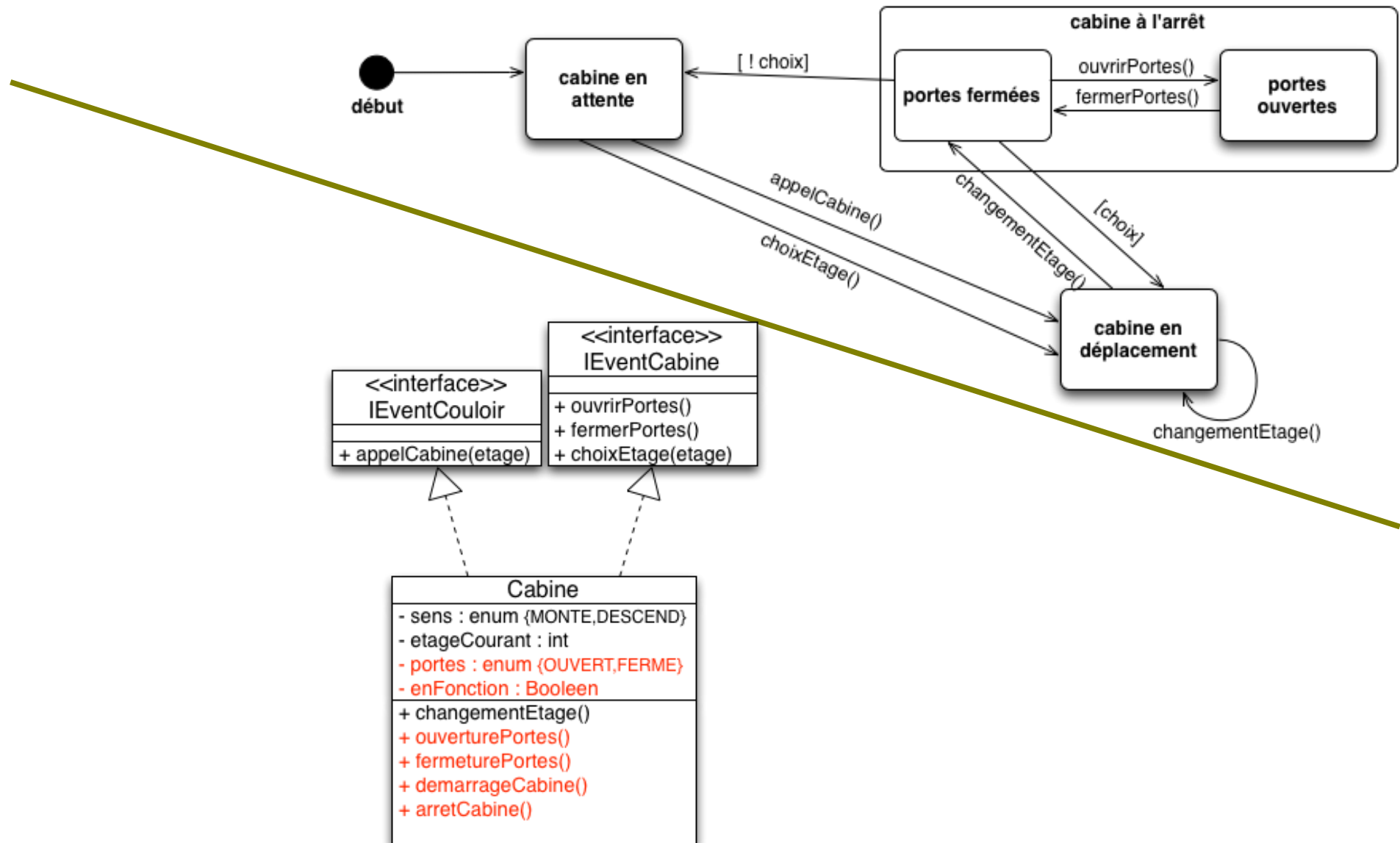
Commandes de la cabine : ouvrir/fermer portes, choisir étage



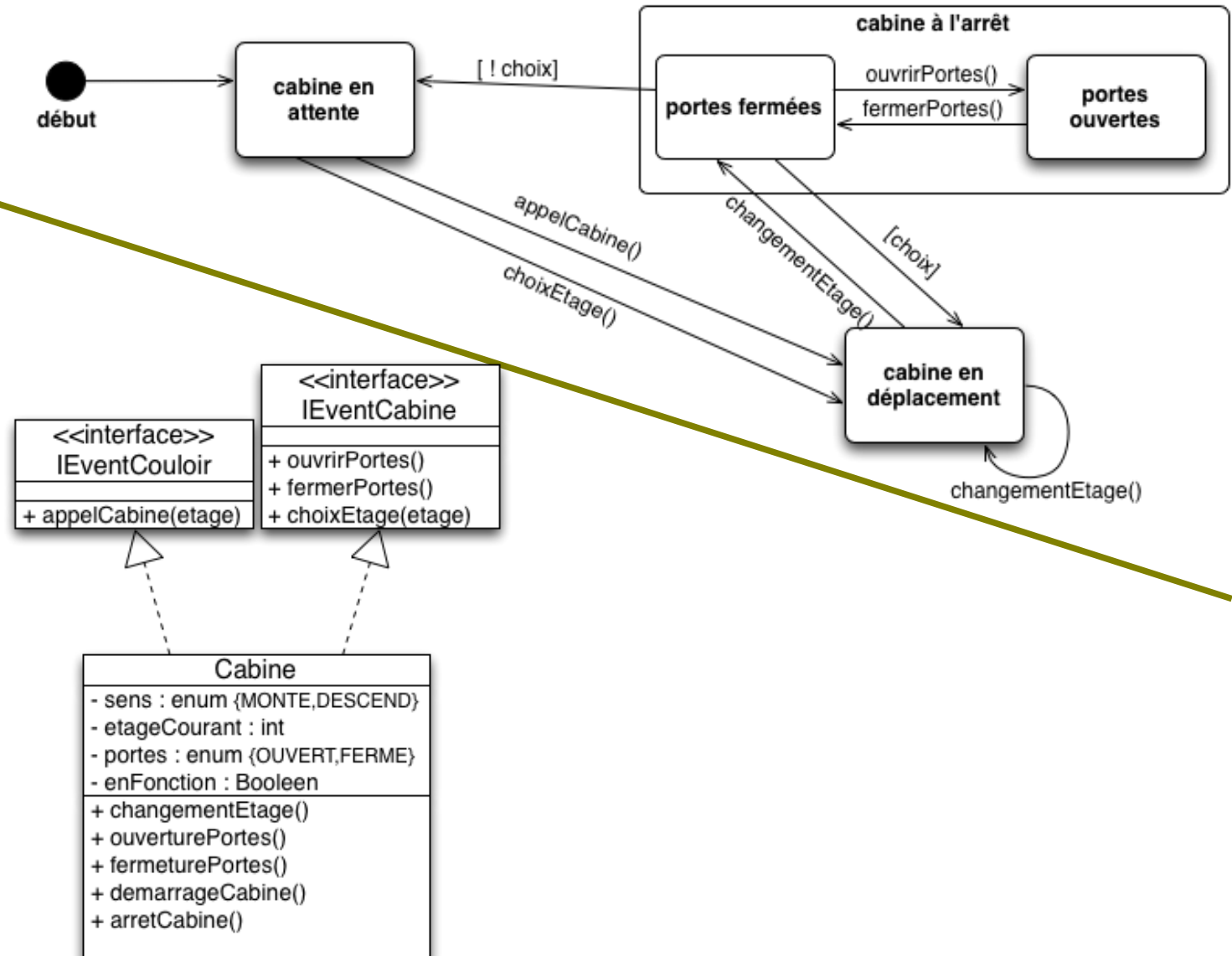
Événements dont la cabine est responsable : ouverture/fermeture des portes, démarrage, arrêt, ...



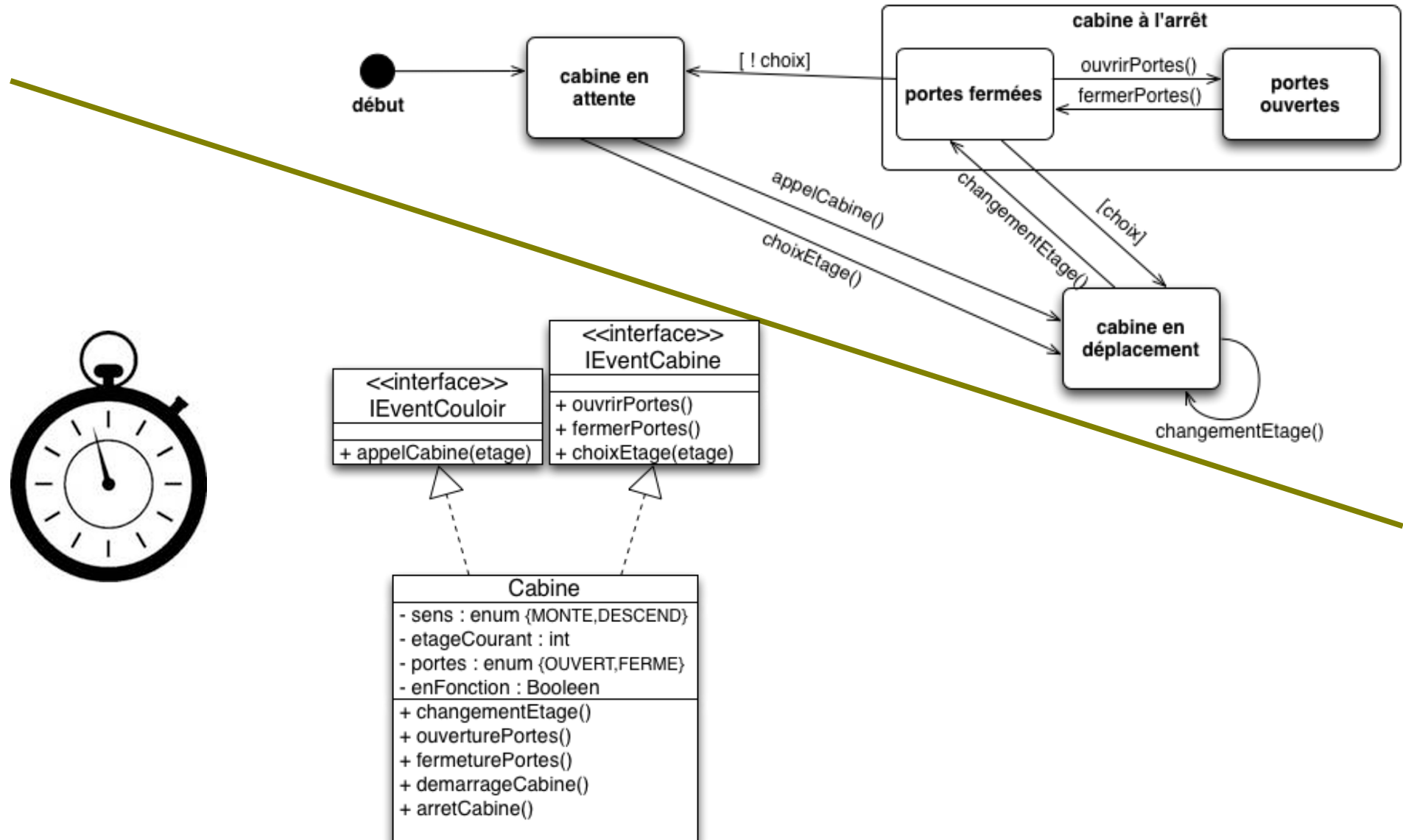
Événements dont la cabine est responsable : ouverture/fermeture des portes, démarrage, arrêt, ...



