Cours 5 - C++ pour les mathématiques appliquées

Conception Objet

La dernière fois . . .

Analyse

- Diagramme des cas d'utilisations
- Diagramme de séquence
- Diagramme de transitions
- Diagramme de classes d'analyse

Conception logicielle

Conception architecturale

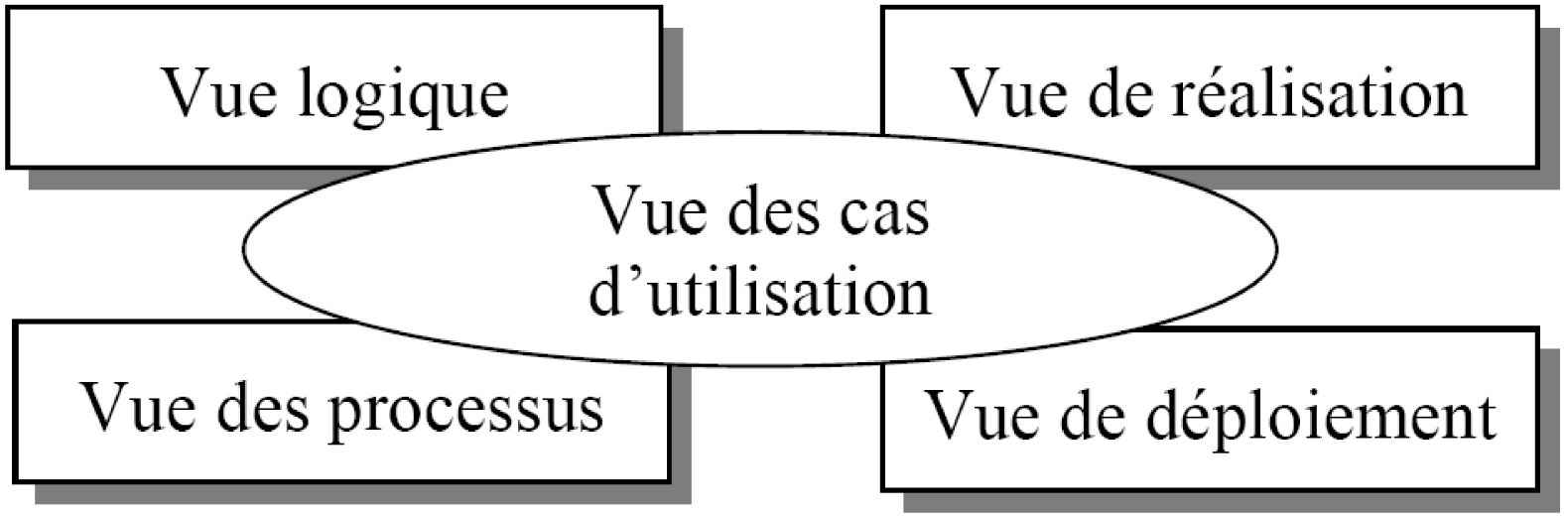
Architecture logicielle

Décrit la structure générale du système sous forme de

- constituants de haut niveau
- interactions entre ces éléments

Description architecturale

L'architecture du système est décrite par des *vues*



Le modèle des *4 + 1 vues* de Philippe Kruchten

Vue logique

- Décrit l'organisation du système en, sous-systèmes, couches, paquetages, classes et interfaces.
- Un paquetage regroupe un ensemble de classes et d'interfaces.
- Un sous-système est un sous-ensemble du système, fournissant un ensemble d'interfaces et d'opérations.

Vue de réalisation

- Concerne l'organisation des différents fichiers (exécutables, code source, documentation...) dans l'environnement de développement, ainsi que la gestion des versions et des configurations.
- Peut être en partie décrite à l'aide d'un diagramme de composants.

Vue des processus

- Représente la décomposition en différents flots d'exécution : processus, fils d'exécution (threads).
- Cette vue est importante dans les environnements multi-tâches.

