Chapitre 4 : Diagrammes de séquences et de collaborations
Éléments de correction

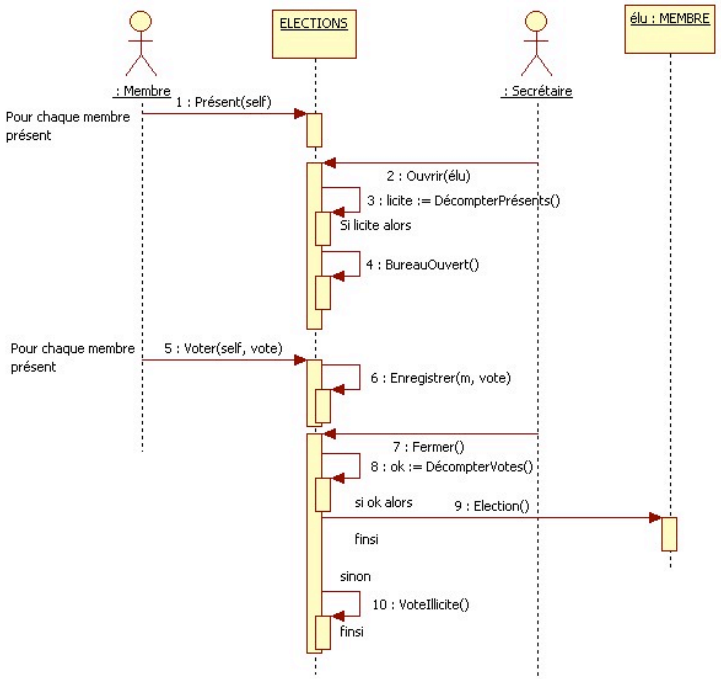
Il s'agit d'UN corrigé. D'autres réponses sont possibles, notamment là où une modélisation est demandée.

Partie I : Modélisation à l'aide de séquences

Exercice 4.1

Les élections sont sommairement décrites. Selon le texte, on considère l'élection individuelle des membres actifs, par un vote à la majorité absolue. Le vote est licite si au moins les deux tiers des membres sont présents. Un membre ne vote qu'une fois à chaque scrutin. Il y a quatre possibilités de vote pour un scrutin donné : oui, non, blanc, abstention.

Deux acteurs et deux objets suffisent à la représentation de l'interaction :



Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)

Voici les hypothèses complémentaires que nous avons dû prendre :

Les votes pour élire un membre (l'élu) ne peuvent avoir lieu que si le bureau est ouvert. Cette ouverture est faite par le secrétaire, après comptage des membres présents. S'il y a au moins deux tiers de membres, le vote est déclaré licite. Quand un membre de l'association se présente pour voter, il y a d'abord vérification de sa situation. En effet, il ne peut voter qu'une fois par vote. Ensuite, son vote est mémorisé : blanc, oui, non ou abstention.

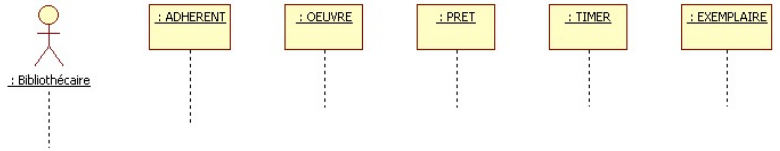
À la clôture du bureau de vote, le nombre de votes positifs est calculé. Le nombre de voix attribués au candidat est supérieur à la moitié des votes, le membre élu est déclaré membre actif de l'association.

Pour faire comptabiliser, par le système, les présents et les votants on mémorise ces informations, relatives à une élection par deux relations à durée de vie temporaire.

Exercice 4.2

a) Quelles sont les classes qui apparaissent dans le scénario 1 (*Emprunt effectué par un adhérent*) ? Nous pouvons en énumérer cinq : *ADHERENT*, *OEUVRE*, *PRET*, *EXEMPLAIRE*, *TIMER*. L'énoncé mentionne explicitement un objet externe au système, *unBibliothécaire*. Nous tenons là « notre » acteur externe. Il n'est pas impossible qu'il y en ait un autre, *unAdhérent*, qui déclenche tout le traitement. Nous ne le ferons toutefois pas figurer dans notre schéma en tant qu'acteur.

Le diagramme de séquences que nous devons produire va donc ressembler à ceci :

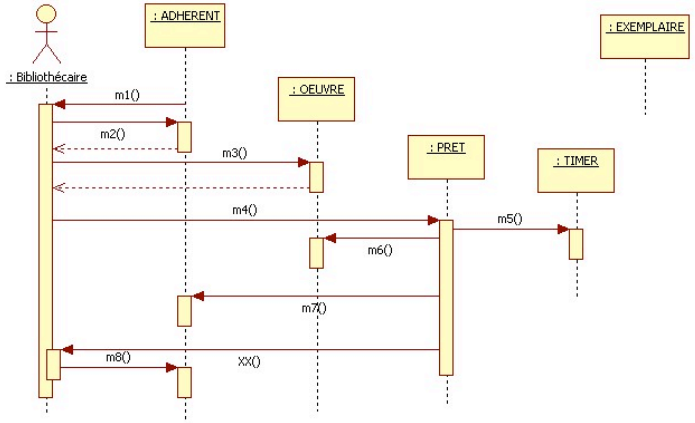


Pour simplifier ce schéma, nous allons attribuer un code à chaque méthode, selon la liste suivante :

m1 : demanderEmprunt (obj : Œuvre)
m2 : nombreExemplairesEmpruntés ()
m3 : exemplairesDisponibles ()
m4 : initialiserPrêt (unExemplaire : Exemplaire)
m5 : lancerTimer ()
m6 : décrémenterExemplairesDisponibles ()
m7 : incrémenterNbEmprunts ()
m8 : attribuer (unExemplaire : Exemplaire)

NB : m5 s'est vu attribuer un nom au hasard, le sujet n'en comportant pas.