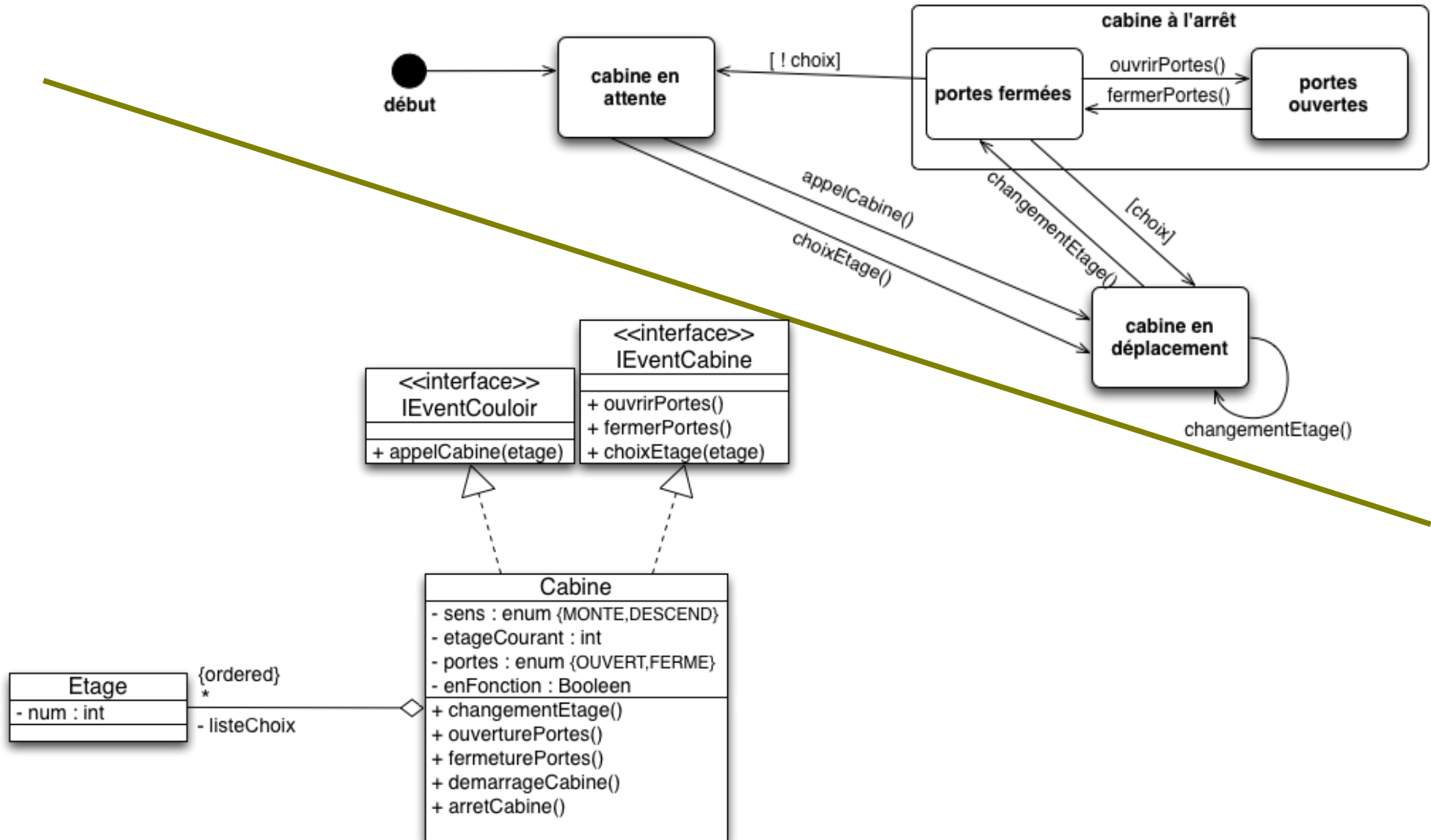
La cabine doit aussi pouvoir gérer (la liste) des étages à desservir



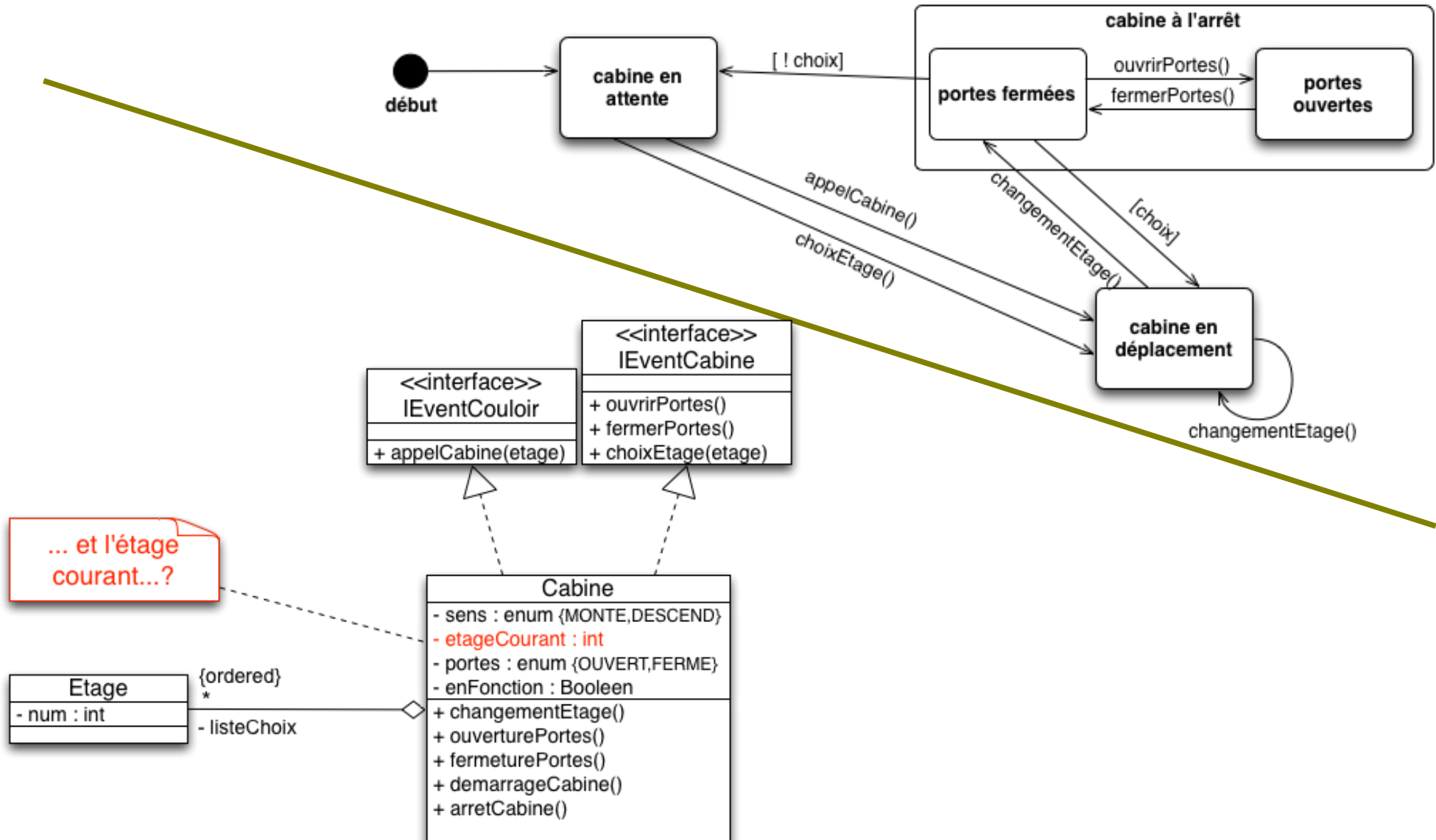
La cabine doit aussi pouvoir gérer (la liste) des étages à desservir



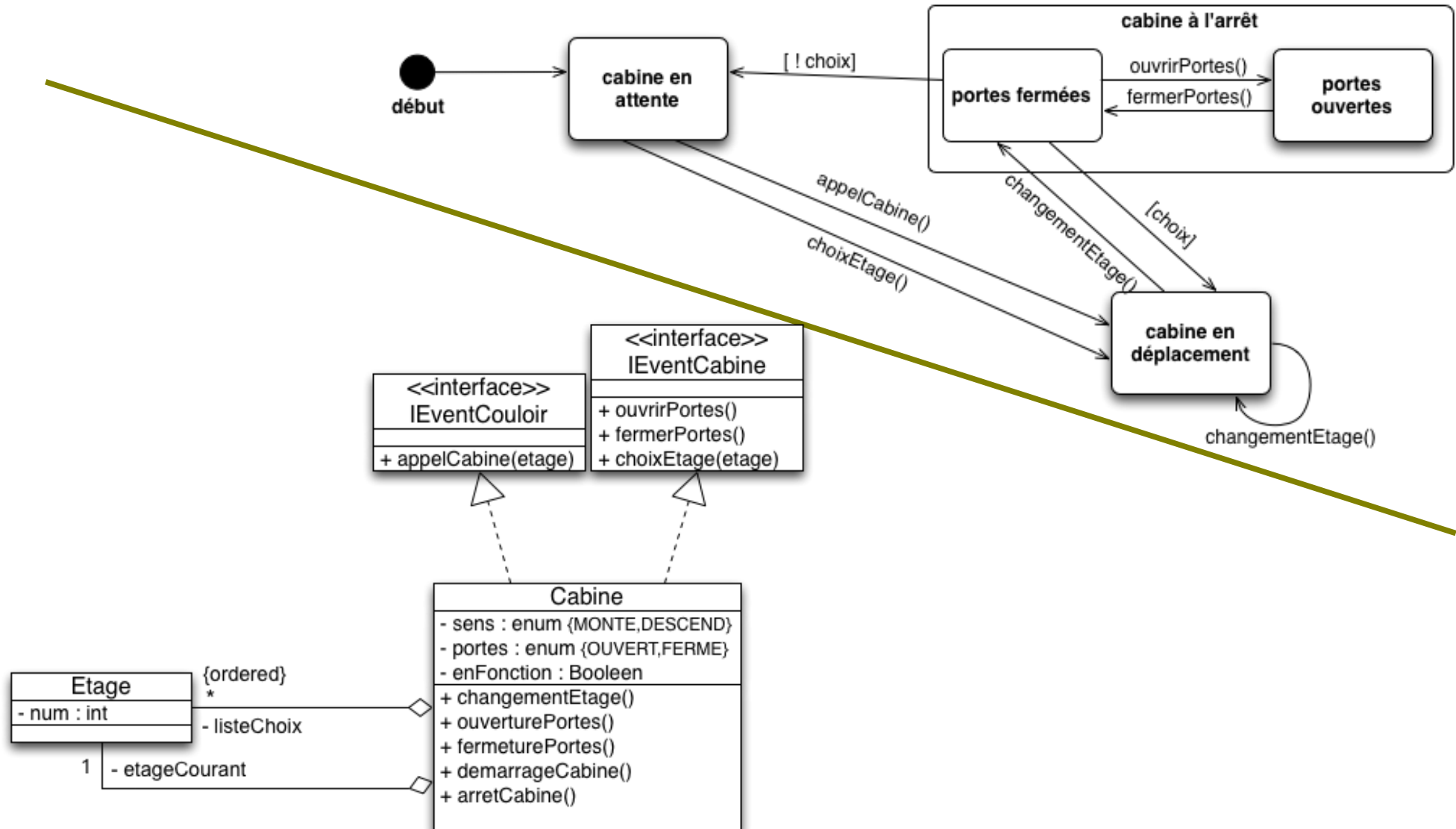
La cabine doit aussi pouvoir gérer (la liste) des étages à desservir



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();
}
```

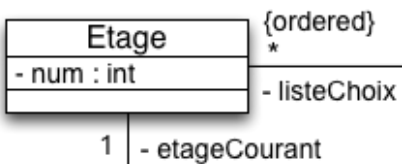
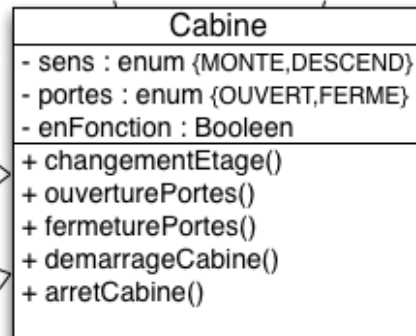
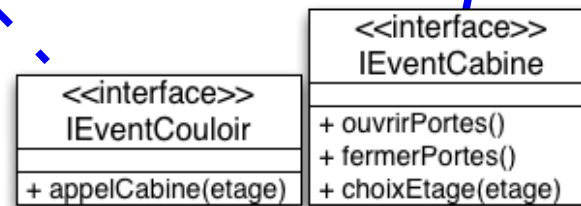
```
interface IEventCabine{
    public void ouvrirPortes();
    public void fermerPortes();
    public void choixEtage(Etage e);
}
```

```
public class Cabine implements IeventCouloir,
IeventCabine{
```

```
    private enum sens {MONTE,DESCEND};
    private enum portes {OUVERT,FERME};
    private Boolean enFonction;
    private ArrayList<Etage> listeChoix;
    private Etage etageCourant;
```

```
    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...

```
interface IEventCouloir{
    public void appelCabine();
}
```

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

```
public class Cabine implements IeventCouloir,
IeventCabine{
```

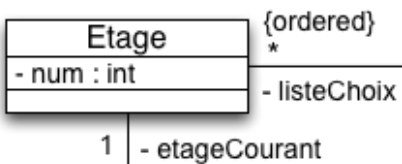
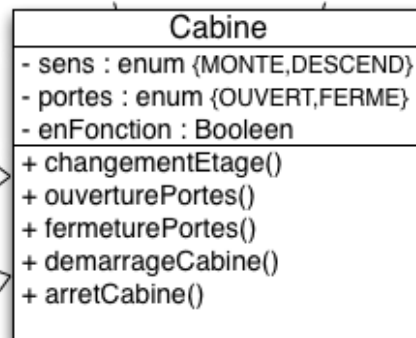
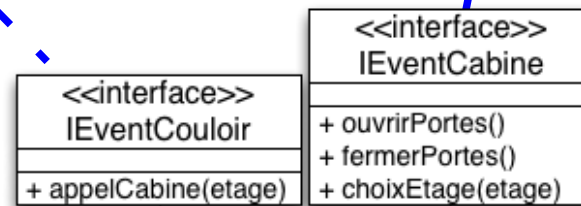
```
    private enum sens {MONTE,DESCEND};
    private enum portes {OUVERT,FERME};
    private Boolean enFonction;
    private A Choix;
    private Etage
```

Our job !