Vue de déploiement

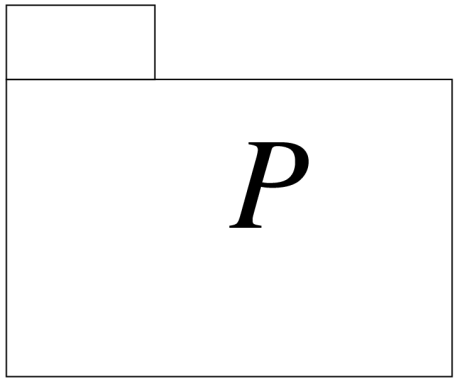
- Décrit les différentes ressources matérielles et l'implantation du logiciel sur ces ressources.
- Cette vue concerne les liens réseau entre les machines, les performances du système, la tolérance aux fautes et aux pannes.
- Cette vue peut être décrite à l'aide d'un diagramme de déploiement.

Paquetage UML

Elément clé : les paquetages

Regroupe un ensemble d'éléments de modélisation, en particulier classes et interfaces.

- On peut utiliser les paquetages pour décrire la vue logique de l'architecture



Principes de conception architecturale

Architecture : pour faire joli mais pas que ;)

Comment respecter les besoins ?

- modularité
- évolutivité
- isolation

Quels diagrammes pour l'architecture ?

Diagramme de classe

Un diagramme de classe qui va évoluer au fur et à mesure de la conception.

Au départ, une structure générale à base de sous-ensemble de classes.

Au fur et à mesure, une structure respectée tout au long de la conception, du développement et de l'évolution du logiciel.

Quels diagrammes pour l'architecture ? Complément

Diagrammes de séquence (ou collaboration)

Explique le fonctionnement sous forme de scénario d'appel.

Structurer mais à quel degré ?

Bons principes de conception architecturale :

- **Couplage :** degré de dépendance d'un paquetage avec d'autres. Le couplage doit être *faible !*
- **Cohésion :** lien et cohérence des services proposés par un paquetage. La cohésion doit être **forte !**
- **Protections des variations :** un paquetage très utilisé doit peut varier au cours du développement

Couplage

- Le couplage mesure le degré selon lequel un paquetage est lié à d'autres paquetages.
- Des paquetages fortement couplés ne sont pas indiqués, car ils seront sensibles à beaucoup de modifications. De plus, ils sont plus difficiles à comprendre isolément, et difficiles à réutiliser.
- Principe de conception architecturale : concevoir des paquetages faiblement couplés.

Cohésion

- Mesure les liens, la cohérence entre les différents services proposés par le paquetage.
- Il est préférable de réaliser des paquetages ayant une forte cohésion, car ils sont plus faciles à comprendre et à réutiliser.

Variations

- Au cours du développement d'un logiciel, certaines parties sont rapidement stables, et d'autres au contraire subissent de nombreuses modifications.
- Le principe de protection des variations consiste à éviter qu'un paquetage utilisé par de nombreux autres paquetages ne subisse trop de variations.
- Un paquetage utilisé par de nombreux autres paquetages doit être rapidement stable.

Exemple de mauvais paquetage

Paquetage

- La documentation de ce paquetage indique : « Le paquetage contient les collections, le modèle d'événements, des utilitaires de date et heure, d'internationalisation et des classes utilitaires diverses comme un « string tokenizer » [analyseur lexical], un générateur aléatoire et un tableau de bits. »
- La cohésion de ce paquetage n'est pas très forte : il propose un certain nombre d'utilitaires n'ayant aucun rapport entre eux.

Architecture logicielle

Deux exemples d'architecture logique

1. Architecture en couches
2. Architecture Modèle - Vue - Contrôleur

Architecture en couches

- Le logiciel est organisé en couches.
- Une couche regroupe un ensemble de classes et propose un ensemble cohérent de services à travers une interface.

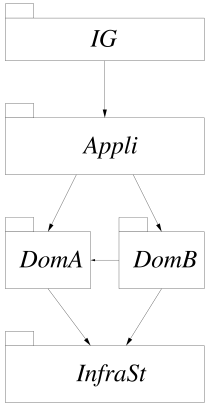
Propriétés

- **couches ordonnées :** une couche ne peut accéder qu'à des couches de niveau inférieur.
- **couches étanches :** une couche d'un niveau n ne peut accéder qu'aux couches de niveau $n-1$.

