Étude de cas complète de la modélisation métier à la conception détaillée en Java ou C#

Ce chapitre va nous permettre de mener une étude de cas complète partant de la modélisation métier et aboutissant à la conception détaillée (cible Java ou C#), en passant par l'expression des besoins fonctionnels et l'analyse orientée objet.

Nous allons voir en particulier :

- Quels diagrammes d'UML utiliser pour la modélisation métier ?
- Comment se servir de cette modélisation métier pour mieux définir les besoins informatiques ?
- Comment l'analyse linguistique permet d'aider à la modélisation du domaine ?
- Comment décrire une architecture en couches avec UML ?
- Comment utiliser les diagrammes de séquence et de de communication pour décrire les interactions entre objets informatiques, et répartir les opérations ?
- Comment répercuter les décisions d'affectation des responsabilités aux objets dans les diagrammes de classes ?
- Comment traduire les diagrammes UML de conception détaillée en code orienté objet ?

Étape 1 – Modélisation métier (*business modeling*)

Dans le cadre de l'amélioration qu'elle veut apporter à son système d'information, une entreprise souhaite modéliser, dans un premier temps, le processus de formation de ses employés afin que quelques-unes de leurs tâches soient informatisées.

1. Le processus de formation est initialisé lorsque le responsable formation reçoit une demande de formation de la part d'un employé. Cette demande est instruite par le responsable qui la qualifie et transmet son accord ou son désaccord à l'intéressé.
2. En cas d'accord, le responsable recherche dans le catalogue des formations agréées un stage qui correspond à la demande. Il informe l'employé du contenu de la formation et lui propose une liste des prochaines sessions. Lorsque l'employé a fait son choix, le responsable formation inscrit le participant à la session auprès de l'organisme de formation concerné.
3. En cas d'empêchement, l'employé doit informer le responsable de formation au plus tôt pour annuler l'inscription ou la demande.
4. À la fin de sa formation, l'employé doit remettre au responsable formation une appréciation sur le stage qu'il a effectué, ainsi qu'un document justifiant de sa présence.
5. Le responsable formation contrôle par la suite la facture que l'organisme de formation lui a envoyée avant de la transmettre au comptable achats.

À retenir

STÉRÉOTYPES POUR LA MODÉLISATION MÉTIER

En matière de modélisation métier, Jacobson[a] a été le premier à proposer d'utiliser les concepts UML d'acteur, cas d'utilisation, classe, package, etc., avec des stéréotypes particuliers. Dans la suite de l'exercice, nous utiliserons les stéréotypes suivants, fournis entre autres par Rational/Rose :

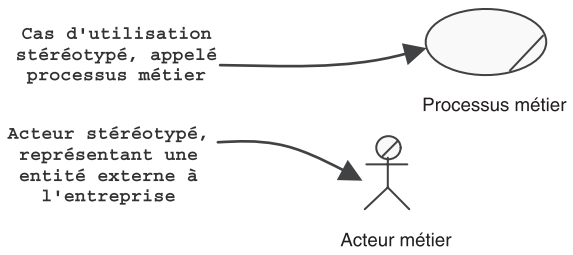
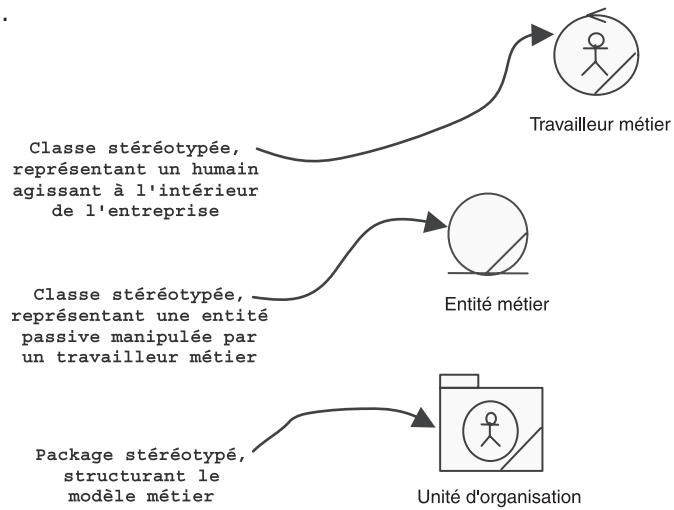


Figure 7-1. *Stéréotypes utilisés pour la modélisation métier*

a. *Software Reuse:* I. Jacobson *et al*., 1997, Prentice Hall, puis *The Unified Software Development Process*, I. Jacobson, G. Booch, J. Rumbaugh, 1999, Addison-Wesley (qui existe en version française chez Eyrolles : *Le processus unifié de développement logiciel*).

À retenir

Figure 7-1.
(suite)



EXERCICE 7-1.
Modélisation d'un processus métier

Utilisez les stéréotypes pour la modélisation métier afin de montrer le processus de formation et ses acteurs sur un diagramme de cas d'utilisation.

Modélisez le processus de formation et ses acteurs.

solution

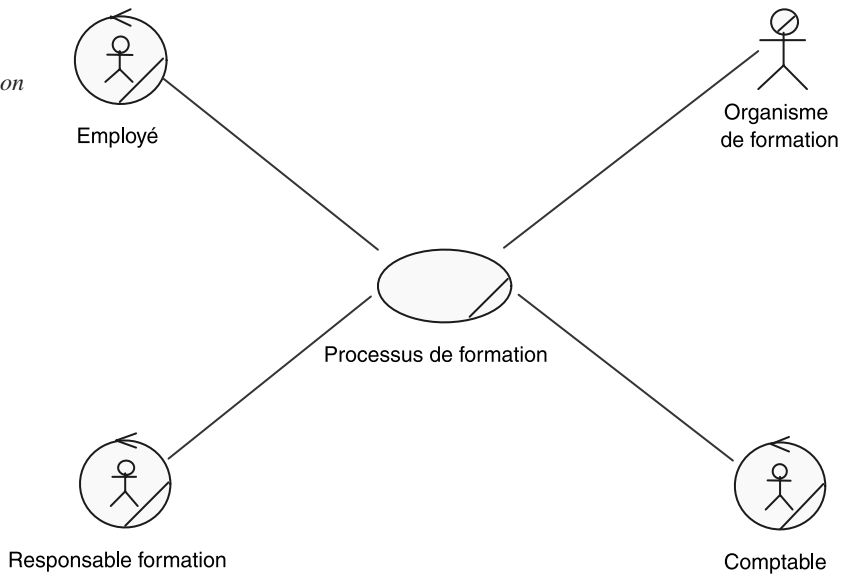
Le processus de formation est représenté par un cas d'utilisation stéréotypé.

Les acteurs impliqués sont (dans l'ordre de l'énoncé) :

- l'employé ;
- le responsable formation ;
- l'organisme de formation ;
- le comptable des achats.

Seul l'organisme de formation est une entité externe à l'entreprise, ce qui donne le schéma suivant :

Figure 7-2.
Modélisation du processus de formation avec ses acteurs



EXERCICE 7-2.
Le diagramme d'activité pour modéliser un processus

Décrivez la dynamique du processus de formation au moyen d'un diagramme d'activité. Utilisez des partitions verticales pour affecter les responsabilités aux acteurs.

Modélisez le processus de formation avec un diagramme d'activité.

solution

Le processus de formation comporte un ensemble d'actions ordonnées dans le temps et affectées à un des acteurs identifiés précédemment. Cet enchaînement se représente parfaitement grâce à un diagramme d'activité.

Les partitions[1] permettent d'agencer graphiquement les actions de telle sorte que celles qui sont affectées à un même acteur se trouvent dans la même bande verticale.