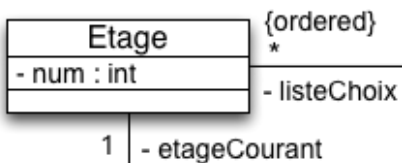
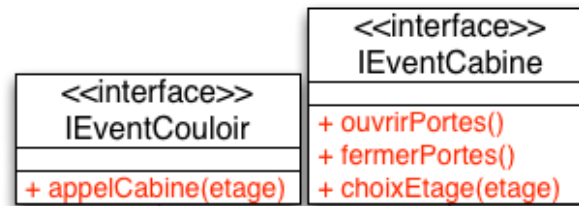
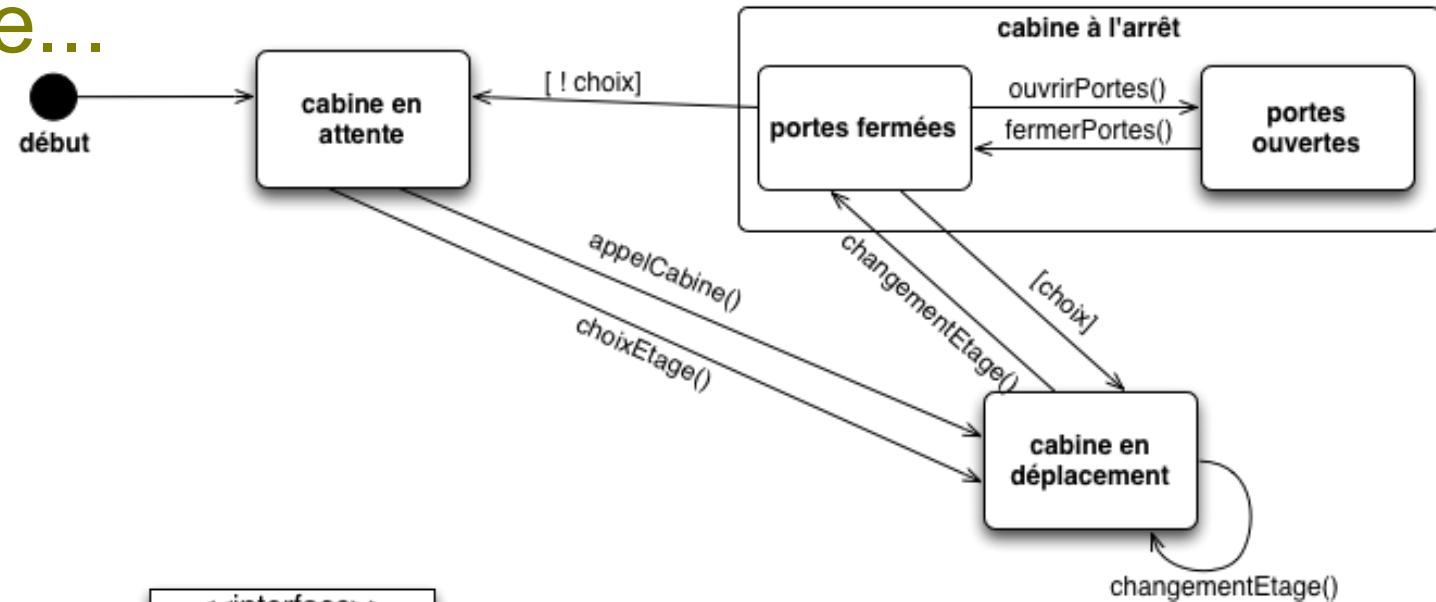
```
    public void appelCabine(Etage e){/*code*/};
    public void ouvrirPortes(){/*code*/};
    public void fermerPortes(){/*code*/};
    public void choixEtage(Etage e){/*code*/};
    public void changementEtage(){/*code*/};
```

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

```
}
```



Un peu de code...



```

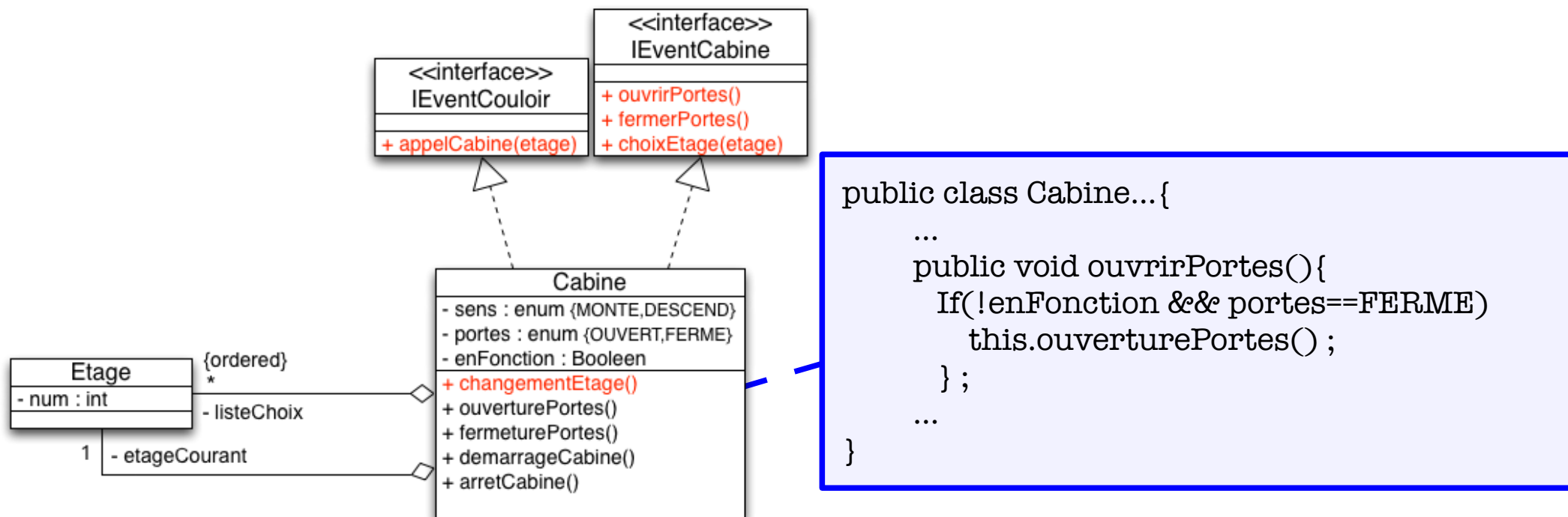
public class Cabine...{
    ...
    public void ouvrirPortes(){
        If(!enFonction && portes==FERME)
            this.ouverturePortes() ;
    } ;
    ...
}
    
```

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**

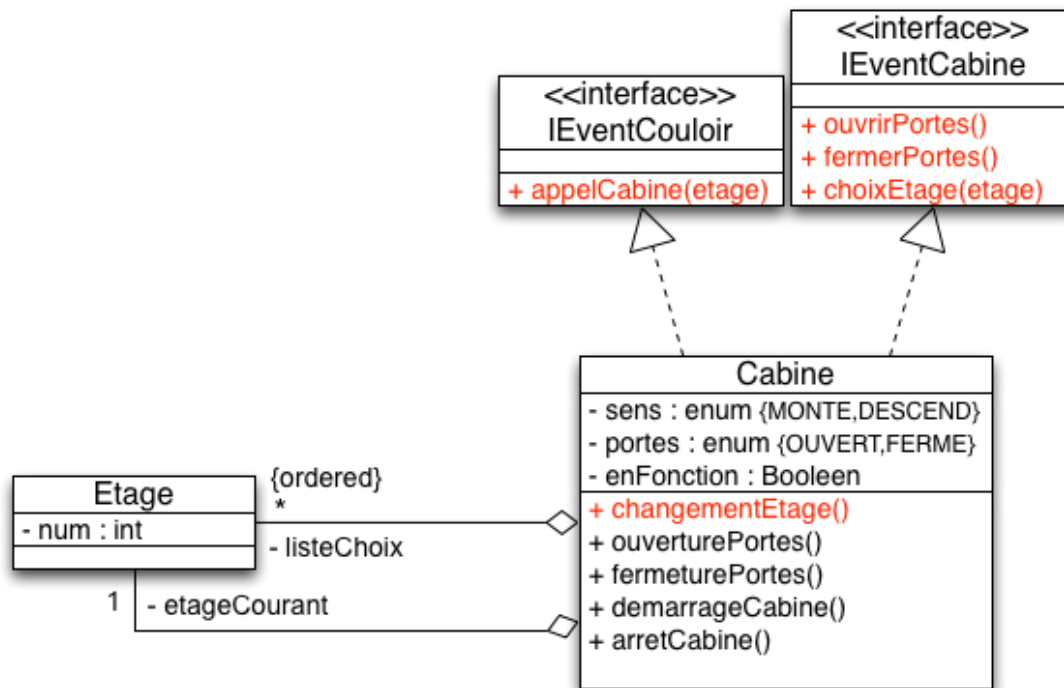
Par exemple, l'ajout d'un état « en maintenance » nécessitera de redéfinir la plupart des opérations.



Question

Comment organiser le code de manière à :

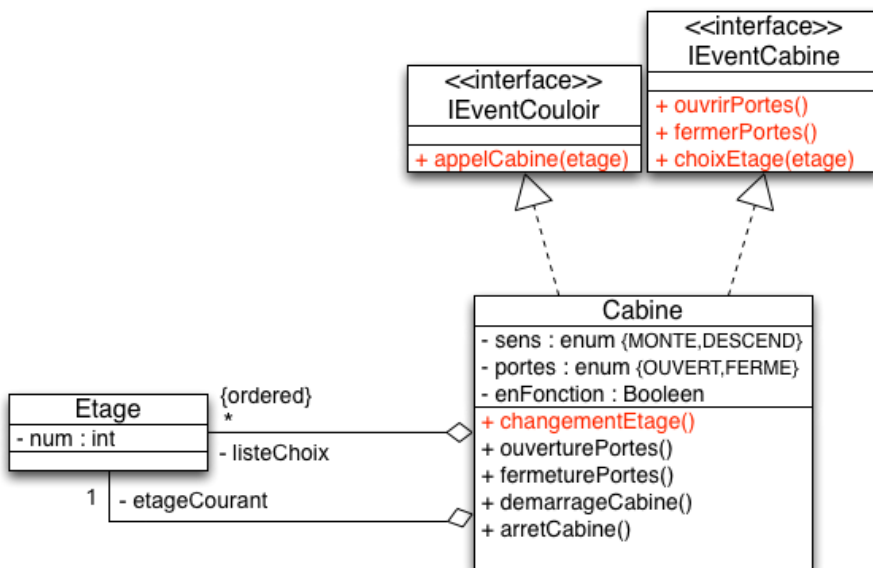
- expliciter les états possibles et l'état courant de la cabine
- lui faire adopter un comportement différent selon son état ?



Question

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** ?