Exemples d'architecture en couches (1)

Application utilisant une interface graphique

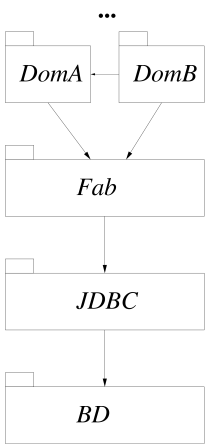
- **couche IG :** spécifique à l'interface graphique
- **couche Appli :** *médiation* entre l'IG et les couches du domaine
- **couches DomA, DomB :** couches du domaine
- **couche InfraSt :** couche infrastructure, service technique de bas niveau



Exemples d'architecture en couches (2)

Application utilisant une base de données

- . . . au dessus . . .
- **couches DomA, DomB :** couches du domaine
- **couche BD :** base de données



Exemples d'architecture en couches (3)

Thales avionique (UML/AADL)

- **couche Visualisation**
- **couche Application :** tout ce qui est spécifique à cette application
- **couche Métiers :** briques fonctionnelles réutilisables
- **couche Service :** bibliothèques (exemple: filtre)

Evaluation de l'architecture en couches (1)

Maintenance

- Le système est plus facilement modifiable. Une modification d'une couche n'affecte pas les couches de niveau inférieur.
- Une modification d'une couche qui ne modifie pas l'interface publique n'affecte pas les couches de niveau supérieur ou égal.

Réutilisation

- Des éléments de chaque couche peuvent être réutilisés. Par exemple, des couches « domaine » peuvent être communes à plusieurs applications.

Evaluation de l'architecture en couches (2)

Portabilité

- Les éléments qui dépendent du système d'exploitation sont confinés dans les couches basses.
- On réécrit les couches basses pour chaque système, et les couches hautes sont portables.

Inconvénient

problème d'efficacité si on a un grand nombre de couches étanches.

Architecture MVC

Généralités

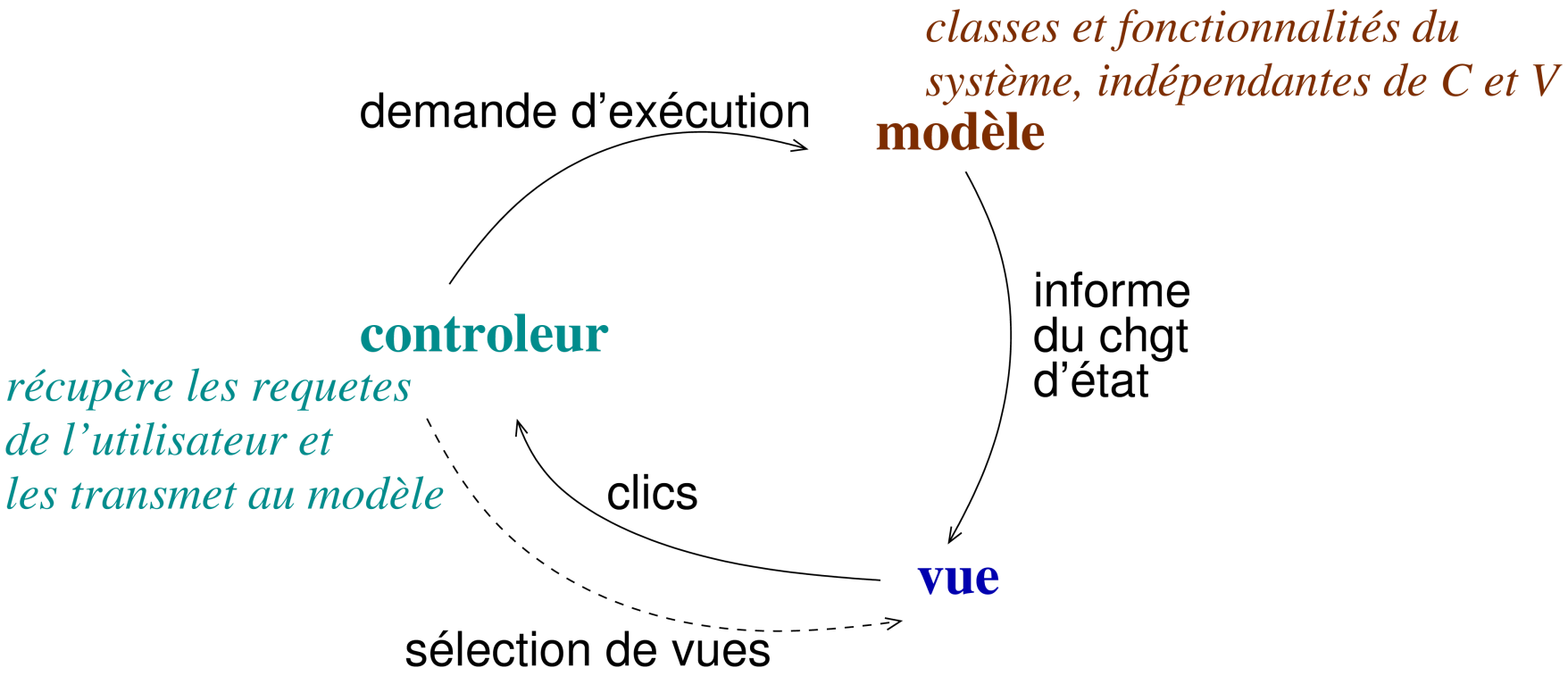
- Fréquemment utilisée pour les interfaces graphiques utilisateurs.
- Historiquement, cette architecture a été introduite dans le langage SmallTalk en 1980.
- C'est un moyen de séparer le traitement des entrées, des sorties et des fonctionnalités principales du logiciel.
- Le principe de cette architecture est de découper le logiciel (ou un morceau du logiciel) en trois parties :
    - le « modèle » ;
    - la « vue », qui correspond au traitement des sorties ;
    - le « contrôleur », qui correspond au traitement des entrées.