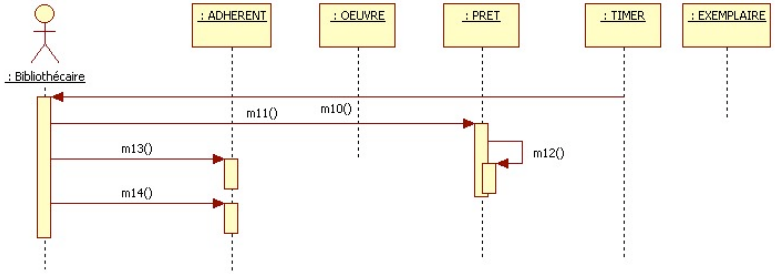
Nous pouvons maintenant compléter le diagramme de séquences :



Trois remarques, pour finir :

1) Dans tout le schéma, nous avons donné le même nom au message et à la méthode du receveur dudit message SAUF pour m1.
2) Le message XX n'est pas mentionné dans le texte. Il est pourtant nécessaire pour assurer la continuïté du flux de commande et le respect du texte.
3) La classe *EXEMPLAIRE* ne participe pas à ce scénario. Elle est mentionnée dans l'énoncé de façon indirecte (on parle d'objet *unExemplaire*) mais elle ne reçoit aucun message, pas plus qu'elle n'en envoie. Une erreur dans le sujet ???

b) Si nous appliquons le même procédé pour le scénario 2 (Cas de litige avec un adhérent), nous obtenons le diagramme de séquences suivant :



Là encore, nous avons codé les différents messages qui circulent d'une classe à l'autre selon la liste suivante :

m10 : litige ()

Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)

m11 : miseEnLitige ()
m12 : envoyerAvertissement ()
m13 : imprimerCourrierLitige ()
m14 : interdireEmprunt ()

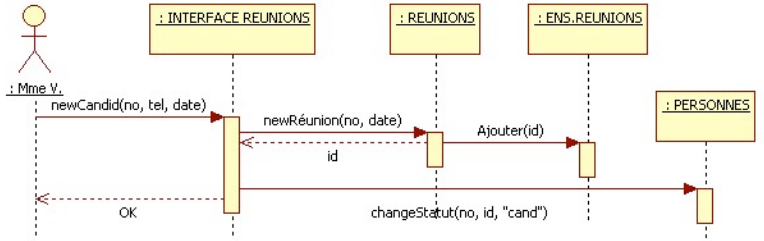
Exercice 4.3

Un diagramme de séquences décrit les échanges entre un acteur externe et le système (des classes du système). Il ne décrit pas les échanges entre les acteurs eux-même. Le calcul du chiffre d'affaires réalisé lors de la réunion et la recherche des candidates sont « affaire » de dialogues entre les acteurs. Nous ne les modéliserons pas, pas plus que nous ne modéliserons le choix des bijoux. Par contre, nous allons décrire ce qui se passe lors de l'enregistrement de ces données dans le système. Ceci se fera en deux temps. Le premier sera consacré à l'enregistrement des candidates à l'organisation des prochaines réunions, avec éventuellement une date de réunion et le numéro de téléphone de la personne. Le second correspondra à l'enregistrement des bijoux choisis par l'hôtesse et, éventuellement, la mémorisation du nombre de points conservés.

La partie consacrée à la saisie de la ou des nouvelles candidates va nécessiter la fourniture des informations relatives à ces personnes (nom, prénom ou numéro, selon les usages de Mme V.) et éventuellement de leur numéro de téléphone (ceci n'est vraiment nécessaire que pour les hôtesses). Une nouvelle réunion doit être créée et reliée à la candidate, ceci pour chacune des candidates déclarées.

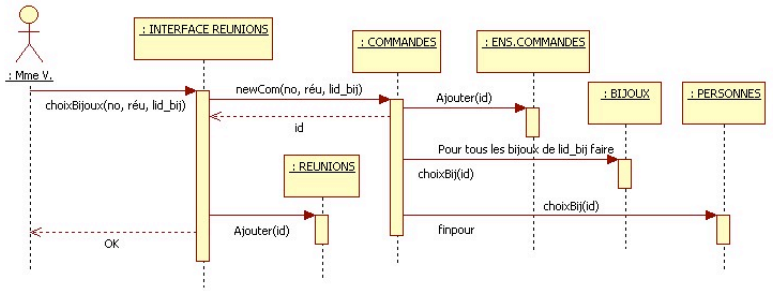
Le diagramme de séquences que nous fournissons ici (voir ci-après) correspond à la saisie d'une candidate déjà connue. Nous avons enrichi le diagramme de classes de la question a d'une classe *ENS.REUNIONS*, chargée de la gestion des occurrences de réunions.