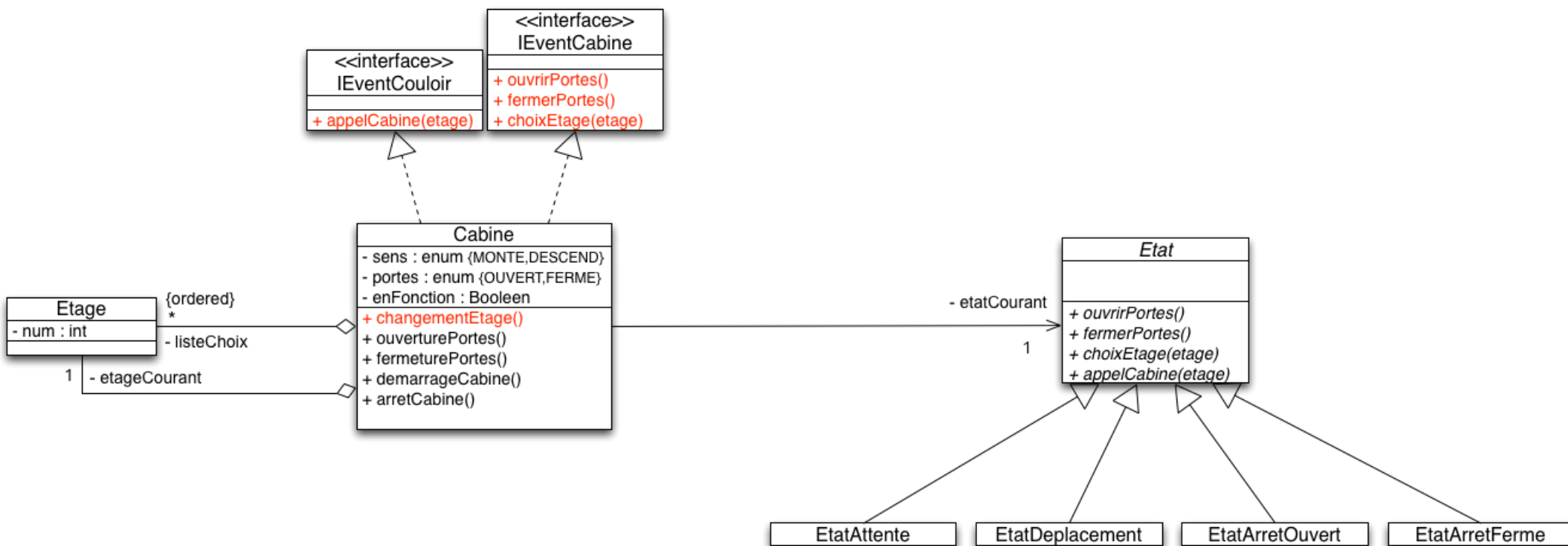


EtatAttente	EtatDeplacement	EtatArretOuvert	EtatArretFerme
+ ouvrirPortes() + fermerPortes() + choixEtage(etage) + appelCabine(etage)	+ ouvrirPortes() + fermerPortes() + choixEtage(etage) + appelCabine(etage)	+ ouvrirPortes() + fermerPortes() + choixEtage(etage) + appelCabine(etage)	+ ouvrirPortes() + fermerPortes() + choixEtage(etage) + appelCabine(etage)

Question

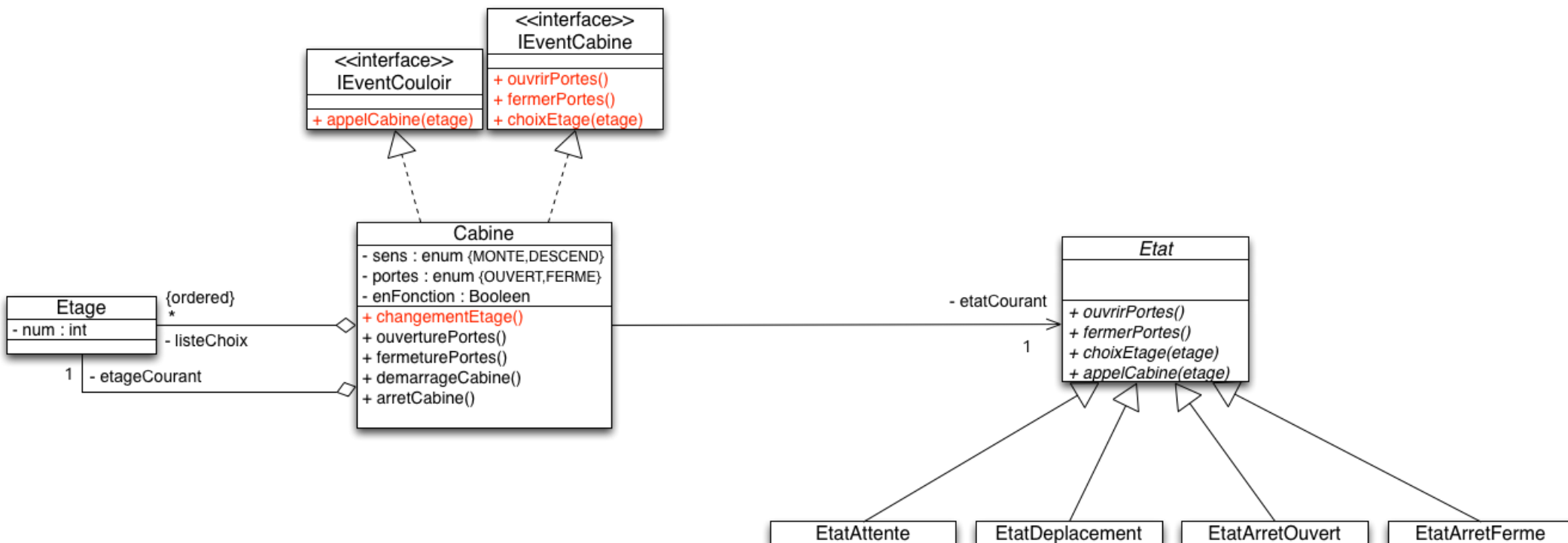
Comment organiser le code de manière à :