Pourquoi MVC ?

Patron de conception utilisé pour des applications reposant sur :

- une *interface graphique*
- un modèle des données manipulées
- un besoin de contrôleur centralisé

Schéma MVC


classes et fonctionnalités du
système, indépendantes de C et V
demande d'exécution
modèle
informe
du chgt
d'état
controleur
récupère les requetes
de l'utilisateur et
les transmet au modèle
clics
vue
sélection de vues

Modèle

- La partie « modèle » comporte les classes principales correspondant aux différentes fonctionnalités de l'application : données et traitements.
- Cette partie,
  - est indépendante des parties « vue » et « contrôleur »,
  - effectue des actions en réponse aux demandes de l'utilisateur (par l'intermédiaire de la partie « contrôleur »), et
  - informe la partie « vue » des changements d'états du modèle pour permettre sa mise à jour.

Vue-Contrôleur

Vue

- La partie « vue » comporte les classes relatives à l'interface graphique (ce que l'utilisateur voit).
- Cette partie est informée des changements d'états du modèle et est mise à jour lors de ces changements d'états.

Contrôleur

- La partie « contrôleur » récupère les actions de l'utilisateur (clics sur les boutons et appuis sur les touches) et associe à ces
  événements des actions qui modifient le modèle.

Avantages d'une architecture MVC

Vues multiples

- Plusieurs vues du même modèle.

Portabilité

- Portage de l'interface sur d'autres plates-formes possible sans modifier le code du noyau de l'application.

Evolution

- Ajout de nouvelles fonctionnalités relatives à l'interface graphique (ajout de lignes dans un menu ou l'ajout de boutons) plus facile.
- Modifications possibles à l'exécution : de nombreux logiciels permettent la personnalisation de leur interface à l'exécution.

Frameworks

- Plusieurs frameworks de développement (Php/Zend ou JEE/Struts) sont fondés sur une variante de cette architecture MVC.

  Vue $\Longleftrightarrow$ Contrôleur $\Longleftrightarrow$ Modèle

- La vue envoie les entrées au contrôleur, qui effectue les traitements en modifiant le modèle.
- Le contrôleur produit des sorties qui sont affichés sur la vue.
- Les échanges directs entre la vue et le modèle sont interdits.
- Seul le modèle peut interagir avec une éventuelle base de données.

Conception objet

Généralités de conception

- Le but de cette étape est de réaliser la conception détaillée du logiciel.
- Un travail important de cette étape consiste à
  - réaliser un diagramme de classes de conception.
  - Il s'agit d'un diagramme de classes logicielles, inspiré du diagramme de classes d'analyse.
  - Par rapport au diagramme de classe d'analyse, des classes, des attributs et des associations peuvent être ajoutés, modifiés, voire supprimés.
  - Le diagramme de classes logicielles peut ensuite être traduit dans un langage de programmation objet.

Affectation des responsabilités

- Une activité de cette étape consiste à affecter les responsabilités aux objets.
- Les responsabilités sont de deux types : connaissance et comportement.
  - Les connaissances peuvent être :
    - la connaissance de données encapsulées ;
    - la connaissance d'objets connexes ;
    - la connaissance d'éléments qui peuvent être dérivés ou calculés.
  - Les comportements peuvent être :
    - la réalisation d'une action, comme effectuer un calcul ou créer un objet ;
    - le déclenchement d'une action d'un autre objet ;
    - la coordination des activités d'autres objets.

