

Contrôle Long UML

Pierre Gérard

pierre.gerard@univ-paris13.fr

DUT Informatique S2

Université de Paris 13

Résumé

Ce contrôle dure 3 heures. Aucun document n'est autorisé. Observez la plus grande rigueur dans les notations. Si vous êtes amenés à émettre des hypothèses, veuillez les expliciter sur la copie. Le barème est donné à titre indicatif.

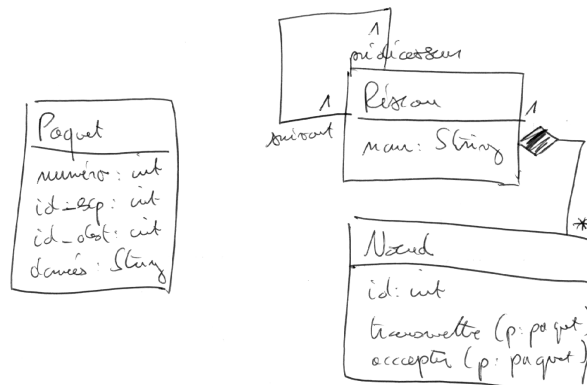
1 Diagramme de classes et d'objets (4 pts)

Lorsque des données sont transmises par un réseau, elles sont découpées en paquets qui contiennent chacun une partie des données. Un paquet comporte :

- le numéro du paquet ;
- l'identifiant de l'expéditeur ;
- l'identifiant du destinataire ;
- les données transmises.

Un réseau en anneau est composé de noeuds reliés les uns aux autres par des relations précédent/suivant : chaque noeud ne connaît que le noeud qui le suit dans la chaîne ainsi celui qui le précède. La chaîne est fermée pour former un anneau. Un réseau est identifié par son nom et chaque noeud possède un identifiant unique dans le réseau. Les paquets sont émis par leur expéditeur et transitent de noeud en noeud suivant jusqu'à arriver à leur destinataire. Chaque noeud peut se voir transmettre un paquet par son prédécesseur dans l'anneau. S'il en est le destinataire, il peut l'accepter.

Question : Donnez un diagramme de classes pour modéliser les réseaux en anneau tels qu'ils sont décrits ci-dessus. Utilisez votre bon sens pour déterminer les multiplicités.

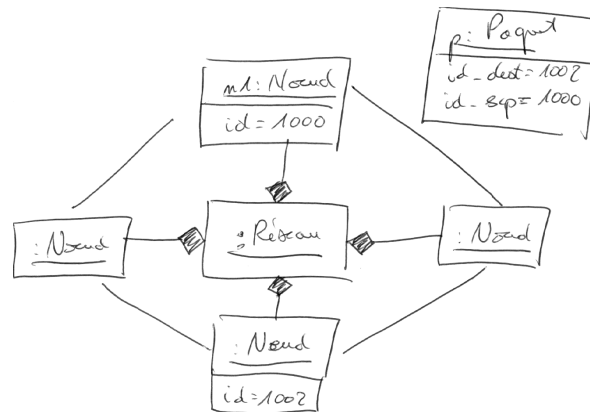


Ooops, l'association réflexive, c'est autour de Noeud. Barème :

- 0.25 Classe paquet
- 0.25 Attributs de paquets (types optionnels)
- 0.25 Classe Réseau avec attribut
- 0.25 Noeud avec id
- 0.25 Association entre réseau et noeud
- 0.25 Composition de l'association
- 0.25 Multiplicités
- 0.25 Transmettre() et Accepter()

- 0.25 Association réflexive
 0.25 Multiplicités de l'assoc réflexive
 -0.5 si non respect des notations (pointillés, ...)

Question : Proposez un diagramme d'objets conforme à la description ci dessus. Ce diagramme devra comporter au moins 6 objets, avec au moins un de chaque classe.