Ceci se représente ainsi :



La liste des bijoux choisis par l'hôtesse sera ensuite fournie, ceux-ci enregistrés, le nombre total de points équivalent calculé et renvoyé à l'hôtesse. S'il y a différence (nous n'envisagerons pas le cas d'un choix représentant un nombre de points supérieur à ce à quoi avait droit l'hôtesse), le système mémorisera ce nombre de points restants (dans la classe *HOTESSE*) et le signalera à l'hôtesse.

Le diagramme de séquences que nous fournissons ajoute une nouvelle commande et la lie aux différents bijoux choisis, à la réunion à la suite de laquelle la commande a été passée et aux différents bijoux choisis. Il correspond au cas où le nombre de points choisis est égal à celui gagné.



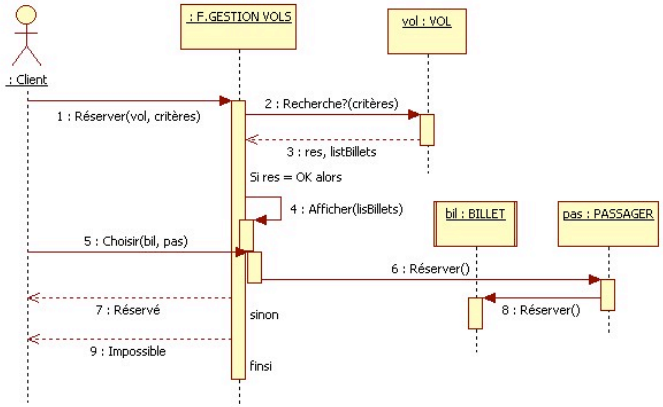
Le cas d'un nombre de points supérieur ou celui d'une commande inférieure à ce qu'a gagné l'hôtesse ne sont que des variantes de celui que nous avons fourni. Nous ne les développerons pas davantage et laissons aux étudiants le soin de le faire. Une bonne séance de révision, en quelque sorte…

Le diagramme de classes de la question a doit, bien entendu, être modifié pour le rendre cohérent avec ces deux diagrammes de séquences.

Exercice 4.4

Le schéma de données que nous avons est un peu « frustre ». Il va nous falloir le compléter. Nous devons également faire une hypothèse relative aux billets. Ceux-ci semblent créés dès la planification des vols, avant même de savoir s'il y aura des clients pour les acheter. Au début (des vols), un billet correspond à une place à vendre. À la fin (des vols), un billet = un voyageur.

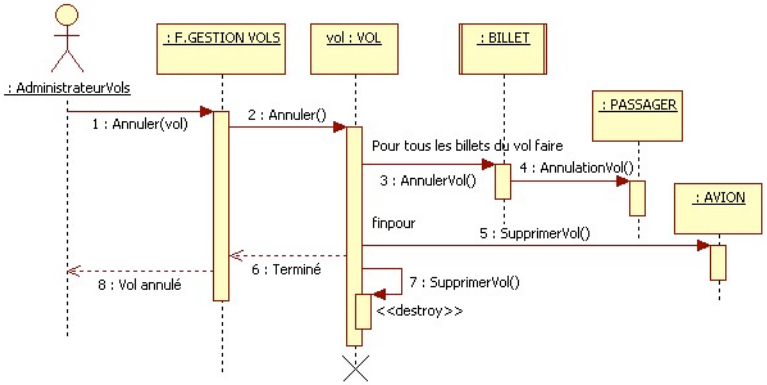
Cette hypothèse posée, réserver un billet revient à marquer un des billets « réservé » et à en informer la classe PERSONNE.



Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)

Pour ce qui concerne l'annulation d'un vol, rien n'est dit quant au devenir des billets (sont-ils détruits ?) et du vol (est-il détruit ?). Nous allons supposer que les billets sont conservés et que les vols sont effectivement détruits. Cette « décision », pas très cohérente, est purement pédagogique. Elle nous permet de visualiser sur un même schéma un message de « suicide » (il s'agit de *SupprimerVol*) et un message ne demandant pas un tel acte (*AnnulerVol*) :



Partie II : Modélisation à l'aide de collaborations

Exercice 4.5

NB : nous avons choisi volontairement de ne pas recourir à StarUML pour convertir le diagramme de séquences en un diagramme de collaborations et de repartir du texte. Nous avons, par contre, adopté les mêmes hypothèses complémentaires.

Nous avons découpé notre solution en plusieurs schémas :

- le premier permet de décrire le comptage des présents, comptage effectué après l'ouverture du bureau par la secrétaire ;
- le deuxième est celui qui termine le comptage, à l'initiative de la secrétaire. Il permet de savoir si le vote est licite ou non. Il semble évident que ce traitement se place après le précédent.
- le troisième correspond à l'enregistrement du vote d'un membre ;
- le quatrième se positionne à la fermeture du bureau et permet de connaître les membres élus.