Principes

Principes de conception

- Pour réaliser une bonne conception, on applique certains principes de conception.

Principe de l'expert

- Une tâche est effectuée par un objet qui possède, ou a accès à, l'information nécessaire pour effectuer cette tâche.

Principe du créateur

- On affecte à la classe B la responsabilité de créer des instances de la classe A s'il existe une relation entre A et B (typiquement, si B est une agrégation d'objets de A, ou si B contient des objets de A). L'idée est de trouver un créateur qui est connecté à l'objet créé.

Principe de faible couplage

- Le couplage est une mesure du degré selon lequel un élément est relié à d'autres éléments.
- Une classe faiblement couplée s'appuie sur peu d'autres classes.
- Une classe fortement couplée a besoin de connaître un grand nombre d'autres classes. Les classes à couplage fort ne sont pas souhaitables car :
    - elles sont plus difficiles à comprendre ;
    - elles sont plus difficiles à réutiliser, car leur emploi demande la présence de nombreuses autres classes ;
    - elles sont plus sensibles aux variations, en cas de modification d'une des classes auxquelles celle-ci est liée.
- Le principe de faible couplage consiste minimiser les dépendances.

Principe de forte cohésion

- La cohésion est une mesure des liens entre les tâches effectuées par une classe.
- Une classe a une faible cohésion si elle effectue des tâches qui ont peu de liens entre elles, et dont certaines auraient du être affectées à d'autres classes. Il est préférable d'avoir des classes fortement cohésives car elles sont plus faciles à comprendre, à maintenir, à réutiliser.
- Elles sont moins affectées par une modification.

Principe du contrôleur

- Consiste à affecter la responsabilité du traitement des événements systèmes (événements générés par un acteur externe) dans une ou plusieurs « classes de contrôle » (des « contrôleurs »).
- Exemple pour une interface graphique :

  Les événements reçus par l'interface sont délégués à un contrôleur. Permet de séparer la logique de l'application de l'interface : la logique de l'application ne doit pas être gérée par la couche interface.