Barème :

- 0.25 Conformité avec le diag de classes
 0.25 Représente un réseau, indépendamment des erreurs sur le diag de classes (cad en quelque sorte conformité avec le diag de classes juste)
 0.25 Respect des contraintes (6 objets, 1 de chaque classe)
 0.25 Valeurs d'attributs
 0.25 Soulignés
 0.25 Deux points devant le nom des classes

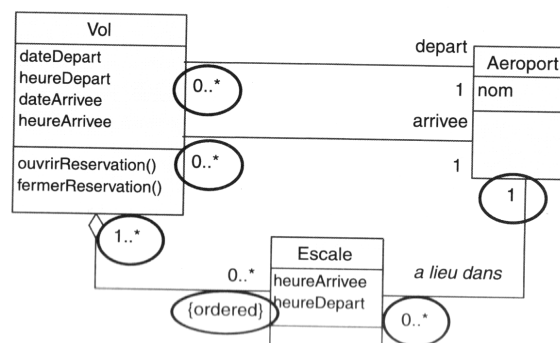
2 Diagramme de classes (3 pts)

Des interviews d'experts métier ont permis de mettre en évidence les éléments suivants :

- Un vol a un aéroport de départ et un aéroport d'arrivée ;
- Un vol a une heure de départ et une heure d'arrivée, ainsi qu'une date de départ et une d'arrivée ;
- Un vol peut comporter des escales dans des aéroports ;
- Les escales interviennent dans un ordre déterminé ;
- Une escale a une heure d'arrivée et une heure de départ ;
- Chaque aéroport a un nom ;
- On peut ouvrir (et fermer) à la réservation chacun des vols.

Question : Proposez deux solutions pour modéliser les éléments ci-dessus :

- Un diagramme de classe sans classe-association ;

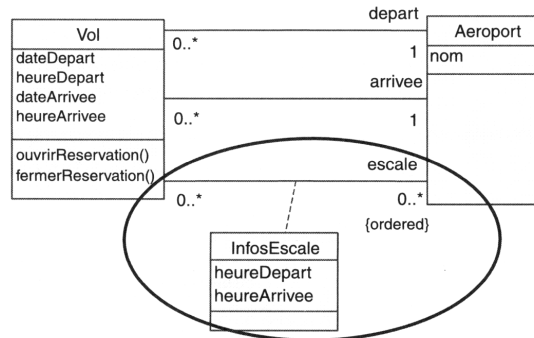


Barème :

- 0.25 Classe Vol avec attributs
 0.25 Ouvrir et fermer réservation comme opérations
 0.25 Classe Aéroport avec attribut nom

- 0.25 Assoc départ et arrivée avec multiplicités
- 0.25 Classe Escale avec attributs
- 0.25 Assoc Vol-Escale avec multiplicités
- 0.25 Agrégation (ou composition)
- 0.25 Ordered
- 0.25 Assoc Escale-Aéroport avec multiplicités

– Un autre avec une classe association.



Barème (Peu importe si les attributs ne sont pas recopiés) :

- 0.5 Transformation de Escale en classe assoc
- 0.25 Multiplicités

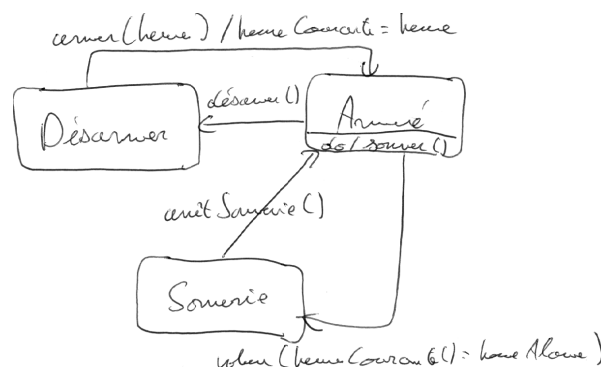
3 Diagramme d'états/transitions (3 pts)

Considérons un réveille-matin simplifié :

- On peut armer ou désarmer l'alarme;
- Quand l'heure courante devient égale à l'heure de l'alarme, le réveille sonne;
- On peut interrompre la sonnerie;
- La sonnerie s'interrompt automatiquement après 30 minutes.