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



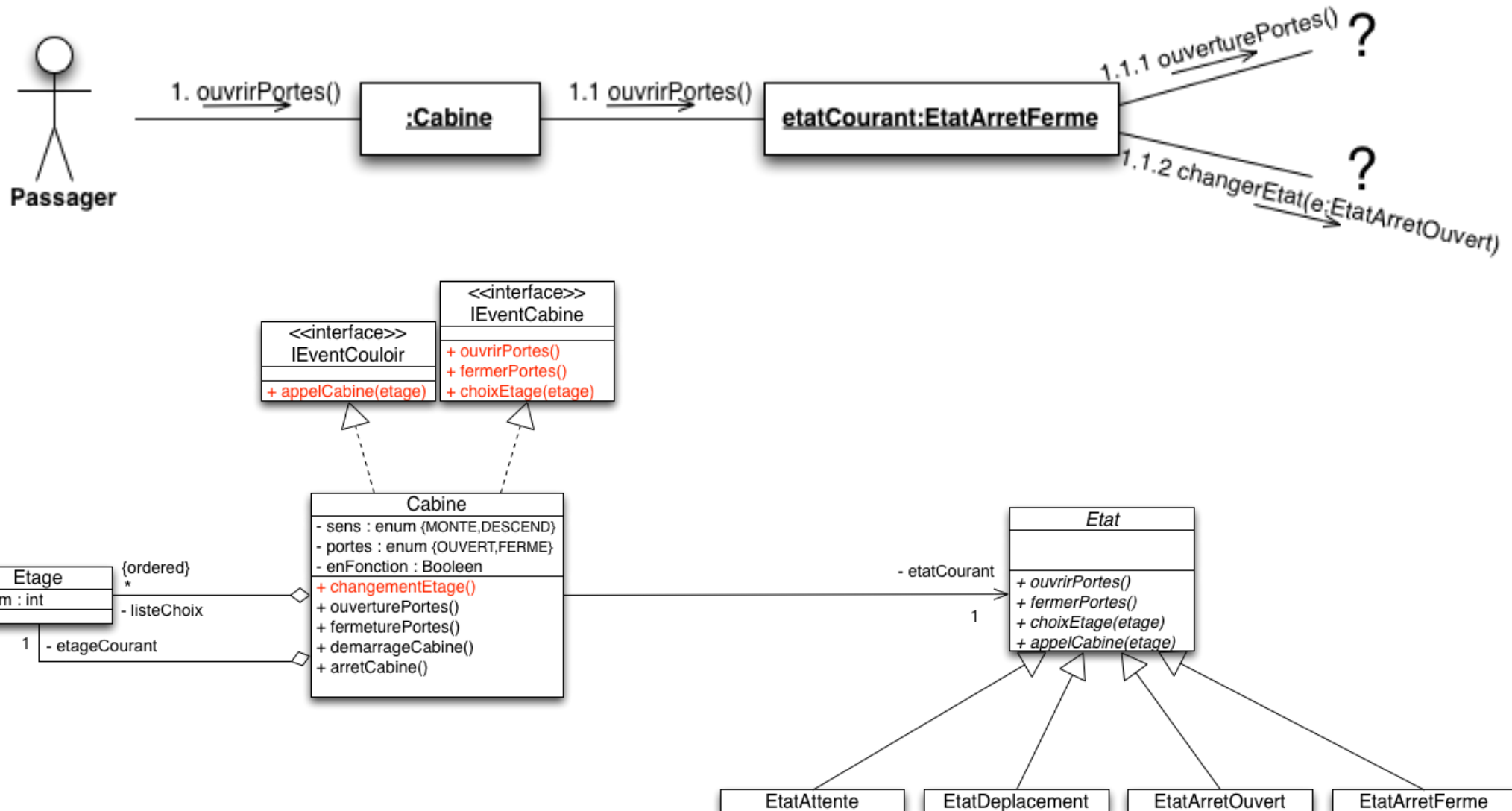
Question

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



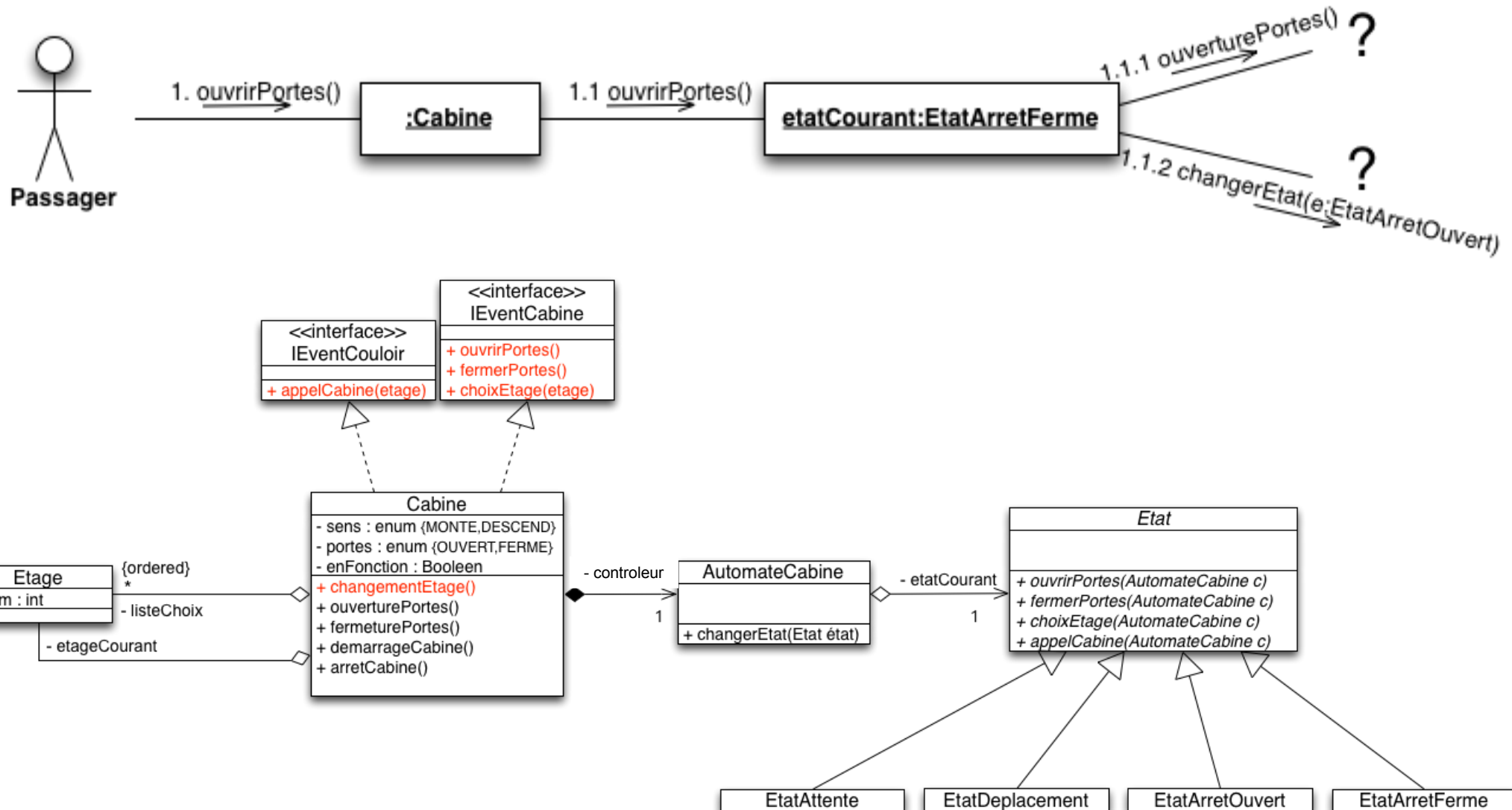
Question

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



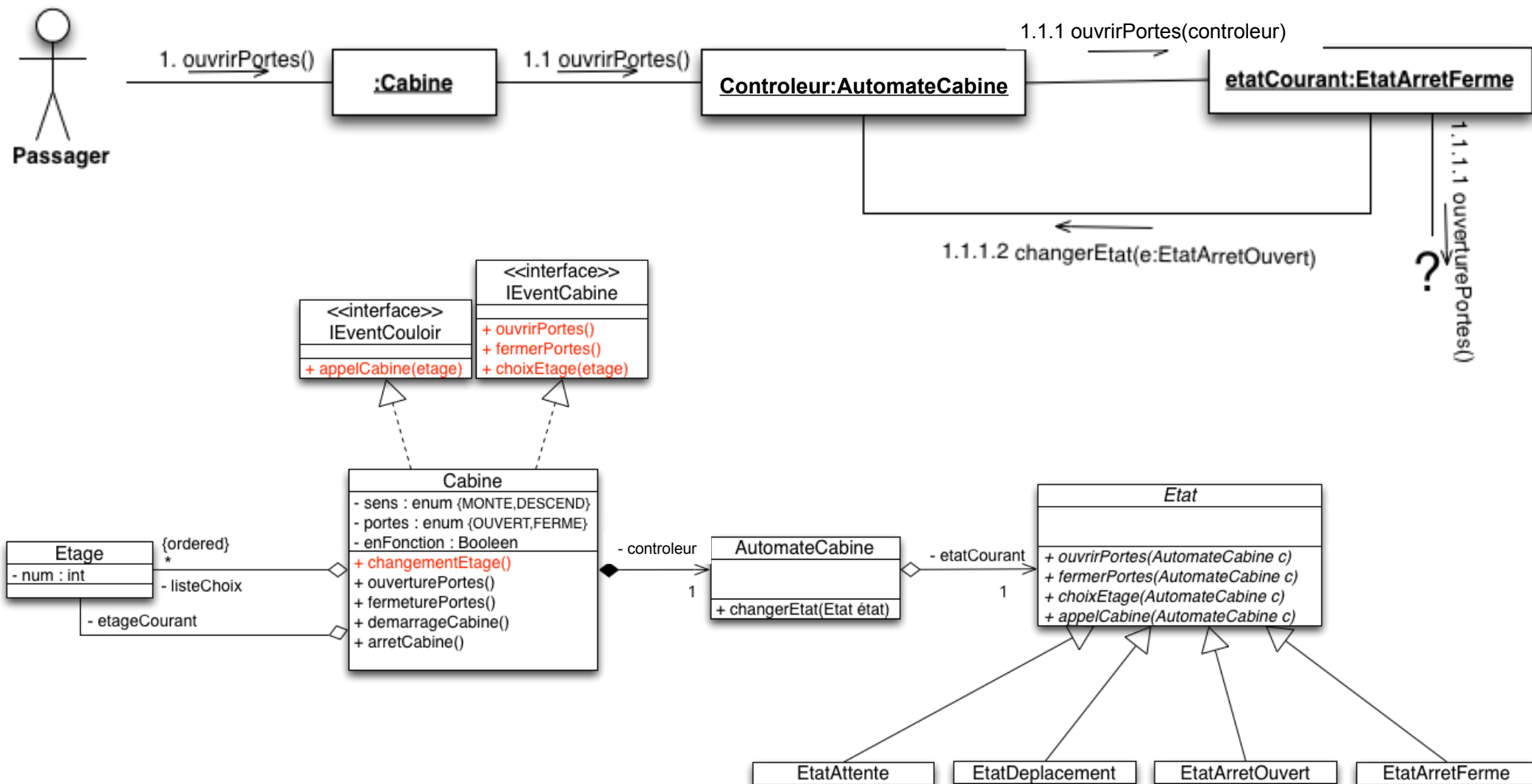
Question

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



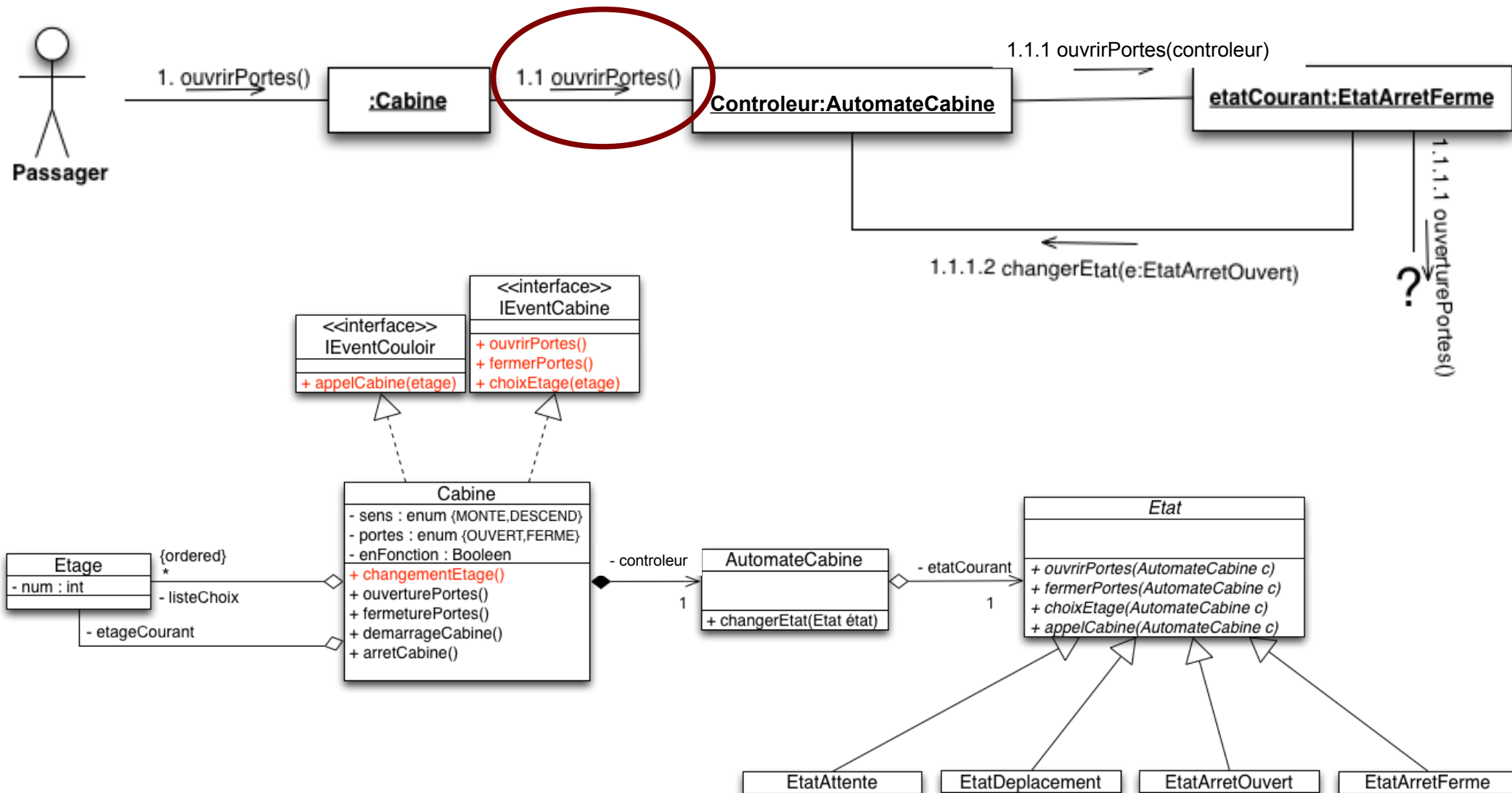
Question

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