Les différents diagrammes sont présentés les uns après les autres :

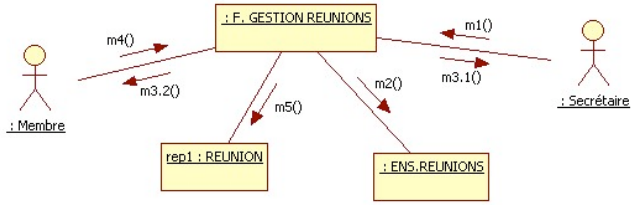


Figure n° 1 : diagramme de collaborations *Comptage des présents*

Les messages sont codés. Ces codes ont la signification suivante :

m1 : Ouvrir (R)
m3.1 : [rep1 = NON] « Réunion inconnue »
m4 : Présent (nom)
m2 : rep1 := Existe? (R)
m3.2 : [rep1 ≠ NON] « Bureau ouvert »
m5 : AjouterPrésent ()

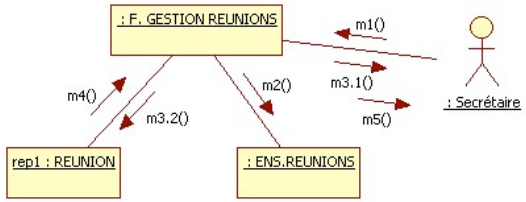


Figure n° 2 : diagramme de collaborations *Vote licite ou non*

Les codes-messages correspondent à ceci :

m1 : CombienPrésents? (R)
m3.1 : [rep1 = NON] « Réunion inconnue »
m4 : rep2
m2 : rep1 := Existe? (R)
m3.2 : [rep1 ≠ NON] rep2 := NbPrésents? ()
m5 : rep2

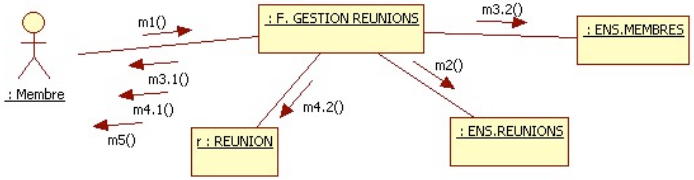


Figure n° 3 : diagramme de collaborations *Enregistrement d'un vote*

Les messages ont la signification suivante :

Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)

m1 : Voter (self, vote, R)
m3.1 : [r = NON] « Réunion inconnue »
m4.1 : [m = NON] « Membre inconnu »
m5 : « a voté ! »
m2 : r := Existe? (R)
m3.2 : [r ≠ NON] m := Existe? (self)
m4.2 : [m ≠ NON] Enregistrer (vote)

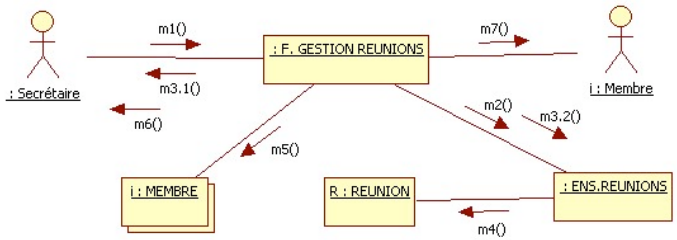


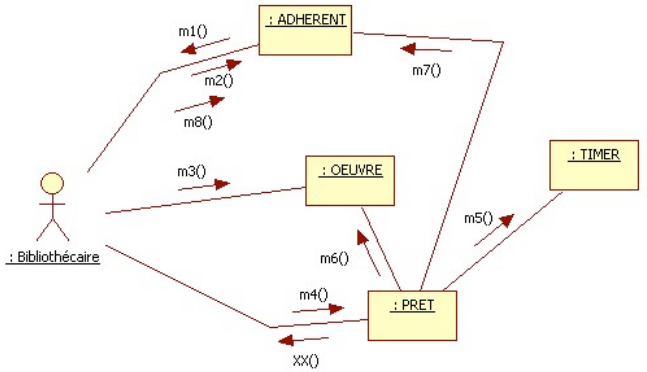
Figure n° 4 : diagramme de collaborations *Résultats des élections*

Les messages correspondent aux échanges suivants :

m1 : Fermer (R)
m3.1 : [rep1 = NON] « Réunion inconnue »
m4 : listElus := Elus? ()
m6 : listElus
m2 : rep1 := Existe? (R)
m3.2 : [rep1 ≠ NON] listElus:= DécompterVotes? (R)
m5 : *[i ∈ listElus] ProclamerElu (R)
m7 : *[i ∈ listElus] « Félicitations ! »

Exercice 4.6

a) Le scénario 1 *Emprunt effectué par un adhérent* implique quatre classes (*ADHERENT*, *ŒUVRE*, *PRET* et *TIMER*) et un acteur.

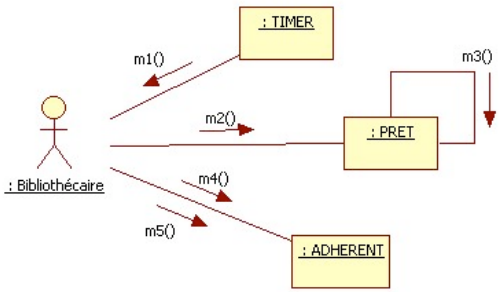


Les messages, pour ceux qui sont connus, ont la signification suivante :

m1 : demanderEmprunt (obj : Œuvre)
m2 : n1 := nombreExemplairesEmpruntés ()
m3 : ex := exemplairesDisponibles ()
m4 : initialiserPrêt (unExemplaire : Exemplaire)
m5 : lancerTimer ()
m6 : décrémenterExemplairesDisponibles ()
m7 : incrémenterNbEmprunts ()
m8 : attribuer (unExemplaire : Exemplaire)

NB : le message XX n'est pas mentionné dans le texte. Il est pourtant nécessaire pour assurer la continuïté du flux de commande et le respect du texte.

b) Le deuxième scénario *Cas de litige avec un adhérent* met en œuvre les mêmes éléments :