Un réveil matin dispose des opérations armer(heure), désarmer(), arrêtSonnerie(), sonner() et heureCourante(). Il dispose d'un attribut heureAlarme.

Question : Proposez un diagramme d'états/transitions pour modéliser la dynamique d'un réveil-matin.



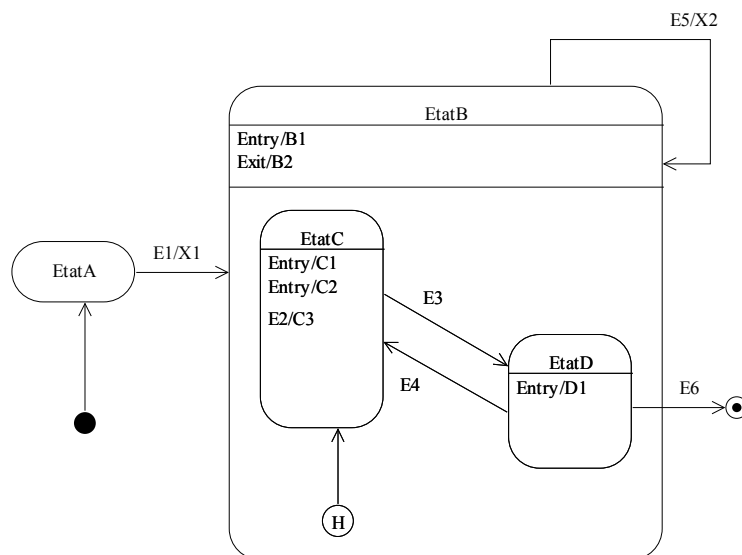
Barème :

- 0.25 Etat Armé
- 0.25 Etat Désarmé
- 0.25 Etat Sonnerie
- 0.25 Transition Armer
- 0.25 Evenement Armer bien écrit (pas de when(...) ou quoi)
- 0.25 Activité heureCourante=heure
- 0.25 Transition Désarmer
- 0.25 Evenement Désarmer bien écrit (pas de when(...) ou quoi)

- 0.25 Transition ArrêtSonnerie
- 0.25 Evenement ArrêtSonnerie bien écrit (pas de when(...) ou quoi)
- 0.25 Transition When
- 0.25 Événement When bien écrit, avec condition bien foutue entre parenthèse
- 0.5 si des états carrés etc...

4 Diagramme d'états hiérarchique (2 pts)

Considérons le diagramme d'états suivant :



Question : Considérant que l'état courant est « Etat A », quelles activités sont produites par la séquence d'événements E1, E2, E3, E5, E3, E4, E6. Présentez votre réponse dans un tableau semblable à celui ci-dessous.

Événement	Activités résultantes	Nouvel état
E1		
E2		
...		

Événement	Activités résultantes	Nouvel état
E1	X1, B1, C1, C2	EtatC
E2	C3	EtatC
E3	D1	EtatD
E5	B1, X2, B1, D1	EtatD
E3	rien	EtatD
E4	C1, C2	EtatC
E6	rien	EtatC

Barème :

- 0.25 pour chaque ligne dans la colonne "Activités résultantes" si tout est bon. Si C1 seulement ou C2 seulement au lieu de C1,C2, tout compter
- 0.25 pour avoir mis les nouveaux états à peu près correctement