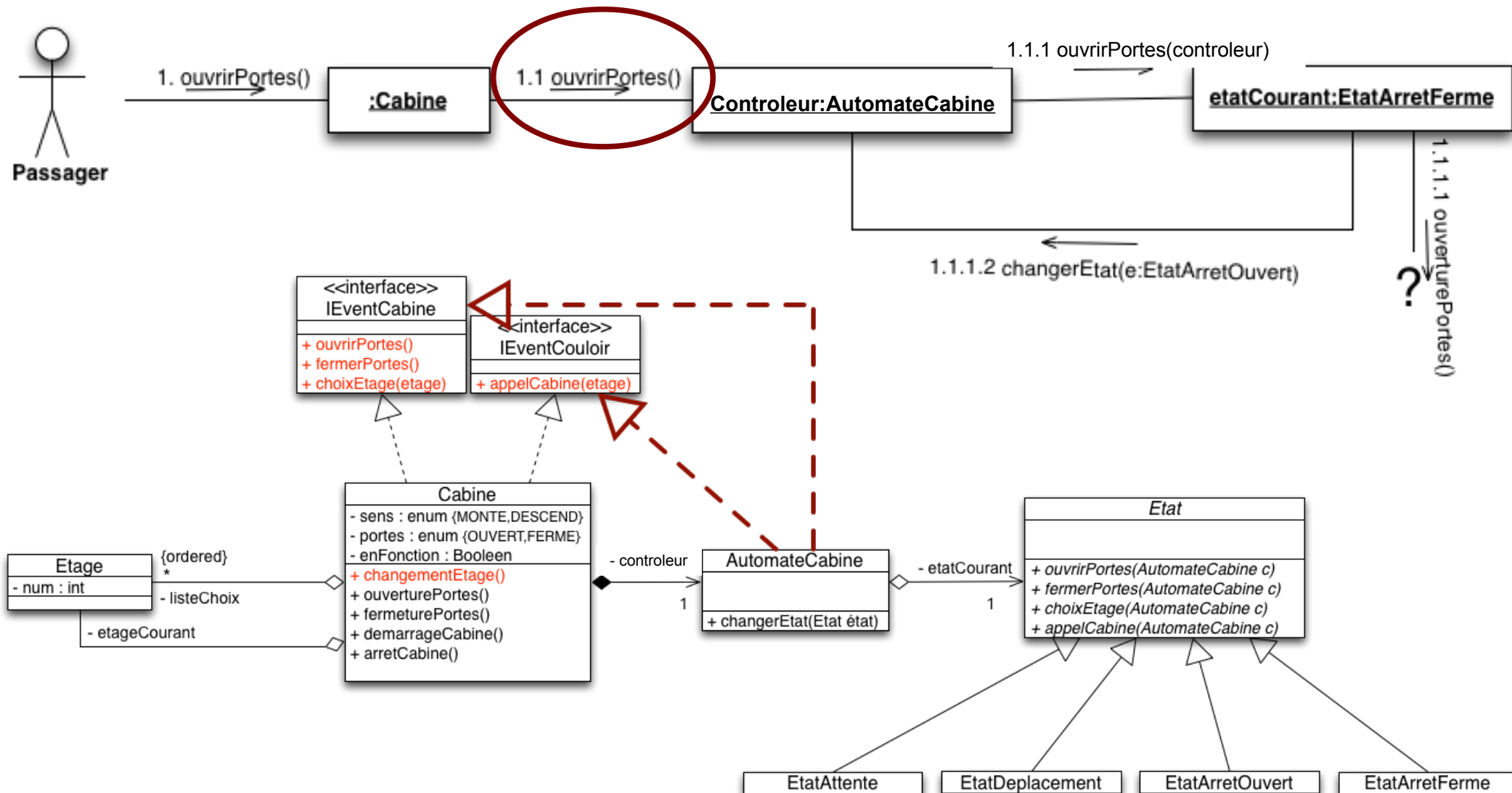
Question

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



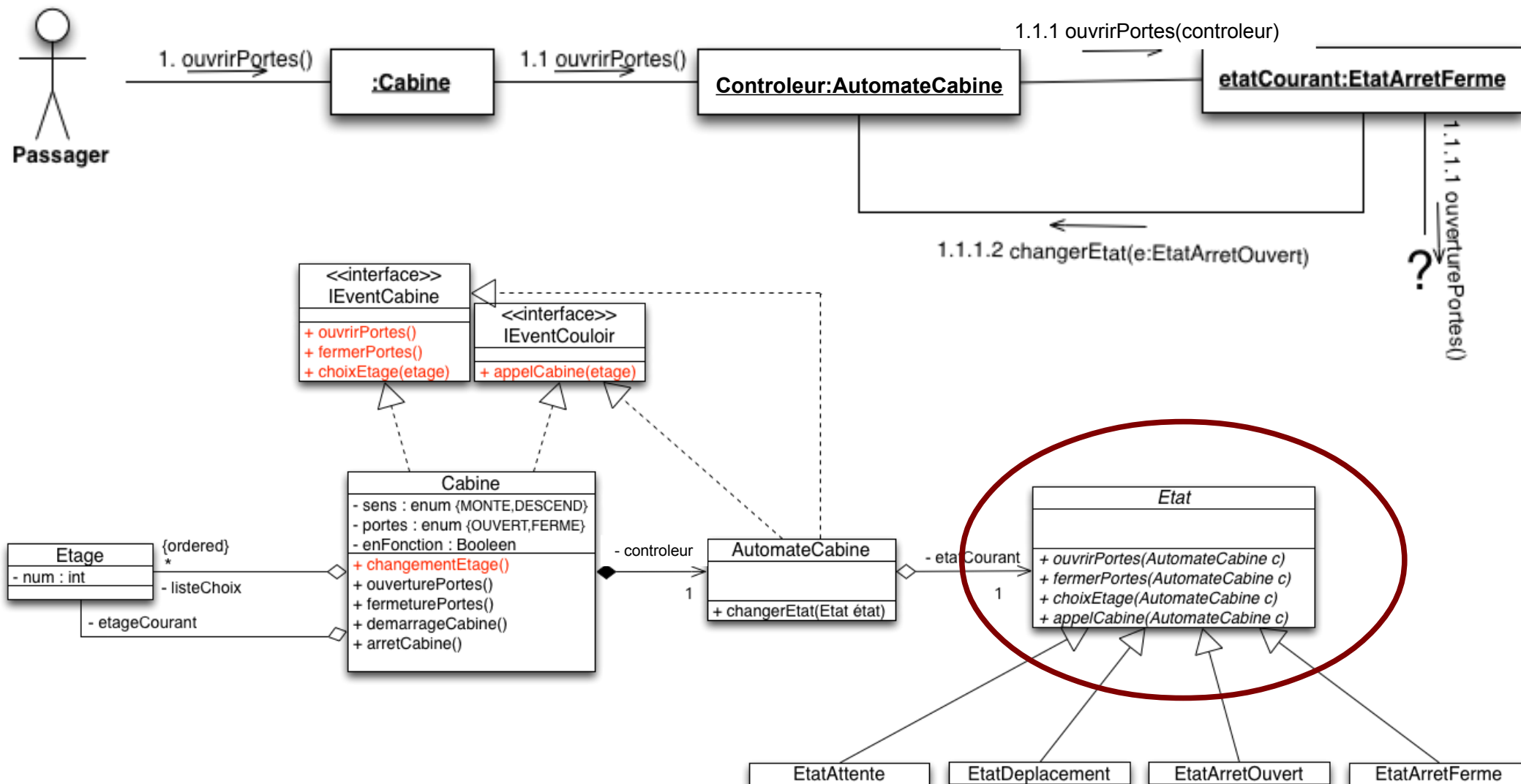
Question

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



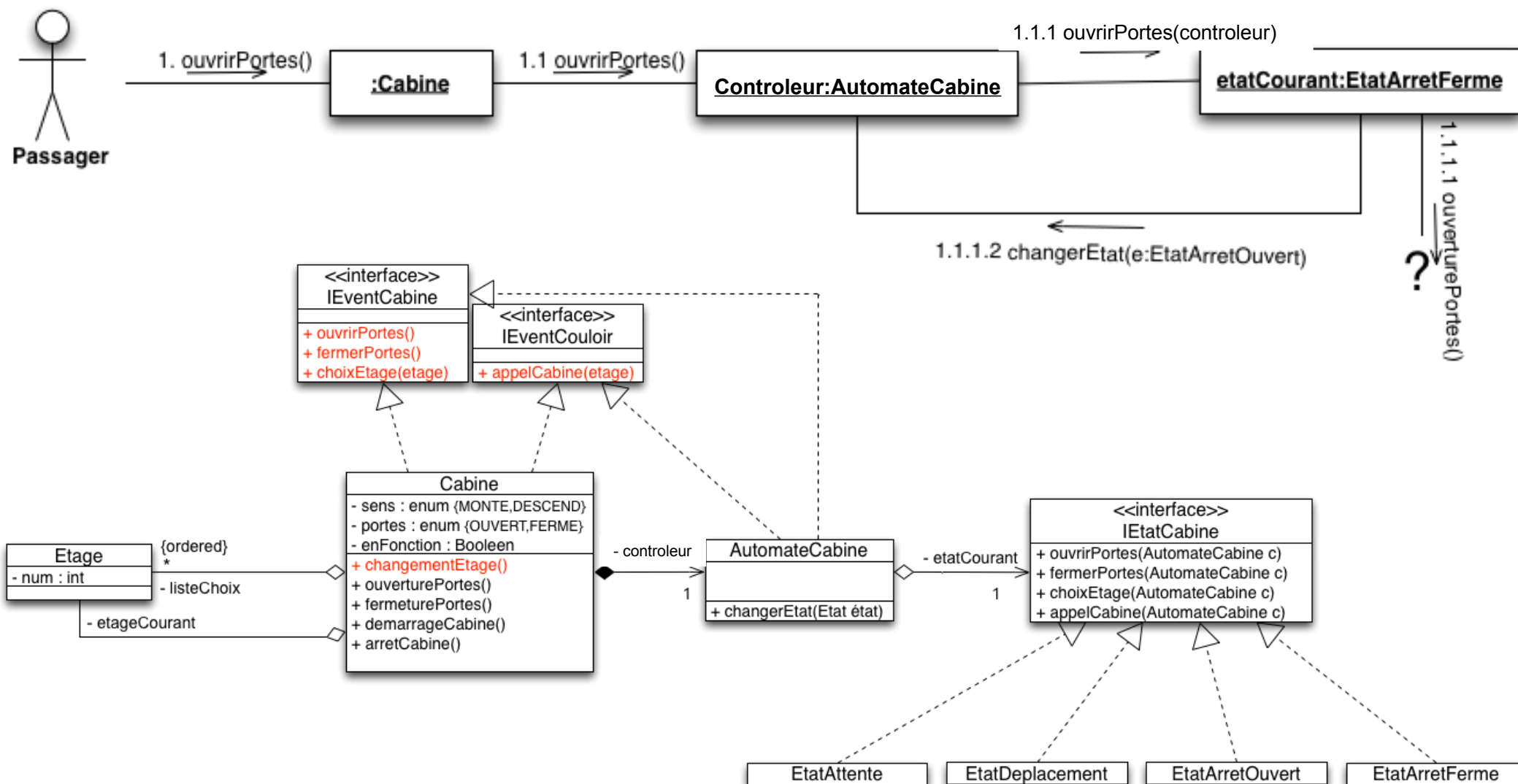
Question

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



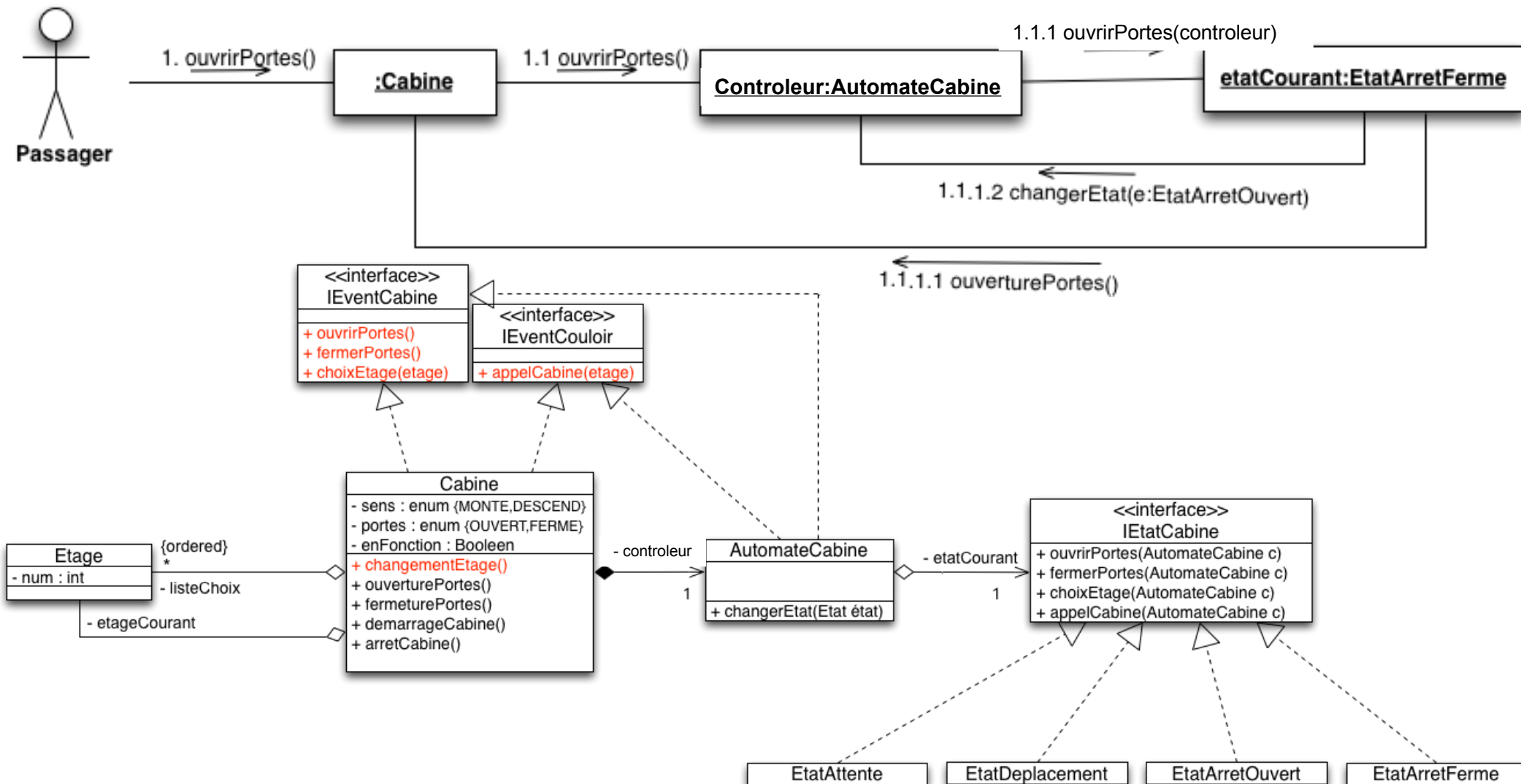
Question

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



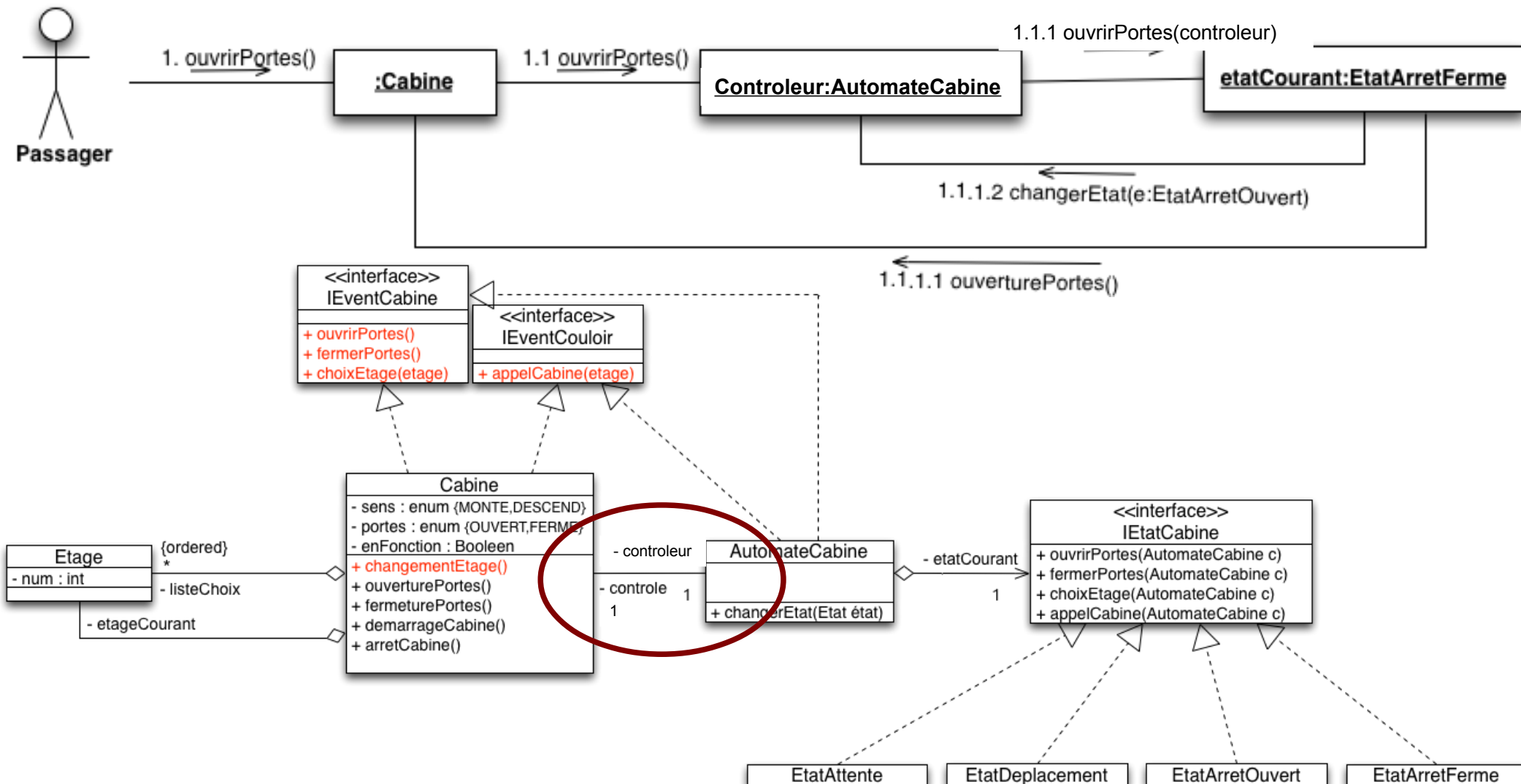
Question