1. Généralisation UML 2 du concept de « swimlane ».

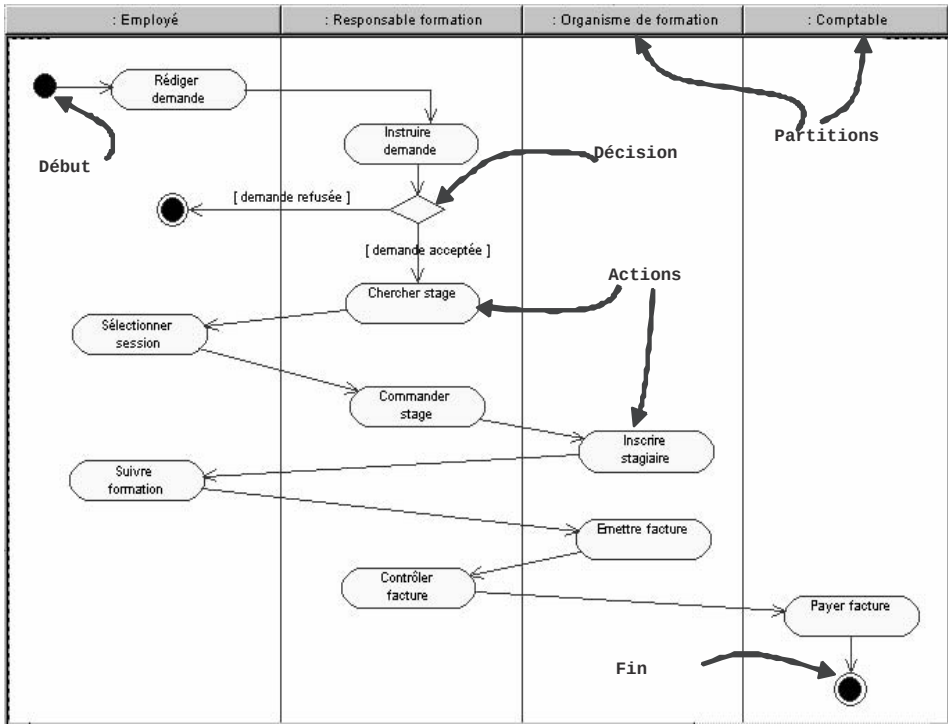


Figure 7-3.
Diagramme d'activité du processus de formation

Pour compléter le diagramme, on peut également ajouter la création et le changement d'état des entités métier, suite à la réalisation des actions.

Notez que nous n'avons pas utilisé l'icône spécifique de l'entité métier pour la *DemandeDeFormation*, afin de pouvoir plus facilement indiquer ses changements d'états entre crochets.

Le diagramme ainsi obtenu est très intéressant, puisqu'il fait le pont entre les trois axes de modélisation : fonctionnel (actions), dynamique (flots) et statique (entités et partitions) !

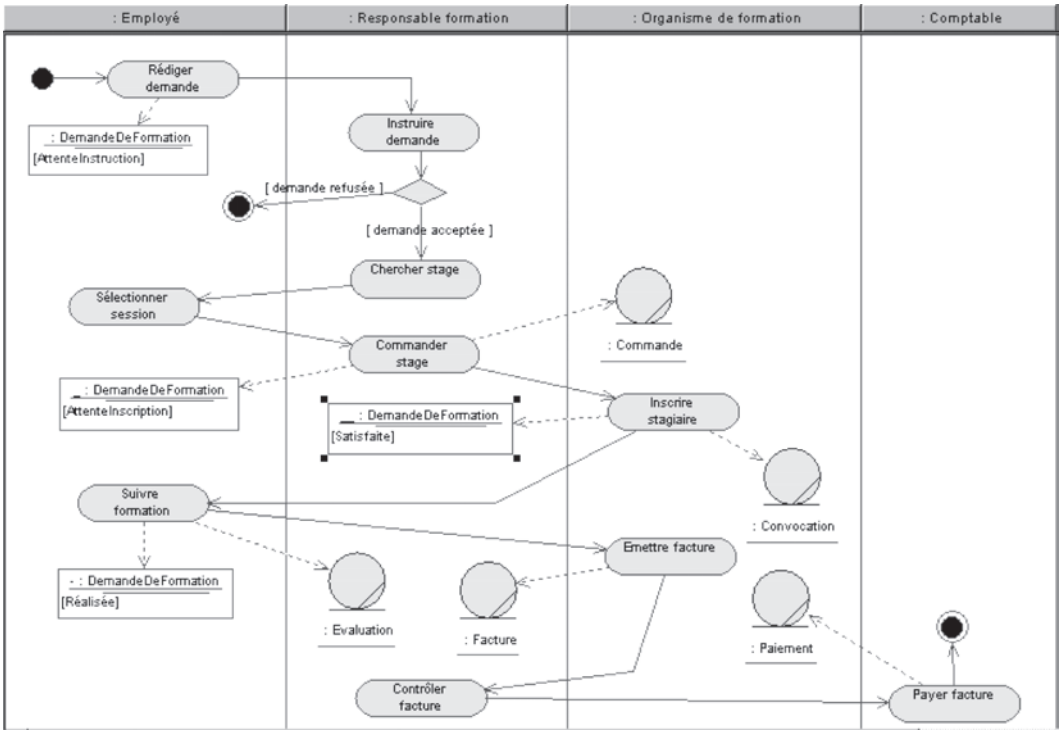


Figure 7-4.
Diagramme d'activité complété du processus de formation

Étape 2 – Définition des besoins du système informatique

Poursuivons notre étude fonctionnelle. La définition des tâches qui seront informatisées est réalisée par sélection de certaines actions du modèle métier. Nous allons ainsi déduire le cahier des charges fonctionnel du système informatique à partir de l'étude précédente, et en particulier du diagramme d'activité.

Le système doit permettre d'initialiser une demande de formation et de suivre cette demande jusqu'à l'inscription effective d'un employé.

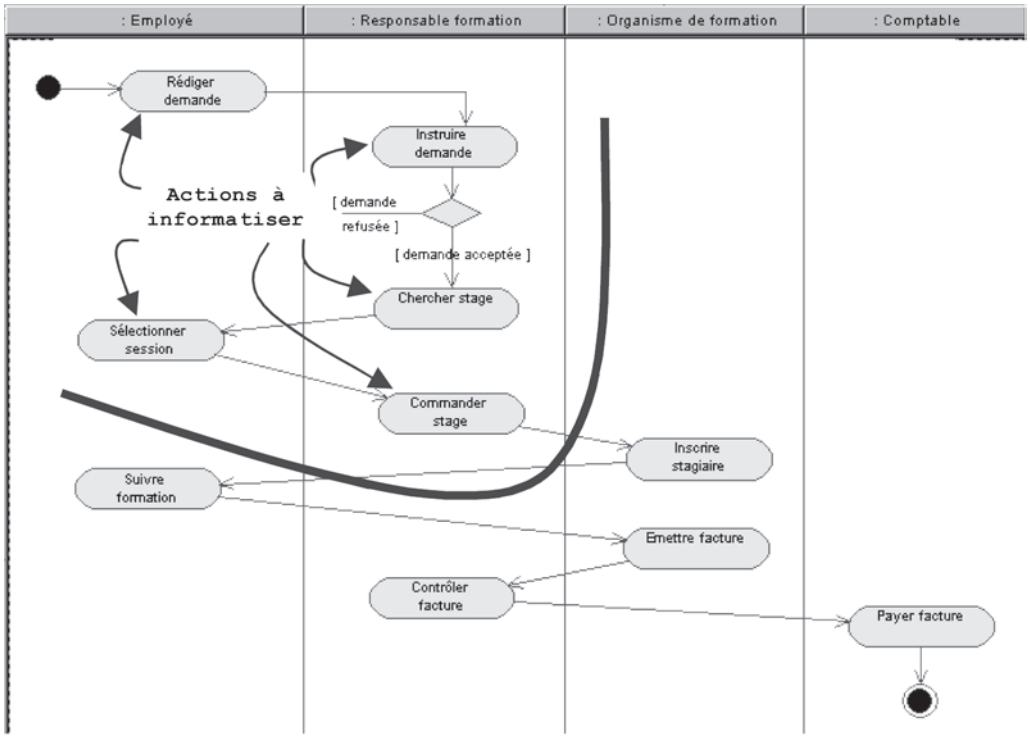


Figure 7-5.
Actions du processus de formation à informatiser

Le système de gestion des demandes de formation doit donc permettre d'automatiser les actions métier suivantes :

- Rédiger une demande (employé) ;
- Instruire une demande (responsable formation) ;
- Chercher un stage (responsable formation) ;
- Sélectionner une session (employé) ;
- Commander un stage (responsable formation).