Les différents messages (m1, m2…) correspondent à ceci :

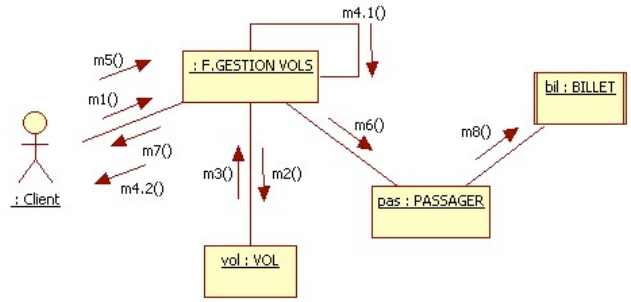
m1 : litige ()
m2 : miseEnLitige ()
m3 : envoyerAvertissement ()
m4 : imprimerCourrierLitige ()
m5 : interdireEmprunt ()

Exercice 4.7

Si nous reprenons dans cet exercice les hypothèses prises pour rédiger la solution de l'exercice 4.4, le diagramme de collaborations décrivant ce qui se passe lors de la réservation d'un billet ressemble à ceci :

Projet ENRC-DVD
MIAGE

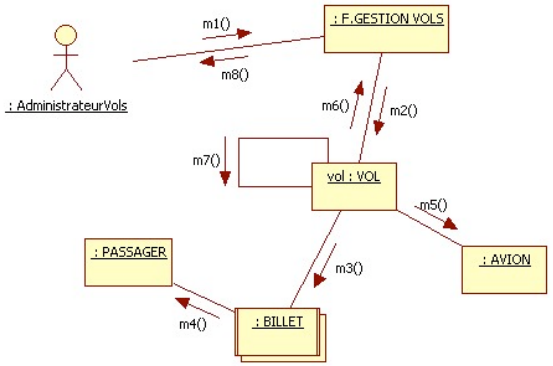
Module UML
(Outils de modélisation)



chaque message ayant la signification suivante :

m1 : Réserver (vol, critères)
m2 : Recherche? (critères)
m3 : res, listBillets
m4.1 : [res=OK] Afficher (listBillets)
m4.2 : [res=NON] « Impossible »
m5 : Choisir (bil, pos)
m6 : Réserver ()
m7 : « Réservé »
m8 : Réserver ()

L'annulation d'un vol se modélise ainsi :



chaque message ayant la « valeur » ci-après :

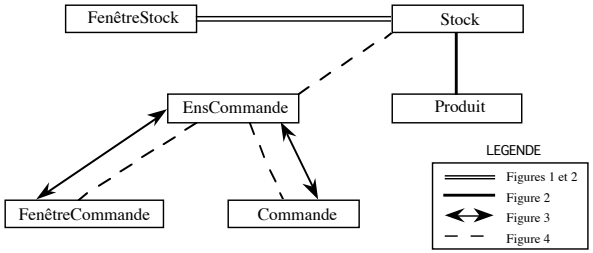
m1 : Annuler (vol)
m2 : Annuler ()
m3 : *[i : 1..n] AnnulerVol ()
m4 : AnnulationVol ()
m5 : SupprimerVol ()
m6 : « Terminé »
m7 : SupprimerVol ()
m8 : « Vol annulé »

Partie III : Cohérence entre séquences, collaborations et classes

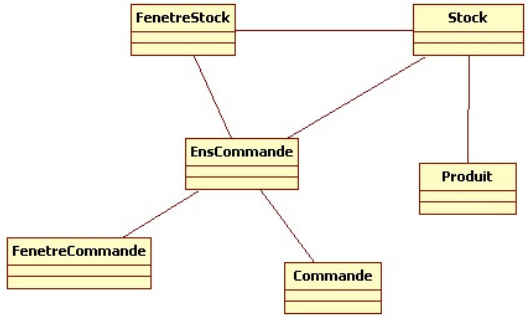
Exercice 4.8

Pour obtenir un diagramme de classes, nous pouvons prendre appui sur les schémas fournis. Toutes les classes citées dans les diagrammes de collaborations ou de séquences doivent figurer dans le diagramme de classes. Nous avons donc au minimum six classes : FenêtreStock, Stock, Produit, FenêtreCommande, EnsCommande, Commande.

Nous savons, en outre, que les messages envoyés d'une classe à l'autre dans un diagramme de séquences ou de collaborations empruntent obligatoirement les relations. Nous avons donc une structure qui ressemble à celle-ci :



Nous pouvons donc déjà prévoir un diagramme de classes composé de six classes structurées ainsi :



La troisième étape va consister à introduire, dans chaque classe, ses opérations. Nous partons d'un principe simple : l'arrivée d'un message X dans une classe Y déclenche l'exécution de l'opération de même nom dans cette classe Y. Les opérations mentionnées dans les quatre figures se répartissent donc ainsi :

Figure n° …	Fenêtre Stock	Stock	Produit	Fenêtre Commande	EnsCommande	Commande
1		nouvelleEntrée() est_référencé()				

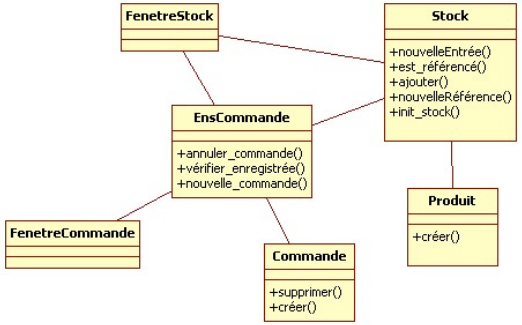
Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)

		ajouter()				
2		nouvelleRéférence() est_référencé() init_stock()	créer()			
3					annuler_commande() vérifier_enregistrée()	supprimer()
4		est_référencé()			nouvelle_commande()	créer()

NB : nous faisons comme hypothèse que les figures sont cohérentes et sans erreur. Sans cette hypothèse, il nous faudrait créer le diagramme *ex nihilo*.