Utilisation de diagrammes UML

Diagrammes de classes logicielles

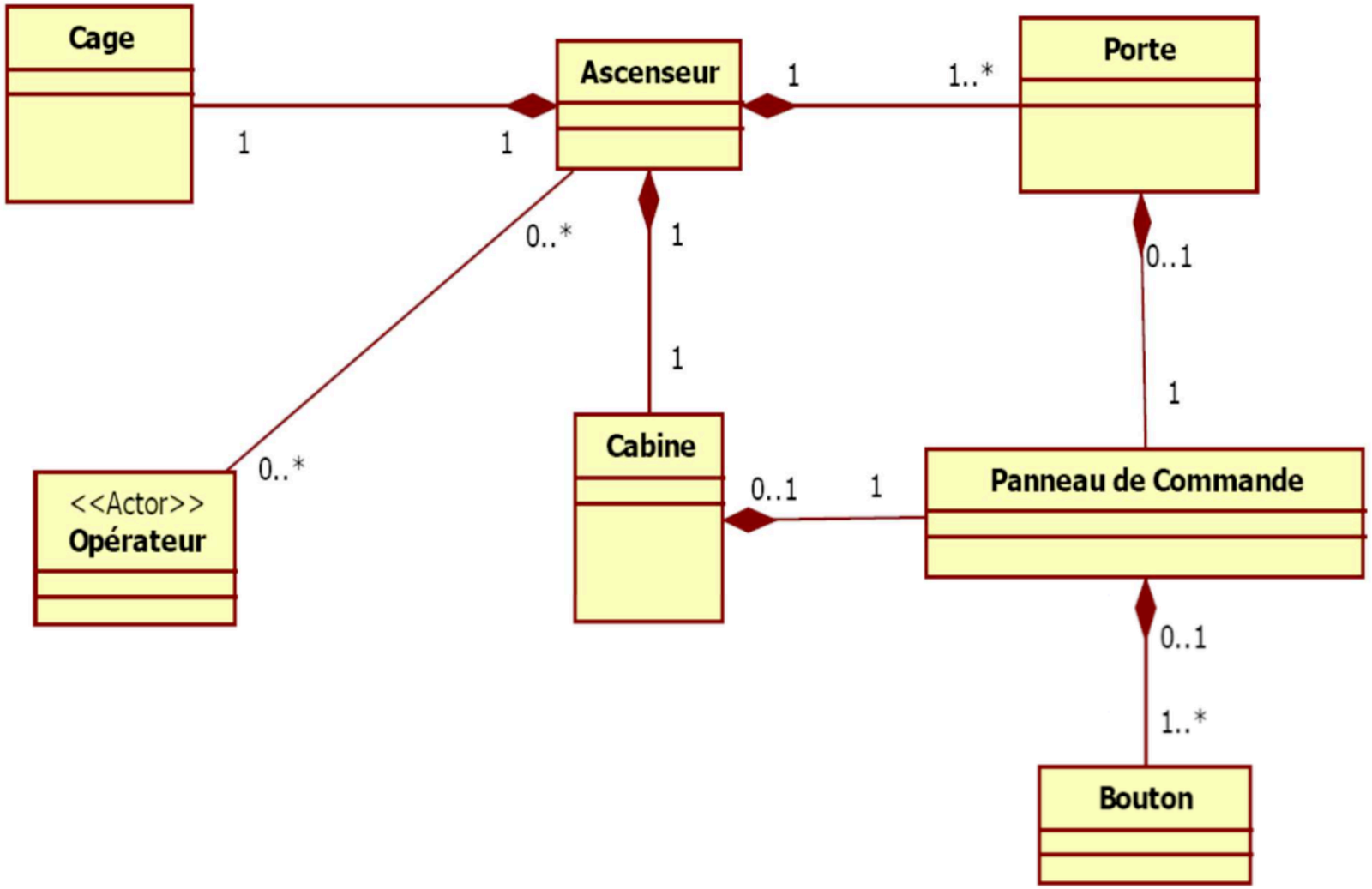
- Le diagrammes des classes logicielles est inspiré du diagramme de classes correspondant au modèle du domaine.
- Des classes peuvent être ajoutées, des liens ajoutés ou modifiés.
- On définit pour chaque classe les opérations qu'elle contient. On peut également donner des diagrammes d'objets montrant des configurations typiques pouvant être construites.

Exemple : description

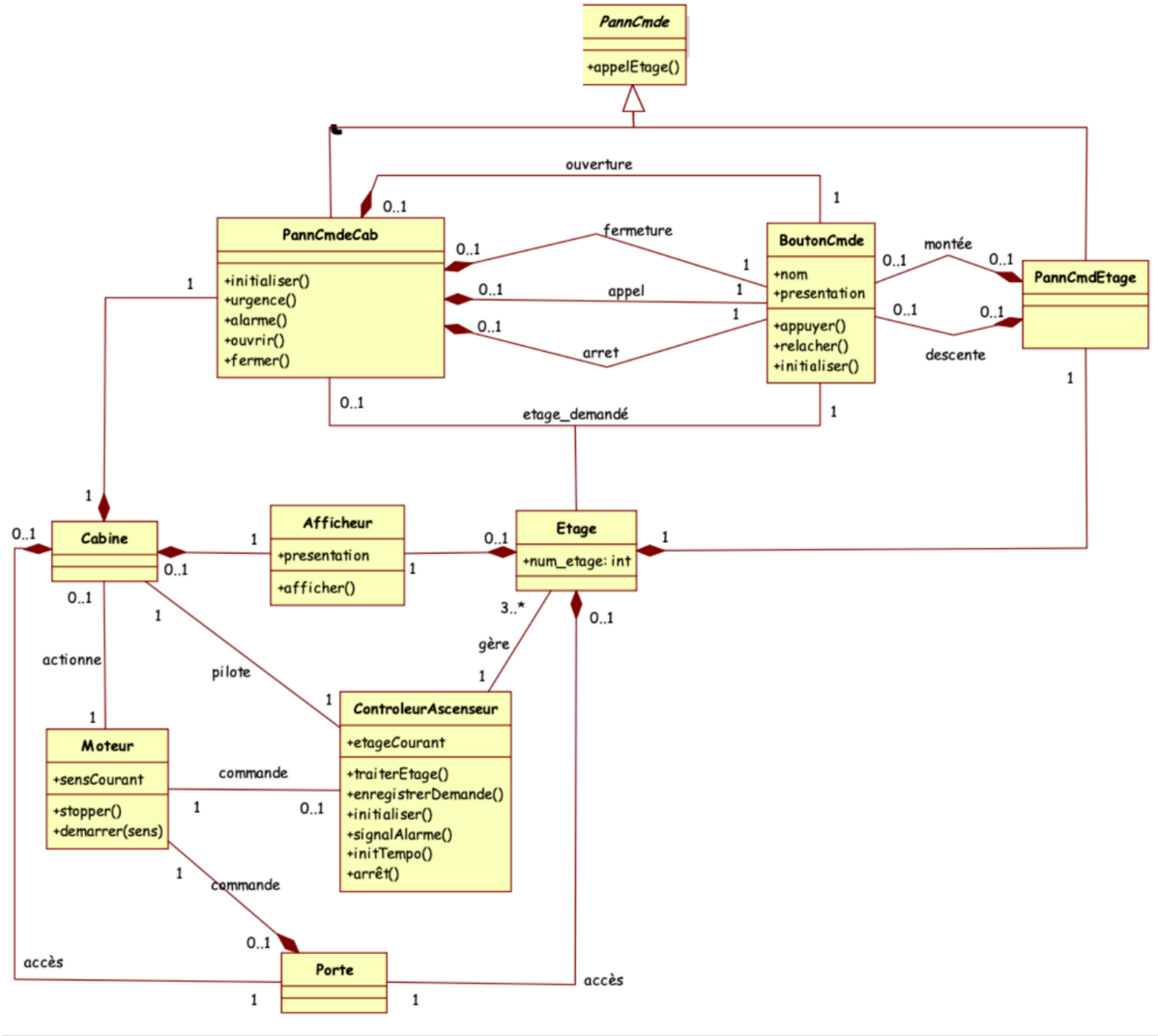
Voici une description simplifiée d'un ascenseur et de son fonctionnement :