Et il ne faut pas oublier qu'un employé a la possibilité d'annuler une demande ou une inscription à une session.

Pour tout cela, il est indispensable que le système informatique gère un catalogue de formations agréées auquel les employés peuvent accéder partiellement en lecture, et le responsable formation globalement en écriture. Ce catalogue contiendra non seulement le contenu technique, la durée, etc., des formations proposées par les organismes agréés,

mais aussi les dates et lieux des prochaines sessions. Le responsable formation pourra également créer des regroupements de formations appelés thèmes.

EXERCICE 7-3.
Le diagramme de cas d'utilisation pour définir les besoins informatiques

Élaborez le diagramme de cas d'utilisation du système informatique de gestion des demandes de formation. Écrivez quelques lignes de résumé pour chaque cas d'utilisation.

Élaborez le modèle des cas d'utilisation du système.

solution

Les acteurs impliqués sont (dans l'ordre de l'énoncé) :

- l'employé ;
- le responsable formation ;
- l'organisme de formation.

En effet, les actions du comptable ne seront pas informatisées et il n'interagira donc pas directement avec le système informatique.

D'après la liste des actions métier, on peut définir les cas d'utilisation suivants :

Demander une formation

L'employé peut consulter le catalogue et sélectionner un thème, ou une formation, ou même une session particulière. La demande est automatiquement enregistrée par le système et transmise au responsable formation par e-mail.

Traiter les demandes

Le responsable formation va utiliser le système pour indiquer aux employés sa décision (accord ou refus). En cas d'accord sur une session précise, le système va envoyer automatiquement par fax une demande d'inscription sous forme de bon de commande à l'organisme concerné. Si l'employé n'a pas choisi une session, mais simplement une formation ou un thème, le responsable formation va consulter le catalogue et sélectionner les sessions qui paraissent correspondre le mieux à la demande. Cette sélection sera transmise par e-mail à l'employé, qui pourra ainsi faire une nouvelle demande plus précise.

Gérer ses demandes

L'employé peut consulter l'état de ses demandes de formation en cours et éventuellement les annuler individuellement. Il peut également préciser une demande incomplète. Le responsable formation est automatiquement averti par e-mail.

Mettre à jour le catalogue

Le responsable formation peut introduire une nouvelle formation dans le catalogue, modifier une formation existante ou supprimer une formation qu'un organisme a abandonnée. Il peut également modifier les regroupements de formations qui ont été faits par thèmes. Il a aussi la possibilité de mettre à jour les dates et lieux des sessions.

Le diagramme préliminaire ci-après synthétise toutes ces réflexions.

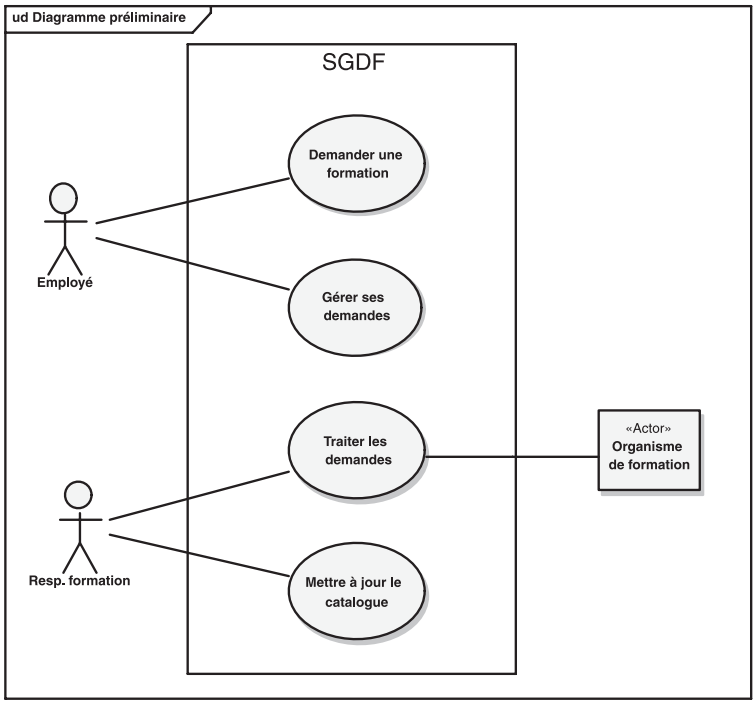


Figure 7-6.
Diagramme de cas d'utilisation préliminaire du système de gestion des demandes de formation

EXERCICE 7-4.
Description essentielle d'un cas d'utilisation

Rédigez une description détaillée essentielle de METTRE À JOUR LE CATALOGUE.

Dessinez un diagramme de séquence représentatif pour ce cas d'utilisation.

solution

Sommaire d'identification

Titre : Mettre à jour le catalogue **Type** : essentiel détaillé

Résumé : le responsable formation est chargé de la mise à jour continue d'un catalogue qui répertorie les formations agréées disponibles pour les employés. La plupart des modifications proviennent des organismes de formation.

Acteurs : Responsable formation.

Date de création : 28/09/04 **Date de mise à jour** : 26/06/06
Version : 3.0 **Responsable** : Pascal Roques

Description des enchaînements

Préconditions : le responsable formation s'est authentifié sur le système.

Postconditions : une nouvelle version du catalogue est disponible en consultation.

Scénario nominal :

1. Ce cas d'utilisation commence en général quand un Organisme de formation informe le Responsable formation de modifications par rapport à son offre.	
2. Le Responsable formation peut introduire une nouvelle formation dans le catalogue, modifier une formation existante ou enlever une formation supprimée par l'organisme. Lors d'une création ou d'une modification, le Responsable formation a la possibilité de modifier l'agenda des sessions prévues pour la formation.	3. Le Système prévient les utilisateurs connectés qu'ils risquent de travailler sur une version obsolète. Lors d'une suppression, le Système indique au Responsable formation la liste des participants qui étaient inscrits aux sessions annulées, et les inscriptions sont annulées.

4. Le Responsable formation valide ses modifications.	5. Le Système prévient les employés connectés qu'une nouvelle version du catalogue est disponible.

Séquences alternatives :

A1 : informations incomplètes

L'enchaînement A1 démarre à l'étape 2 du scénario nominal.

2. Lorsque les informations relatives à une nouvelle formation sont incomplètes (par exemple, absence de date de session), la formation est mise au catalogue mais aucune inscription ne pourra être prise. La description doit être modifiée et complétée plus tard.

Le scénario nominal continue à l'étape 2.

A2 : gestion des thèmes

La séquence A2 démarre à l'étape 2 du scénario nominal.

2. Le responsable formation peut commencer par créer un nouveau thème ou renommer un thème existant. Il peut également déplacer une formation d'un thème à un autre.

Le scénario nominal continue à l'étape 2.

Spécifications supplémentaires

Concurrence : ce cas d'utilisation ne peut être exécuté que par un responsable à la fois.

Disponibilité : le catalogue est accessible *via* l'Intranet du lundi 9 h au vendredi 17 h. Les actions de maintenance doivent être limitées au strict minimum pendant ces heures.

Exemple de diagramme de séquence système du cas d'utilisation : « Mettre à jour le catalogue ».

Nous avons utilisé un fragment de type alternatives (`alt`) pour indiquer que les actions effectuées par le Responsable formation peuvent arriver dans n'importe quel ordre. Ce fragment alternatives est lui-même imbriqué dans une boucle (`loop`).