Sur nos six classes, donc, seules quatre sont dotées d'opérations :



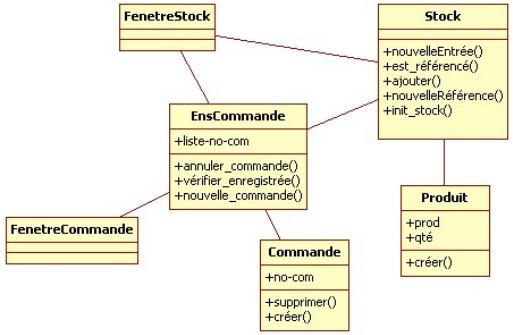
Nous devons ensuite travailler sur les attributs. Il y a deux cas possibles : soit les paramètres des opérations correspondent à des attributs dans les classes, soit il s'agit d'attributs « de travail ». La figure 1 met en évidence deux attributs, *prod* et *qté*. Le premier sert manifestement d'identifiant. Tous les deux sont des attributs de la classe *Stock*.

La figure 2 fait apparaître un attribut *nom* de la classe *Produit*. À la suite de la création du produit, la classe retourne *prod*. Ceci veut très certainement dire :
1) que *prod* est l'identifiant du produit créé,
2) que la classe *Stock* agit comme une classe *EnsProduit*, chargée de veiller sur la « liste » des produits créés.

La figure 3 n'ajoute qu'un seul attribut, *no-com*, « appartenant » à la classe *EnsCommande* ou à un attribut contenant une liste de numéros de commande. Nous faisons l'hypothèse que cet attribut est bel et bien un attribut de la classe *Commande* et donc que la classe *EnsCommande* contient, elle, un attrbut *liste-no-com*.

La figure 4 n'apporte pas plus d'attributs.

Le diagramme des classes quasi-final est donc le suivant :



Il ne reste plus qu'à renseigner les cardinalités des relations… Ensuite, si l'on veut « fignoler », on peut toujours exprimer le fait que la classe *EnsCommande* est reliée à la classe *Commande* par une agrégation, que les classes *FenetreCommande* et *FenetreStock* peuvent hériter d'une même classe *Fenetre*…

Exercice 4.9

a) Il s'agit pour ce premier scénario d'enregistrer un nouvel avion, avec son type (celui-ci étant créé s'il s'agit d'un nouveau type d'avion) et le contrôle subi (celui-ci devant être ajouté). Nous allons adopter les règles que nous avons développées dans nos ouvrages, à savoir :

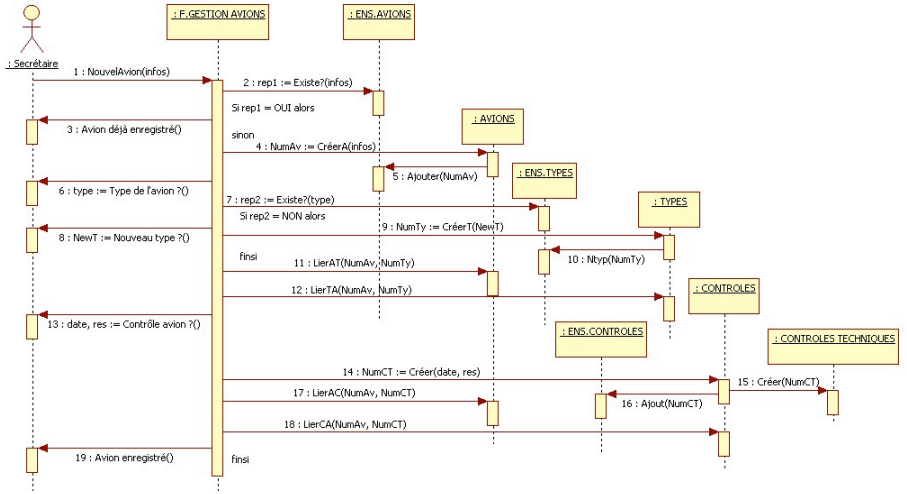
- une classe interface, *F. GESTION AVIONS*, qui se chargera de piloter tout le dialogue ;
- des classes ensembles, dont le rôle sera de tenir à jour la liste des occurrences créées.

Quelles sont, mises à part celles dont nous venons de parler, les classes concernées par ce traitement ? Il y en a quatre, *AVIONS*, *TYPES*, *CONTROLES* et *CONTROLES TECHNIQUES*.