5 OCL (4 pts)

Considérons les contraintes OCL suivantes :

```
// contrainte 1
context Compte : debiter(somme : int)
pre: somme > 0
```

```

    post: solde = solde@pre - somme

// contrainte 2
context Compte
    inv: banque.clients -> includes (propriétaire)

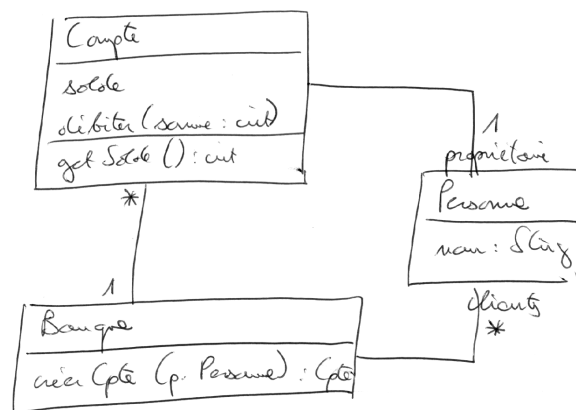
// contrainte 3
context Compte::getSolde() : int
    post: result = solde

// contrainte 4
context Banque :: créerCompte(p : Personne) : Compte
    post: result.ocIsNew() and
        compte = compte@pre -> including(result) and
        p.compte = p.compte@pre -> including(result)

// contrainte 5
context Personne
    inv: Personne.allInstances() -> forAll(p1, p2 |
        p1 <> p2 implies p1.nom <> p2.nom)

```

Question : Proposez un diagramme de classes compatible avec ces contraintes OCL



Barème (2) :

- 0.25 Classes Compte, Personne et Banque
- 0.25 Solde avec type int
- 0.25 Nom (avec ou sans type)
- 0.25 Opérations débiter, getSolde, créerCpte
- 0.25 Assoc Cpte-Personne
- 0.25 Assoc Banque-Personne
- 0.25 Rôles propriétaire ou client (un seul suffit pour avoir 0.25)
- 0.25 Assoc Compte-Banque

Question : Que signifient chacune de ces contraintes ?

Barème (1.25) :

- 0.25 par contrainte bien comprise (ne pas être trop à cheval sur le français, mais chercher à comprendre si la contrainte est comprise)

Question : Ecrire des contraintes OCL pour les énoncés suivants :

- Le solde d'un compte doit toujours être positif ou nul ;
- context Compte inv : solde > 0
- Si une personne possède au moins un compte bancaire, alors elle est cliente d'au moins une banque.