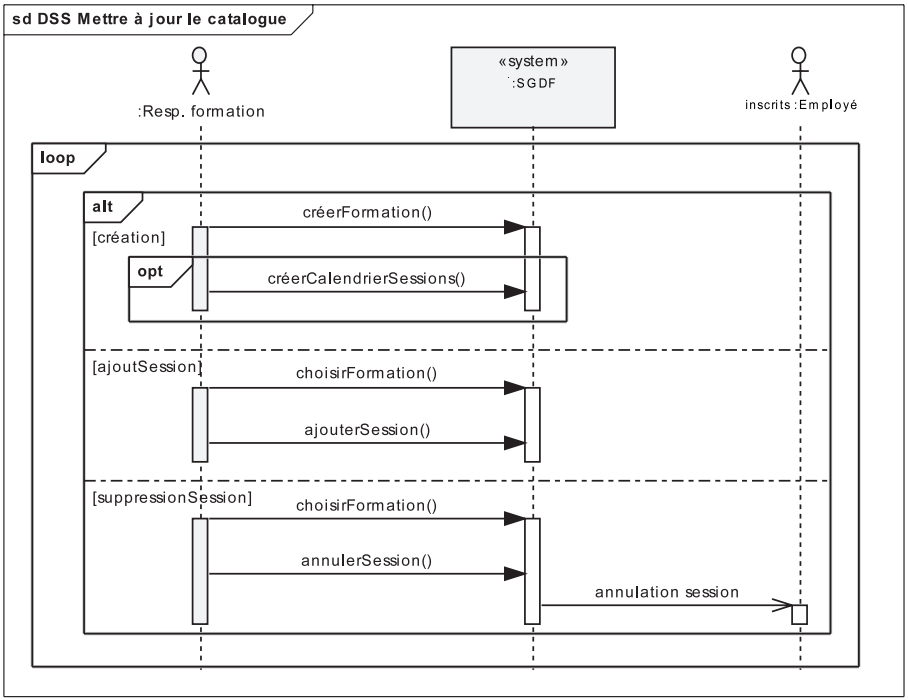


Figure 7-7. *Diagramme de séquence système du cas d'utilisation :* Mettre à jour le catalogue

Remarquons que les employés inscrits à une session annulée sont automatiquement avertis par le système (e-mail). La flèche du message « annulation session » est ouverte pour indiquer qu'il s'agit d'un message asynchrone[2].

À retenir

FLOTS DE CONTRÔLE DES MESSAGES

Un flot de contrôle synchrone signifie que l'objet émetteur se bloque en attendant la réponse du récepteur du message.

Dans un flot de contrôle asynchrone au contraire, l'objet émetteur n'attend pas la réponse du récepteur et poursuit sa tâche sans se soucier de la réception de son message.

Il faudra donc ajouter un acteur secondaire à notre diagramme de cas d'utilisation préliminaire.

2. Attention, ceci est conforme à la nouvelle norme UML 2. Dans les versions précédentes, le message asynchrone était représenté par une demi-flèche ouverte.

EXERCICE 7-5.
Amélioration du diagramme de cas d'utilisation

Améliorez le diagramme de cas d'utilisation préliminaire du système informatique de gestion des demandes de formation.

En particulier, pour demander une formation et pour maintenir le catalogue, le système doit proposer une fonctionnalité de base de consultation du catalogue.

Améliorez le modèle des cas d'utilisation du système.

solution

Pour demander une formation, le système doit proposer une fonctionnalité de base de consultation du catalogue. L'employé peut consulter le catalogue sans faire de demande..La création d'une demande vient étendre optionnellement la consultation[3].

L'acteur « Organisme de formation » ne fait que recevoir des messages venant du système, nous utiliserons donc une association unidirectionnelle. De même, l'employé est acteur secondaire (récepteur) des cas d'utilisation du responsable formation et réciproquement.

De plus, un responsable formation peut également effectuer une demande de formation, etc. Nous ajoutons donc une relation de généralisation entre acteurs.

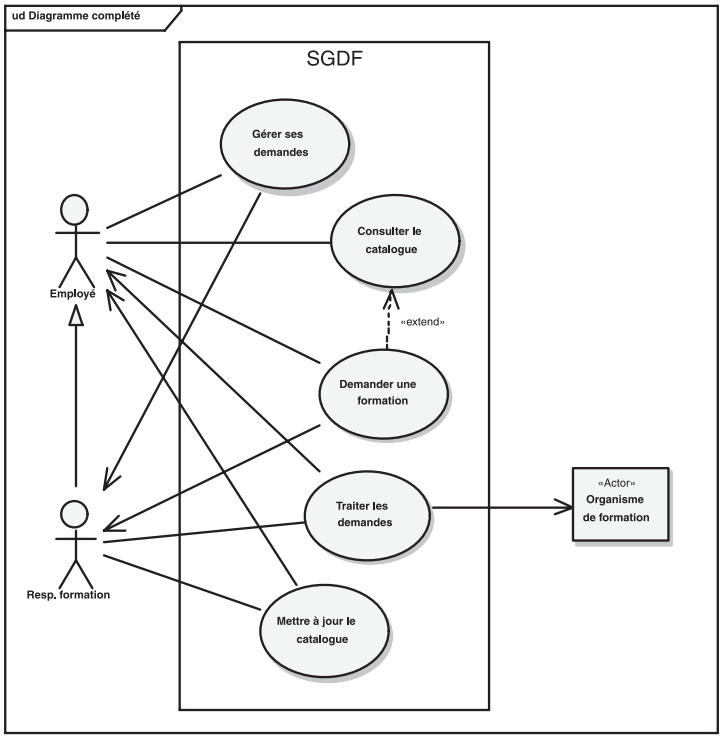
Enfin, pour ne pas surcharger le diagramme, nous n'y représenterons pas le processus d'identification de l'employé ou du responsable formation. Mais nous créerons plutôt un package différent. Le modèle de cas d'utilisation est ainsi divisé en deux packages :

- cas d'utilisation opérationnels ;
- cas d'utilisation de support.

Le diagramme de cas d'utilisation du premier package devient :

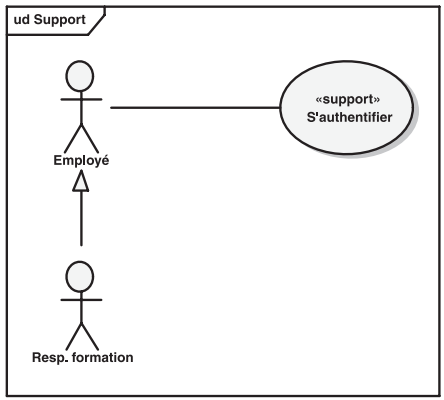
3. On pourrait également argumenter que pour demander une formation, il faut obligatoirement consulter le catalogue, ce qui pourrait justifier une relation d'inclusion (« include ») entre cas d'utilisation. Nous voyons là une fois de plus la difficulté (et le danger) des relations entre cas d'utilisation !

Figure 7-8.
Diagramme de cas d'utilisation du package des cas opérationnels



Celui du second package est fourni par la figure suivante.

Figure 7-9.
Diagramme de cas d'utilisation du package des cas de support



EXERCICE 7-6.
Diagramme de contexte statique

La contrainte de concurrence sur ce cas d'utilisation amène une dernière question.

Élaborez le diagramme de contexte statique du système.

solution

Le système de gestion des demandes de formation est fondamentalement multi-utilisateurs (typiquement : un intranet), sauf pour le responsable formation qui doit en être le seul utilisateur en modification à un moment donné.

Les organismes de formation n'ont pas accès au système : ils ne font que recevoir des commandes (une à la fois), ce qui explique la flèche de navigabilité sur l'association entre le système et l'acteur non-humain.