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

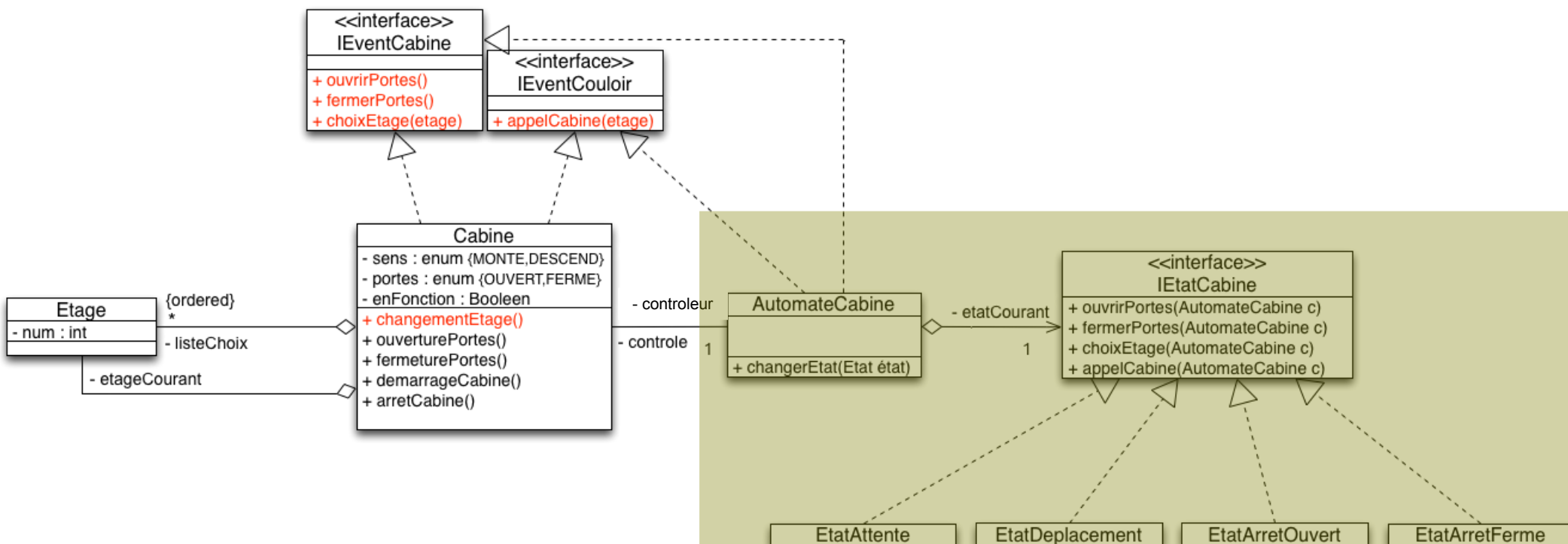


Question

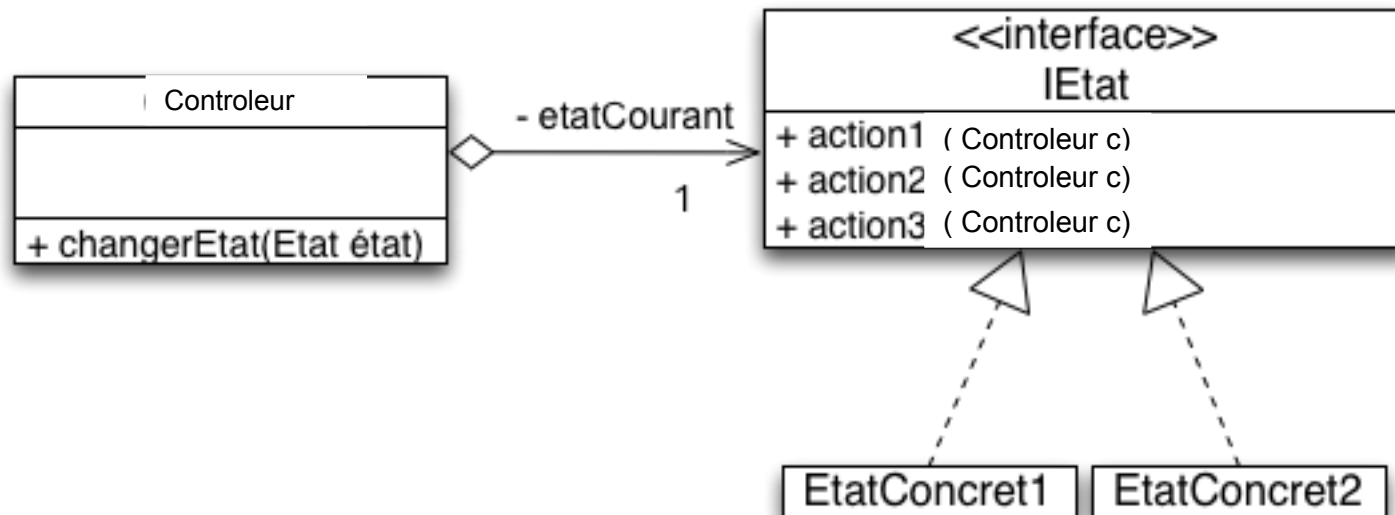
Formaliser l'appel de *ouvrirPortes()* lorsque la cabine est dans l'état d'arrêt portes fermées



Design pattern « Etat » (state)



Design pattern « Etat » (state)



Questions ?

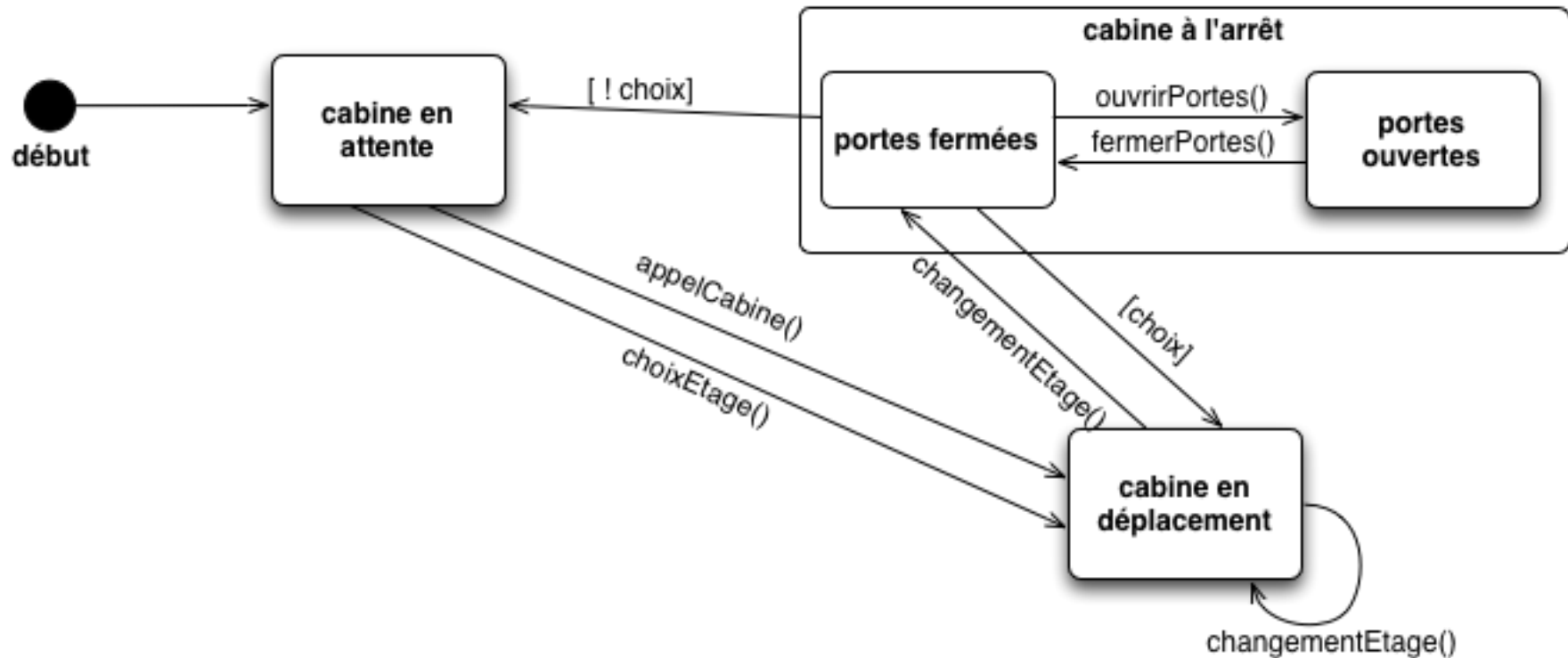


Ce diagramme UML correspond à un diagramme :

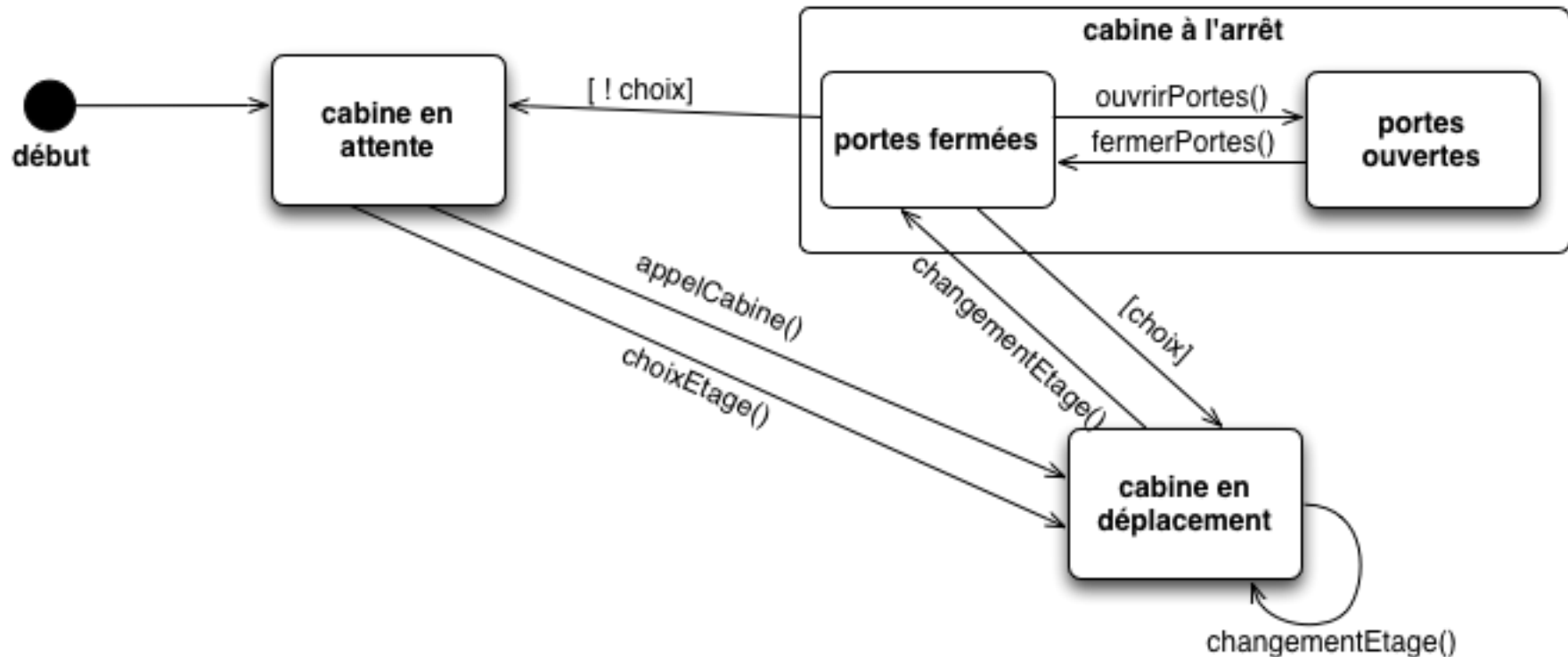


Ce diagramme UML correspond à un diagramme :