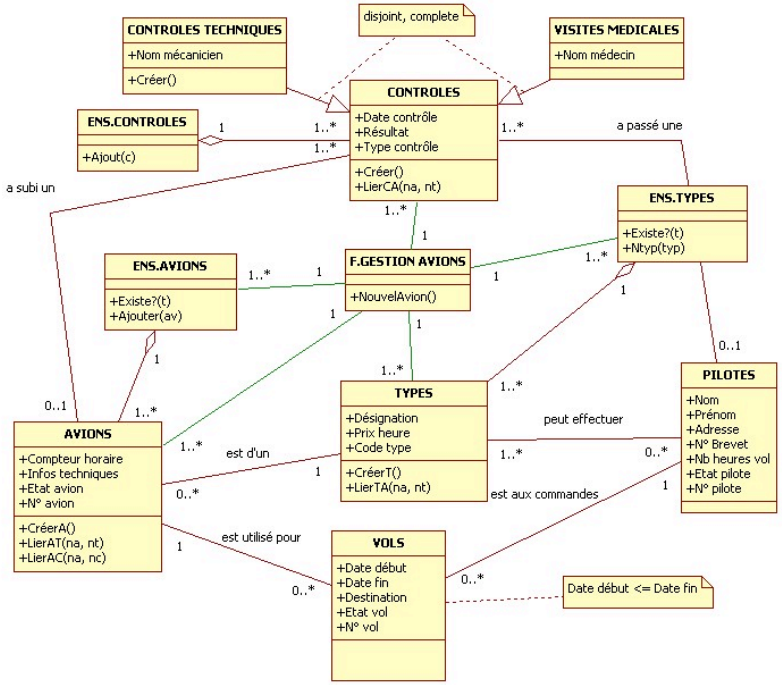
Le diagramme de séquences est le suivant :

Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)



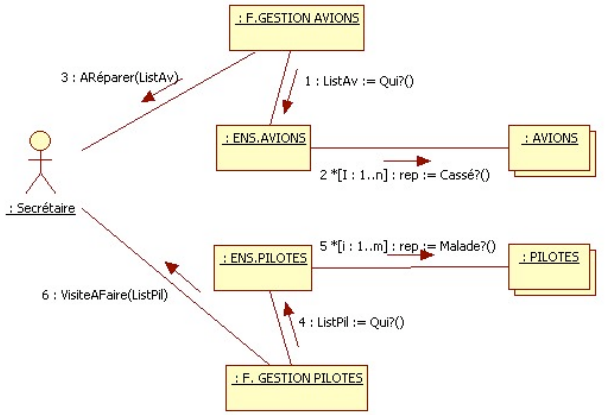
b) Le diagramme précédent a mis en évidence de nouvelles classes et des opérations (la réception d'un message dans une classe déclenche l'exécution d'une opération dans cette classe). Il met aussi en avant les liens existant entre les classes. Les messages vont, en effet, « emprunter » les associations pour passer d'une classe à une autre. Il nous faut donc modifier le diagramme de classes fourni en début d'énoncé, pour obtenir celui-ci :



c) Le second scénario parcourt tous les avions, regarde pour chacun s'il a effectué le nombre d'heures « fatidique » et, si oui, met l'avion hors service (état avion = « à contrôler »). Un travail similaire est effectué pour les pilotes (état pilote = « visite à passer »). Sont donc concernées par ce scénario, outre la classe interface et les classes ensembles, *AVIONS* et *PILOTES*. Le diagramme de collaborations est le suivant :

Projet ENRC-DVD
MIAGE

Module UML
(Outils de modélisation)
Région PAYS DE LA LOIRE



NB : nous avons pris comme hypothèse de travail le lancement, tous les matins, à l'initiative des deux classes interfaces, *F.GESTION AVIONS* et *F. GESTION PILOTES*, de ce traitement de mise en évidence. Nous avons supposé que les requêtes *Qui ? ()* étaient lancées en parallèle. Elles peuvent tout à fait ne pas l'être.

Remarque StarUML Le schéma StarUML ci-dessus ne met pas en évidence le lancement en parallèle des requêtes. Ceci se fait normalement, selon la notation UML, par //. Nous n'avons pas trouvé, dans le logiciel, la commande ou la technique permettant de le faire.

d) Là encore, de nouvelles classes apparaissent. Des opérations ont également été signalées via les messages. Des messages circulent entre les classes. Le diagramme de classes initial modifié est celui de la page suivante. Nous avons (**ceci étant hors sujet**) rajouté à ce schéma une classe *ENSEMBLES*, dont pourront hériter toutes les classes *ENS. AVIONS*, *ENS. TYPES*… On pourra en effet mettre dans cette classe les deux opérations de test d'existence (il est présent pour l'instant dans *ENS. TYPES* et *ENS. AVIONS*) et d'ajout d'un élément (présent actuellement dans *ENS. AVIONS*, *ENS. CONTROLES* et *ENS. TYPES*). Ceci ne se fera toutefois qu'au prix d'un renommage de ces opérations (*Ajouter ()* à la place de *Ntyp ()* dans *ENS. TYPES*, *Ajouter ()* à la place de *Ajout ()* dans *ENS. CONTROLES*)… et donc d'une retouche du diagramme de séquence de la première question.