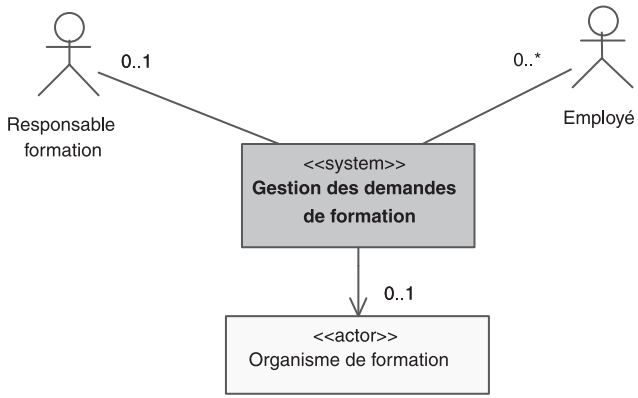


Figure 7-10.
Diagramme de contexte statique du système de gestion des demandes de formation

Étape 3 – Analyse du domaine (partie statique)

Nous allons reprendre l'énoncé de l'étude de cas, déjà traitée du point du vue fonctionnel aux étapes 1 et 2, en le reformulant et en le simplifiant légèrement.

1. Le processus de formation est initialisé lorsque le responsable formation reçoit une demande de formation de la part d'un employé.
2. Cette demande est instruite par le responsable qui qualifie la demande et transmet son accord ou son désaccord à l'intéressé.
3. En cas d'accord, le responsable recherche dans le catalogue des formations agréées un stage correspondant à la demande.
4. Il informe l'employé du contenu de la formation et lui propose une liste des prochaines sessions.
5. Lorsque l'employé retourne son choix, le responsable formation inscrit le participant à la session auprès de l'organisme de formation concerné.
6. Le responsable formation contrôle par la suite la facture que lui a adressée l'organisme de formation avant de la transmettre au comptable des achats.

Nous avons déjà identifié les travailleurs métier impliqués dans le processus de formation (exercice 7-1). Il nous faut maintenant aborder ce dernier sous l'angle statique qu'il présente et découvrir les principales entités métier.

Pour cela, une analyse lexicale du texte de l'énoncé est tout à fait indiquée. Cette technique est en général sous-utilisée, car elle peut sembler fastidieuse. Elle est pourtant très efficace pour découvrir des objets candidats dans les cas difficiles, par exemple si le modélisateur connaît mal le domaine métier.

EXERCICE 7-7.
Analyse linguistique (1/6)

Modélisez la phrase 1, en employant les stéréotypes de Jacobson.

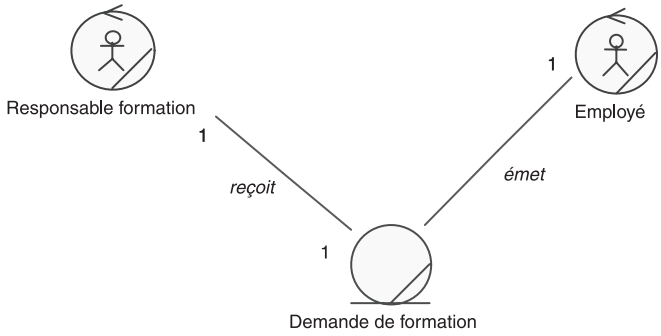
solution

Une analyse simpliste des noms et groupes nominaux fournit les entités suivantes : processus de formation, responsable formation, demande de formation, employé. Considérons chacun des candidats à tour de rôle.

- *Processus de formation* a déjà été identifié à l'étape 1 en tant que processus métier : il n'apparaîtra pas sur le diagramme de classes.
- En revanche, *responsable formation* et *employé* y figureront, car ils ont été identifiés comme travailleurs métier.
- Articles « un/une » ou « le/la ». L'article indéfini (« un/une ») indique que le nom est utilisé de façon générique, alors que l'article défini (« le/la ») indique que le nom est unique dans le contexte de la phrase. Attention cependant : l'article « un » signifie souvent « un en général », (comme dans : lorsque le responsable formation reçoit *une* demande de formation), mais aussi parfois « un et un seul » pour indiquer que le pluriel ne serait pas possible (comme dans : de la part d'*un* employé). Dans ce cas, on obtient une multiplicité 1 sur une association.

Nous en déduisons aisément le diagramme de classes suivant.

Figure 7-11.
Modélisation statique de la phrase 1



EXERCICE 7-8.
Analyse linguistique (2/6)

Modélisez la phrase 2.

solution

En procédant, comme pour la première phrase, à une analyse simpliste des noms et groupes nominaux, on obtient les entités suivantes : demande, responsable, accord, désaccord, l'intéressé.

- Référence indirecte par « ce/cette », « ces » : une phrase utilisant le mot « ce » fait presque toujours référence au sujet de la phrase précédente. Les concepts *demande* et *demande de formation* sont donc identiques.
- Attention aux synonymes ! Il est clair que *responsable* n'est pas un nouveau concept, mais simplement une forme plus courte de *responsable formation*. C'est un peu moins évident avec le mot *intéressé* qui fait référence à l'employé qui a émis la demande.
- Possessifs : « son/sa », « ses ». Nous pouvons traduire la possession de deux façons : une association ou un attribut. Nous choisissons l'association si le possesseur et la possession sont tous les deux des concepts. Nous choisissons l'attribut si la possession est une simple caractéristique du possesseur.
- Conjonction de coordination « ou ». Un « ou exclusif » doit faire penser à une relation de généralisation/spécialisation, mais uniquement si les concepts spécialisés ont des attributs et des comportements différents. Dans le cas contraire, il vaut mieux introduire un simple type énuméré. Dans notre exemple, nous pouvons considérer que l'accord ou le désaccord sont des spécialisations d'une entité *réponse* relative à la demande. En effet, le désaccord aura probablement un attribut *motif*, contrairement à l'accord.
- Verbes : la demande est reçue par le responsable, puis instruite et enfin qualifiée. Il n'est pas question de dessiner trois associations pour modéliser toutes les actions que le responsable peut effectuer à propos de la demande. Au contraire, le diagramme de classes doit représenter une vue statique qui soit valable à tout moment. Nous renommons donc l'association entre *responsable* et *demande* avec un verbe plus neutre (*traiter*), et nous modifions les multiplicités en conséquence.

Pour compléter le diagramme, nous avons supposé qu'un employé ne peut pas émettre plus d'une demande à la fois. On notera également les multiplicités entre *demande* et *réponse* : une réponse est forcément liée à une et une seule demande ; une demande peut exister sans réponse (tant qu'elle n'est pas instruite).

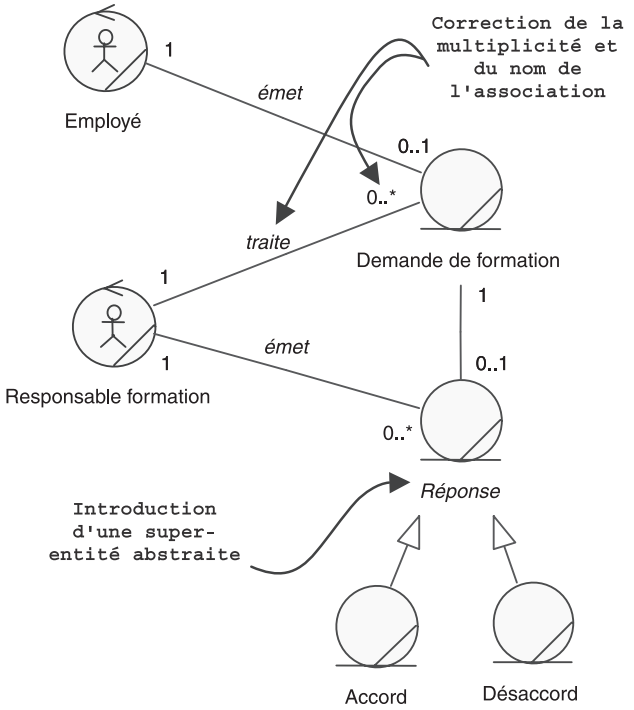


Figure 7-12.
Modélisation statique de la phrase 2

EXERCICE 7-9.
Analyse linguistique (3/6)

Modélisez la phrase 3.