- A : de collaboration
- B : d'objets
- C : d'Etats-Transitions
- D : de séquence

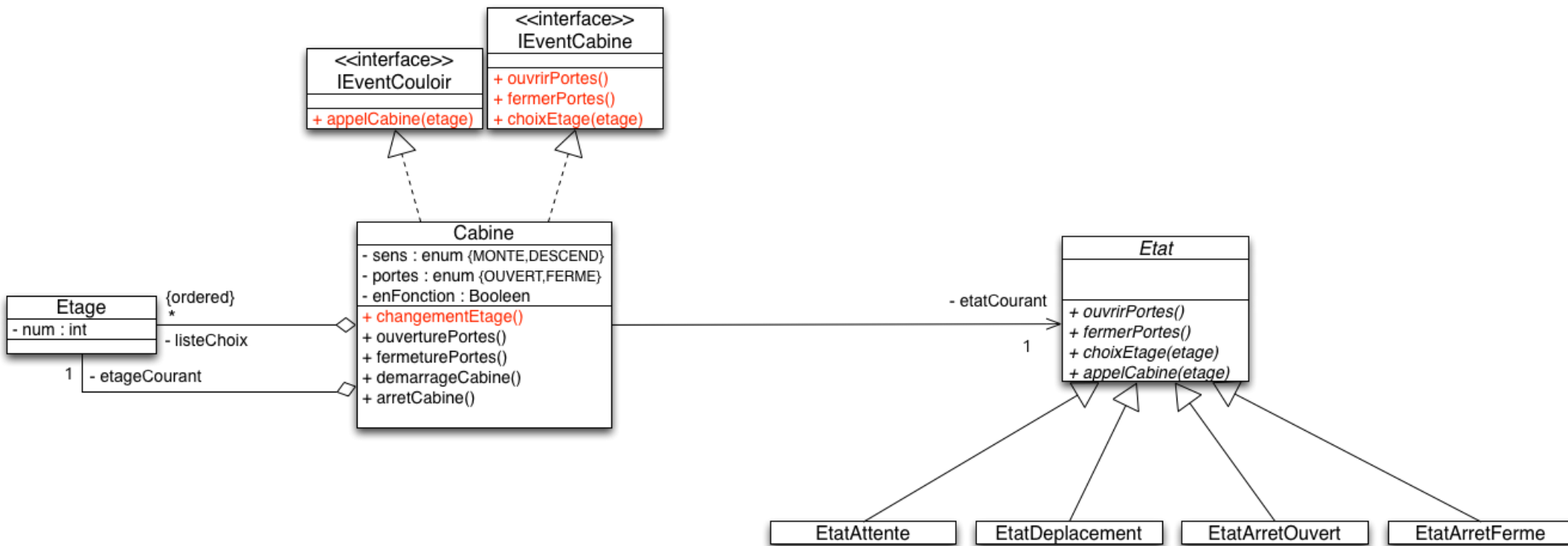


Parmi les affirmations suivantes laquelle est bien modélisée par ce diagramme ?

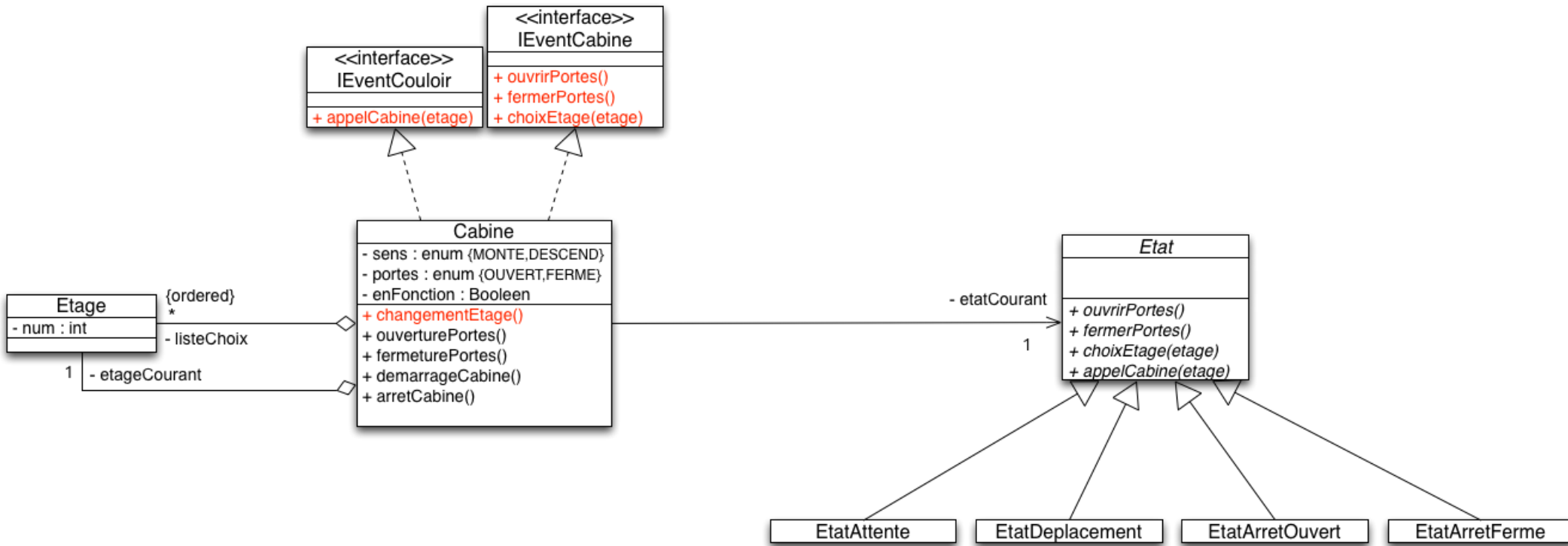


Parmi les affirmations suivantes laquelle est bien modélisée par ce diagramme ?

- A : à l'arrêt, les portes de la cabine sont toujours ouvertes
- B : les portes de la cabine doivent être fermées pour passer en « attente »
- C : l'appel de *changementEtage()* entraîne le déplacement de la cabine
- D : à l'initialisation, la cabine a les portes ouvertes

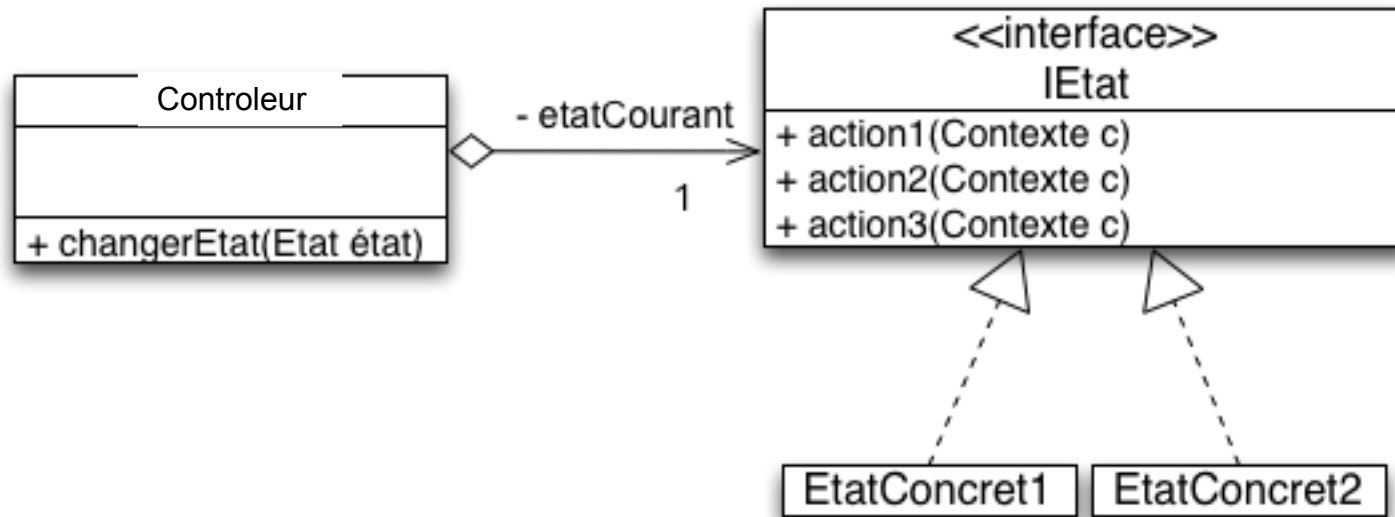


Quelle modification a été apportée à ce diagramme pour permettre le changement d'état de la cabine ?

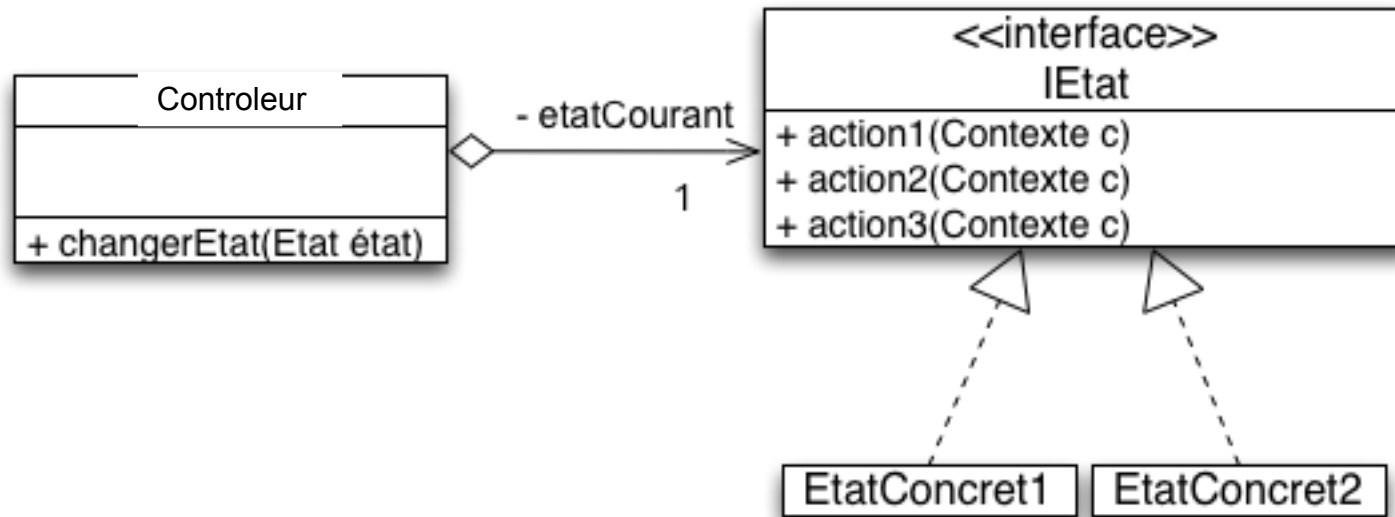


Quelle modification a été apportée à ce diagramme pour permettre le changement d'état de la cabine ?

- A : ajout d'une classe entre Cabine et *Etat*
- B : ajout d'une méthode dans *Etat*
- C : ajout d'associations entre les sous-classes d'*Etat*
- D : transformation de *Etat* en une interface



Dans le design pattern « Etat », quelle classe peut jouer le rôle de la cabine d'ascenseur ?



Dans le design pattern « Etat », quelle classe peut jouer le rôle de la cabine d'ascenseur ?

• A : Controleur

• B : IEtat

• C : EtatConcret1

• D : aucune