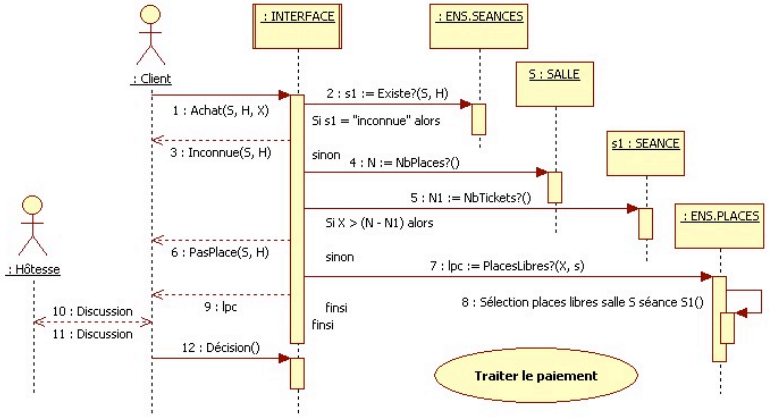
Exercice 4.10

Il faut préciser un peu le travail effectué lors de l'achat de places par un client. En l'absence de cadre précis dans l'énoncé, nous allons supposer que cela se passe comme ceci :

Le client demande X places pour la séance de H heures ayant lieu dans la salle S. Le premier travail à faire est donc de vérifier qu'il y a bien une séance (S, H). Ceci suppose que toutes les séances soient créées avant la mise en vente des billets. La vérification est effectuée par la classe *ENS.SEANCES*. Il faut ensuite vérifier qu'il reste des places libres pour cette séance. Il faut ensuite trouver des places voisines en nombre suffisant.

Nous supposerons que cette détection est effectuée par l'hôtesse de caisse. Les places proposées au client acceptées, l'hôtesse le signalera au logiciel qui va créer les tickets… mais ceci est le début du scénario de paiement, qui est hors sujet.

Cette histoire « racontée », nous pouvons produire le diagramme de séquences demandé :



Ce diagramme de séquences met en évidence plusieurs nouvelles classes, telles *INTERFACE*, *ENS.SEANCES*, *ENS.PLACES*. Elles devront être ajoutées au diagramme de classes de la question 1.

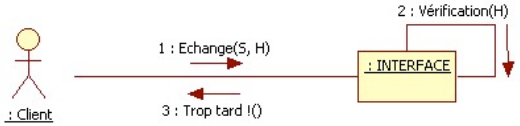
NB : bien entendu, notre réponse ne vaut que par rapport à la réponse que nous avons apportée à la question précédente. Avec un autre diagramme de classes, nous aurions assurément produit un autre diagramme de séquences.

b) NB : cette question était plus difficile qu'il n'y paraissait. Elle nécessitait, en particulier, un nombre important de schémas, du moins si l'on voulait y répondre de façon précise.

L'échange d'un (seul !) ticket ne peut se faire qu'au plus tard une heure avant la séance correspondante. Deux cas sont envisagés dans l'énoncé : un échange pour une autre séance ou un échange pour une autre place de la même séance. Ces demandes d'échange peuvent échouer. Il n'est pas dit si l'échec est dû à une impossibilité du cinéma ou à un refus du client (qui refuse la séance ou la place qui lui sont proposées).

Nous allons traiter ces différents cas en plusieurs schémas (CECI N'ÉTAIT PAS DEMANDÉ DANS L'ÉNONCÉ, CELUI-CI PARLANT MÊME EXPLICITEMENT D'UN DIAGRAMME) pour plus de clarté :

I) demande d'échange tardive (moins d'une heure avant le début de la séance) :



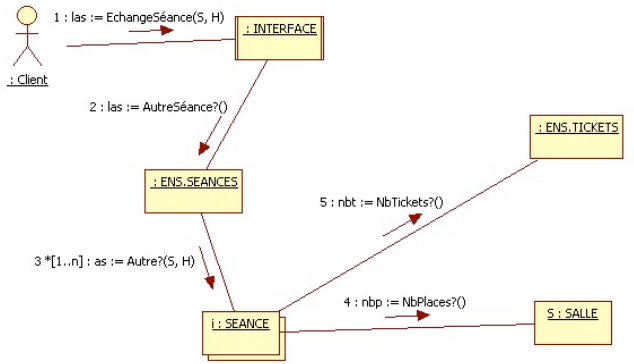
Nous supposons ici que le ticket (qui doit être fourni à l'appui de cette demande) contient toutes les informations permettant de connaître l'heure de début de la séance. À défaut, il faudra interroger la classe*TICKET*.

Projet ENRC-DVD
MIAGE

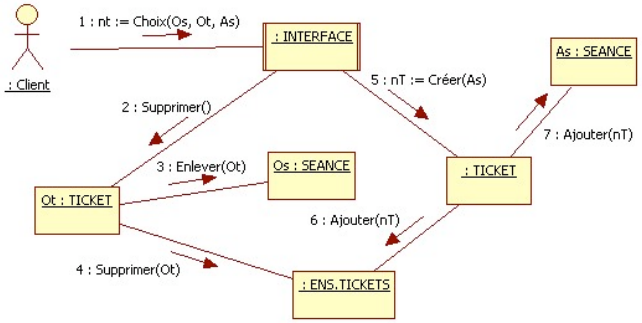
Module UML
(Outils de modélisation)

II) demande d'échange pour une autre séance, cette demande étant présentée dans les délais : il faut procéder en deux temps, le premier fournissant une ou plusieurs autres séances de remplacement, le second correspondant à l'enregistrement de l'échange (suppression de l'ancien ticket, création du nouveau). Là encore, nous ferons deux schémas.

II-1 : proposition de séances de remplacement (las)



II-2 : enregistrement de l'acceptation du client (il doit fournir l'ancien ticket, Ot)



----------fin du texte----------