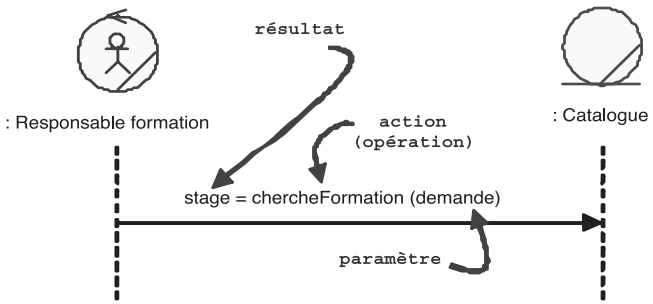
solution

Une nouvelle analyse rapide des noms et groupes nominaux fournit les entités suivantes : accord, responsable, catalogue, formation, stage, demande.

- *Accord*, *responsable* et *demande* ont été identifiés précédemment.
- Conteneur et contenu : *catalogue* est un conteneur composé de *formations* ; les deux peuvent donner lieu à des entités, si elles portent des attributs et des comportements. C'est bien le cas dans notre exemple. On doit alors étudier la possibilité d'une agrégation ou d'une composition. Sinon, le contenu peut être un simple attribut du conteneur.
- Pluriel : le pluriel sur un nom (catalogue *des* formation*s*) donne souvent lieu à une entité au singulier, mais avec une multiplicité « 0..* » sur une association.
- Verbes : attention, les verbes correspondent souvent à des actions effectuées sur les entités (le responsable *recherche*…). Ces actions ne se traduisent

généralement pas dans le diagramme de classes d'analyse. Elles donnent en revanche des indications sur la dynamique, et peuvent donner lieu à des fragments de diagramme de séquence ou de communication.

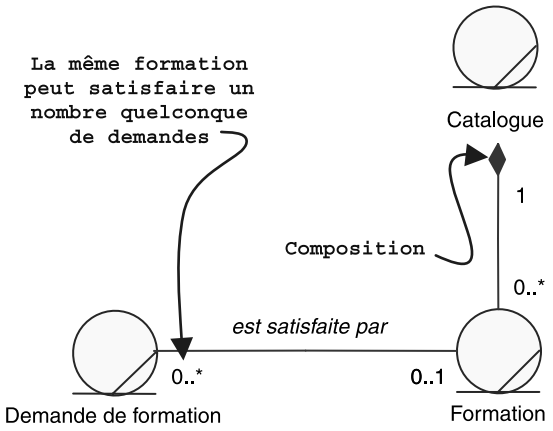
Figure 7-13.
Fragment de modèle dynamique issu de la phrase 3



- Adjectifs : ils représentent soit des attributs d'une entité déjà identifiée, soit une possibilité de relation de généralisation. Attention : ils peuvent aussi simplement ajouter du « bruit » dans le texte, comme dans notre cas où seules les *formations agréées* ont une existence notable dans le processus de formation.
- Participes présents : ils indiquent souvent une association entre deux entités. Par exemple, « un stage *correspondant à* la demande » amène la création d'une association entre les entités *stage* et *demande*.
- Attention aux synonymes ! Pour éviter les répétitions qui alourdissent le style, l'usage des synonymes est fréquent : *formation* et *stage* en sont une bonne illustration. Le modélisateur doit débusquer ces synonymes et les « réduire » en choisissant un nom principal d'entité. Nous préférerons le terme *formation* plutôt que celui de *stage*.

Toute cette discussion conduit au diagramme de classes ci-après.

Figure 7-14.
Modélisation statique de la phrase 3



EXERCICE 7-10.
Analyse linguistique (4/6)

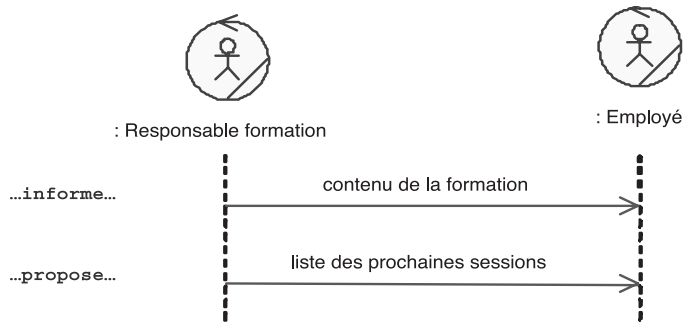
Modélisez la phrase 4.

solution

Une analyse sommaire des noms et groupes nominaux permet de relever les entités suivantes : employé, contenu, formation, liste, session.

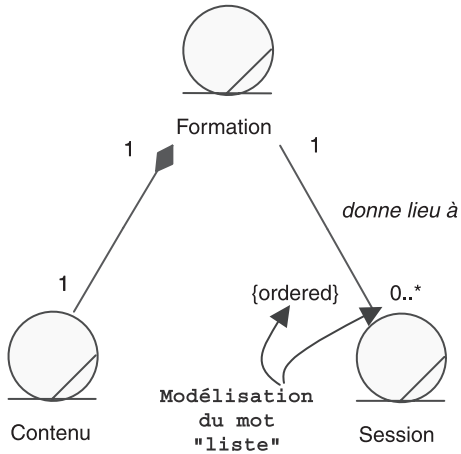
- Référence indirecte par un pronom : « il/elle », etc. Les pronoms sont des références à un autre nom qui est souvent le sujet de la phrase précédente. Ici, « *il* informe… » concerne de toute évidence le responsable.
- *Employé* et *formation* ont été identifiés précédemment.
- Contenance ou possession : entité à part entière ou attribut suivant les cas. Si l'on considère qu'une formation a un contenu dont la structure est complexe (prérequis, objectifs, plan détaillé, etc.) et un comportement, il est tout à fait justifié d'en faire une entité. Comme nous l'avons souligné précédemment, on doit étudier la possibilité d'une agrégation ou d'une composition.
- Conteneur : le mot *liste* indique simplement une multiplicité « * » et apporte souvent une notion d'ordonnancement (contrainte UML {`ordered`}). Il ne faut surtout pas identifier une entité *liste* lors de la phase d'analyse : le choix des types de conteneur est vraiment du ressort de la conception détaillée, voire de l'implémentation.
- Attention aux faux synonymes ! Cette fois-ci, il ne faut pas croire que *session* est synonyme de *formation* ou *stage*. En effet, le concept de *session* ajoute des notions de date et de lieu qui ne font pas partie du concept plus générique de *formation*. On peut évoquer les mérites de la « formation UML de 4 j proposée par Valtech Training », et s'inscrire à la « session qui a lieu à Toulouse du 9 au 19 mai 2005 ». De plus, ces entités ont des comportements bien distincts : on peut reporter ou annuler une *session*, sans modifier de quelque manière que ce soit la *formation*.
- Verbes : là encore, les verbes représentent des échanges de messages entre instances, et absolument pas des associations.

Figure 7-15.
Fragment de modèle dynamique issu de la phrase 4



Le résultat de ces cogitations est synthétisé sur le schéma présenté ci-après.

Figure 7-16.
Modélisation statique de la phrase 4



La relation entre *formation* et *session* est une nouvelle illustration de l'important « pattern de la métaclasse », étudié au chapitre 3, exercice 3-11.

EXERCICE 7-11.
Analyse linguistique (5/6)

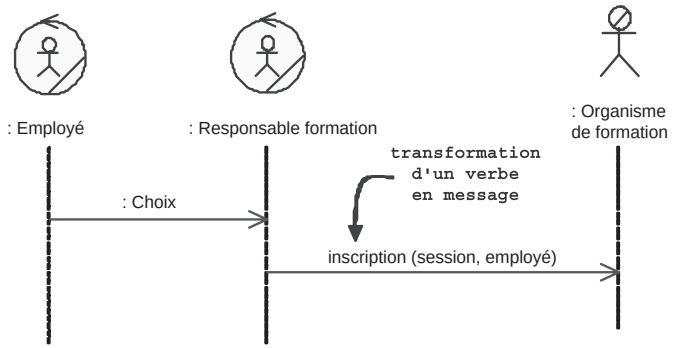
Modélisez la phrase 5.

solution

Une fois encore, l'analyse linguistique nous fournit les entités candidates : employé, choix, responsable formation, participant, organisme de formation.