Un ascenseur est composé d'une cabine, d'une cage et de portes à chaque étage. La cabine comporte un panneau de commande avec un bouton pour chaque étage, un bouton de commande d'ouverture de porte, un bouton de commande de fermeture de la porte, une commande d'arrêt et un bouton d'appel urgent. A chaque porte d' étage se trouve un panneau de commande avec deux bouton d'appels : montée et descente. Le fonctionnement de l'ascenseur n'est pas décrit dans la documentation que nous avons trouvée. Nous supposons alors que le fonctionnement est celui habituellement rencontré. Un opérateur remet le système en route en cas de panne.

Exemple : Diagramme de classes d'analyse



Exemple : Diagramme d'objets



Diagrammes de séquences et/ou de collaboration

- Les diagrammes de séquence ou de collaboration illustrent la façon dont les objets collaborent pour réaliser une fonctionnalité.
- Ces diagrammes permettent d'illustrer et de motiver les choix d'affectation de responsabilités aux objets.

Diagrammes d'états/transitions

- Les diagrammes d'états-transitions permettent de spécifier le comportement des objets d'une classe.
- Ces diagrammes peuvent être considérés comme généralisant les diagrammes de séquence et de collaboration car ils doivent décrire tous les comportements, et non seulement quelques comportements possibles.

cohérence entre différents diagrammes

- Il est important de s'assurer de la cohérence entre les différents diagrammes élaborés au niveau de la conception. En effet, la mise en oeuvre est a priori très proche de ces modèles.
- Cela suppose d'effectuer certaines vérifications.
- Les vérifications contextuelles consistent à vérifier que les différents éléments de modélisation (objets, attributs, événements, opérations, associations, rôles. . . etc.) sont correctement déclarés et utilisés.

Exemple 1

Si on dispose de diagrammes d'objets, il faut vérifier que chaque diagramme d'objets est cohérent avec le diagramme de classes associé :

- correction des attributs de chaque objet ;
- correction des liens entre les objets ;
- respect des multiplicités.

Exemple 2

On considère une classe A, à laquelle est associé un automate comportant une transition étiquetée par :

Il faut vérifier que :

- m1 est une méthode de la classe A;
- o est un objet, d'une certaine classe B, accessible depuis un objet de la classe A (attribut de A de type B, ou lien entre les deux classes A et B, ayant pour nom de rôle o) ;
- m2 est une méthode de la classe B.

La prochaine fois

Patrons de conception