context *Personne inv : compte -> notEmpty()* implique *banque -> notEmpty()*

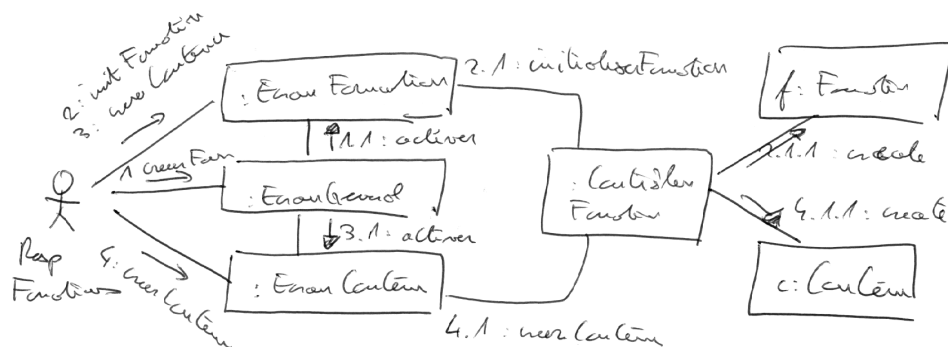
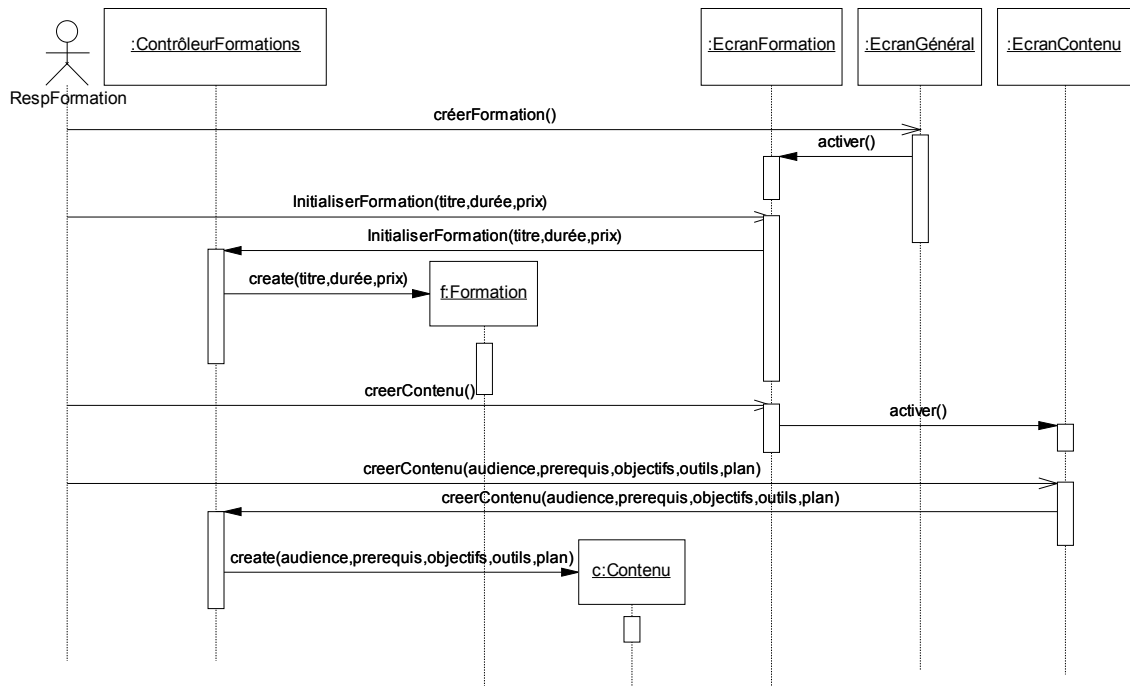
Barème (0.75) :

0.25 pour la première contrainte

0.5 pour la seconde contrainte

6 Diagrammes de séquences et de communication (2 pts)

Question : Transformez le diagramme suivant en un diagramme de communication équivalent.



Barème :

0.25 Reprise des lignes de vie et acteur

0.5 Liens entre lignes de vie

0.25 Pas de flèche directement sur les lignes de vie

0.25 Flèches à part, synchrones ou asynchrones selon le cas

0.25 Noms des messages (certains sont longs avec plein de paramètres, autoriser les abréviations)

0.25 Ordre des numéros

0.25 Hiérarchisation des numéros

7 Questions de cours (2 pts)

Barème :

1 pour chaque question

Question : Suivant la terminologie employée dans la méthode utilisée dans le projet UML (Monoply), décrivez les différents types de classes participantes. Expliquez le principe présidant à cette distinction entre types de classes.

Types de classes participantes :

- *Interfaces : permettent les interactions entre les utilisateurs et l'application ;*
- *Contrôle : pour définir la dynamique de l'application indépendamment de l'IHM (appli, web...)*
- *Métier : concepts invariants du domaine d'application*

Les classes participantes (ou d'analyse) sont organisées en couches (contrôle au milieu), si bien qu'on rend par exemple l'IHM indépendante des algorithmes et des données.

Question : Que sont les Design Patterns ? A quoi sont-ils utiles ?

Un patron de conception (design pattern) est la description d'une solution classique à un problème de conception récurrent. Les design pattern permettent

- *une bonne capitalisation de l'expérience ;*
- *une conception beaucoup plus rapide ;*
- *l'élaboration de constructions logicielles de meilleure qualité grâce à un niveau d'abstraction plus élevé ;*
- *la réduction du nombre d'erreurs, d'autant que les patrons sont examinés avec attention avant d'être publiés ;*
- *une communication plus aisée ;*
- *d'écrire du code facilement compréhensible par les autres ;*
- *d'apprendre en suivant de bons exemples.*