- Employé, responsable et organisme de formation ont été identifiés précédemment.
- Une nouvelle fois, il faut veiller à ne pas modéliser un comportement dynamique dans le diagramme de classes ! La phrase 5 se traduirait directement par le fragment de diagramme de séquence suivant.

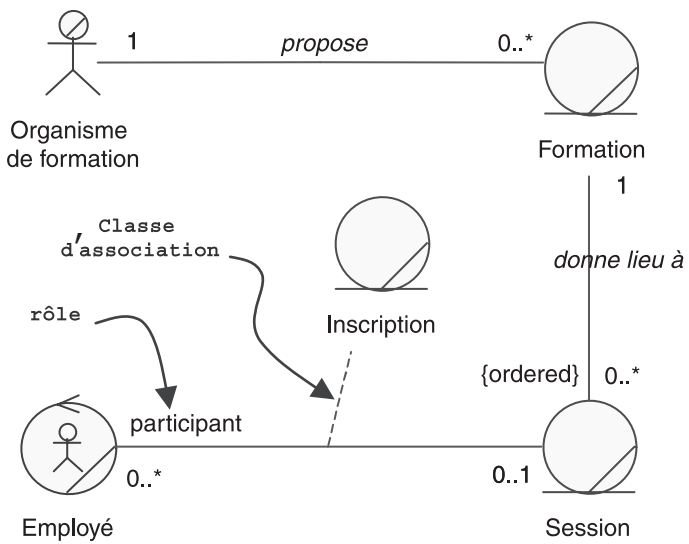
Figure 7-17.
Modélisation dynamique de la phrase 5



- Verbes : souvent le verbe cache un nom ! Dans l'exemple précédent, où « le responsable formation *inscrit* le participant », le diagramme de séquence fait apparaître un message *inscription* qui porte des paramètres. En fait, nous avons besoin d'une entité *inscription* qui représente une sorte de contrat entre le responsable et l'organisme externe. Cette entité porte des attributs (date, prix, etc.) et des comportements (reporter, annuler, etc.). Nota – Les entités de type *contrat* se modélisent très fréquemment comme des classes d'association.
- Termes vagues : le mot *choix* est délicat à modéliser. En effet, il s'agit d'un mot imprécis, d'un terme vague. Il faut donc le situer dans le contexte auquel il se rapporte. D'après la phrase 4, l'employé choisit une des sessions proposées par le responsable. Le mot *choix* dans ce contexte ne sert qu'à identifier une *session* particulière pour laquelle le responsable va faire une demande d'inscription auprès de l'organisme de formation. Il ne s'agit donc pas d'une nouvelle entité, mais plutôt d'un rôle joué par une session dans une relation avec une inscription.
- Rôles : il faut veiller à ne pas créer systématiquement de nouvelles entités. En effet, certains noms représentent simplement des rôles joués par des entités déjà identifiées. C'est le cas pour *participant*, qui ne décrit qu'un rôle joué par un employé dans le cadre d'une session.
- Acteurs. Faut-il relier *organisme de formation* à *session* ? C'est ce que semble indiquer la phrase 5. Toutefois, nous avons vu avec la phrase 4 que les sessions se rapportent toutes à une formation. Il est donc plus judicieux de relier directement *organisme de formation* à *formation*.

La modélisation statique de la phrase 5 est illustrée sur la figure suivante.

Figure 7-18.
Modélisation statique de la phrase 5



EXERCICE 7-12.
Analyse linguistique (6/6)

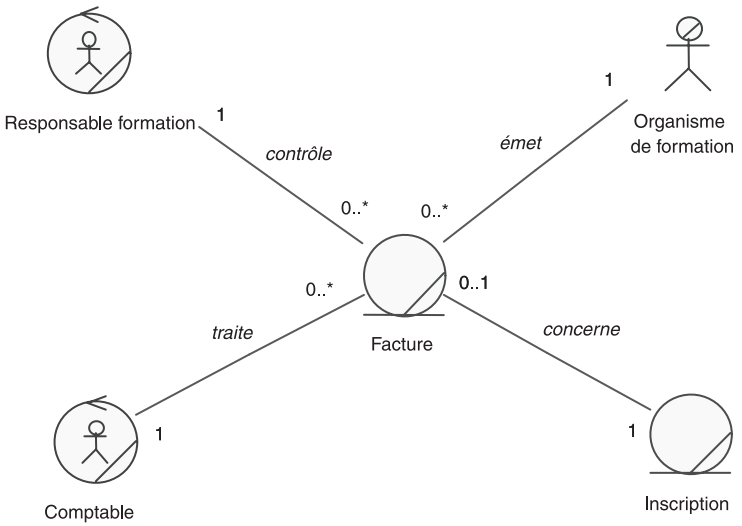
Modélisez la phrase 6.

solution

Pour cette dernière phrase aussi, l'analyse linguistique nous fournit les entités candidates : responsable formation, suite, facture, organisme de formation, comptable des achats.

- *Responsable formation* et *organisme de formation* ont été identifiés précédemment. *Comptable des achats* est un travailleur métier comme nous l'avions indiqué à l'étape 1.
- Propositions temporelles : elles ne servent que pour la modélisation dynamique. Dans notre cas, « contrôle *par la suite*… » ne fait qu'indiquer une succession temporelle de messages. Elle permet implicitement de relier la *facture* à *l'inscription* (voir phrase 5).

Figure 7-19.
Modélisation statique de la phrase 6



EXERCICE 7-13.
Modèle métier – découpage en packages

Rassemblez tous les fragments précédents sur un même diagramme de classes.

Proposez une découpe du modèle en packages représentant des unités d'organisation métier.

Élaborez un diagramme de classes global montrant un découpage en packages.

solution

Le modèle statique préliminaire de notre étude de cas provient de la réunion de tous les diagrammes précédents.

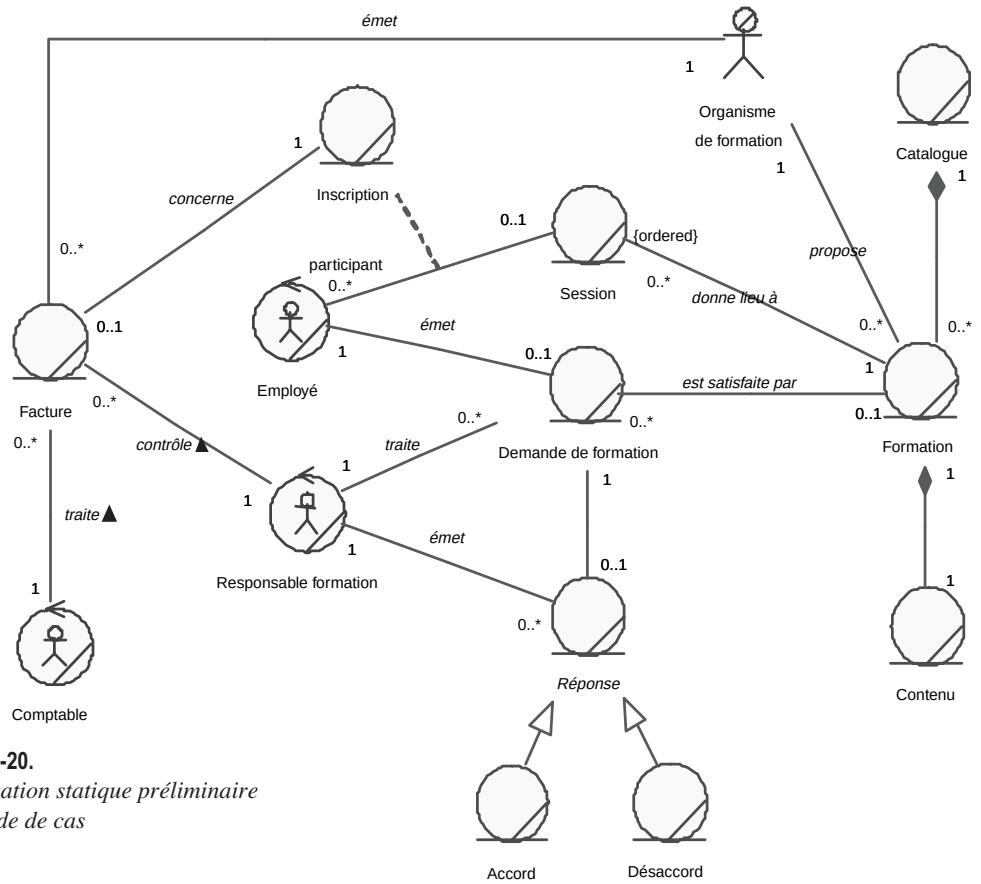


Figure 7-20.
Modélisation statique préliminaire de l'étude de cas

Comment procéder pour découper ce modèle en unités d'organisation métier ?

- Il est clair que toute la partie droite du modèle (y compris l'entité *session*) concerne le catalogue de formation et constitue une unité cohérente, relativement stable dans le temps.
- Le couple *facture-comptable* est aussi relativement indépendant du reste, et correspond d'ailleurs à un service bien identifié de l'entreprise.
- Tout le reste est de la responsabilité du responsable formation et constitue un ensemble cohérent, centré sur la demande de formation.

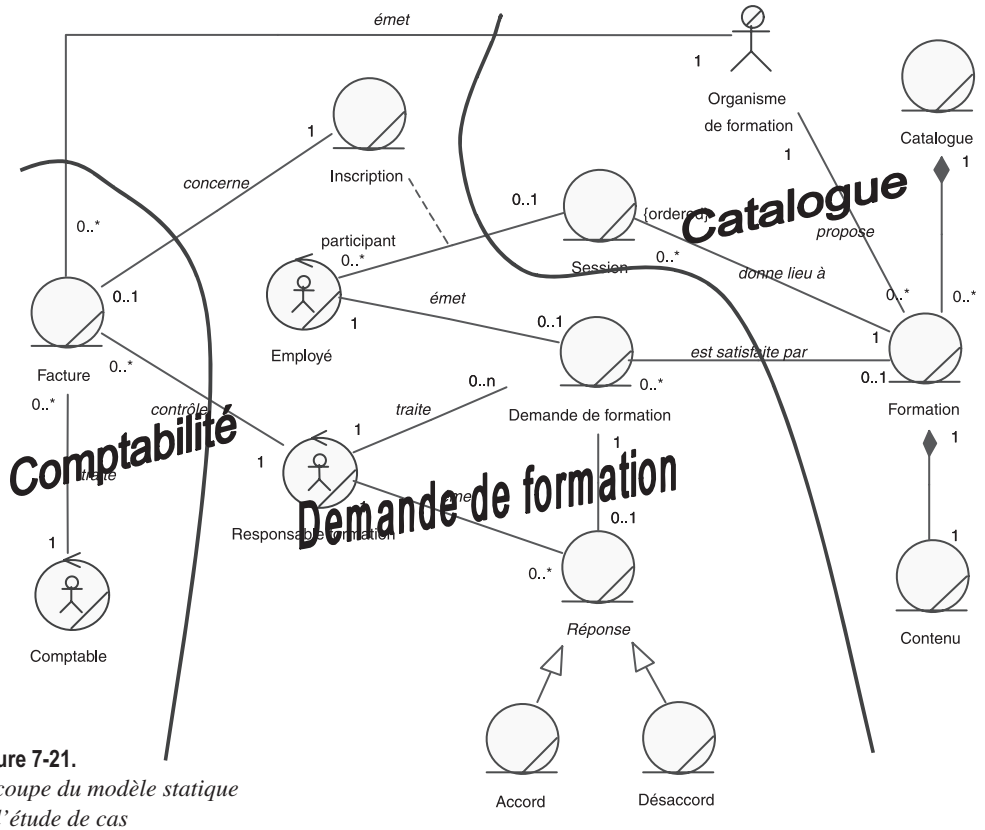
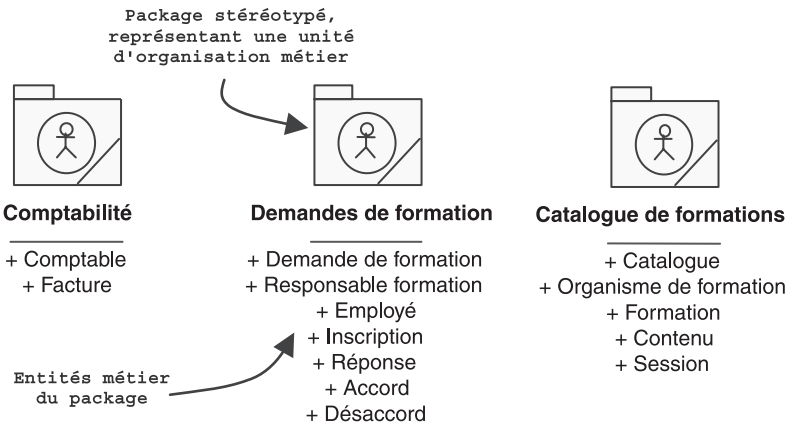


Figure 7-21.
Découpe du modèle statique de l'étude de cas

Nous pouvons représenter cette structuration en découpant le schéma précédent grâce à des packages stéréotypés, comme cela a été indiqué à l'étape 1.

Figure 7-22.
Packages stéréotypés représentant la découpe du modèle métier



EXERCICE 7-14.
Modèle métier – diagramme de classes par package

Dessinez un diagramme de classes par unité d'organisation, en essayant de minimiser les dépendances entre packages. Ajoutez quelques attributs métier pertinents pour compléter le modèle métier statique.

Élaborez un diagramme de classes complet par package.

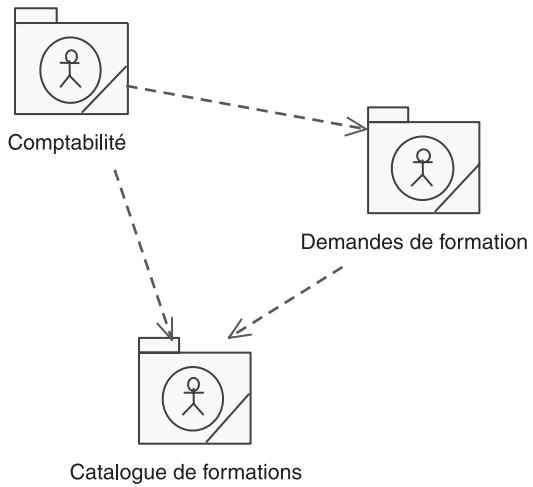
Commencez par un diagramme de packages avec des dépendances.

solution

Il est clair que le package *Catalogue de formation* peut être autonome et qu'il peut donc constituer un composant métier réutilisable. Il est également logique de faire dépendre la facture de la demande de formation plutôt que le contraire. Le schéma de dépendances entre unités d'organisation métier que l'on obtient est donc celui qui est présenté ci-après. Il s'agit d'un diagramme de packages qui respecte les sacro-saints principes des dépendances entre packages :

- pas de dépendances mutuelles ;
- pas de dépendances circulaires.

Figure 7-23.
Diagramme de packages montrant les dépendances souhaitées entre packages



Cet objectif de dépendances entre packages impose une contrainte sur la navigabilité des associations qui traversent deux unités d'organisation, de la façon suivante.

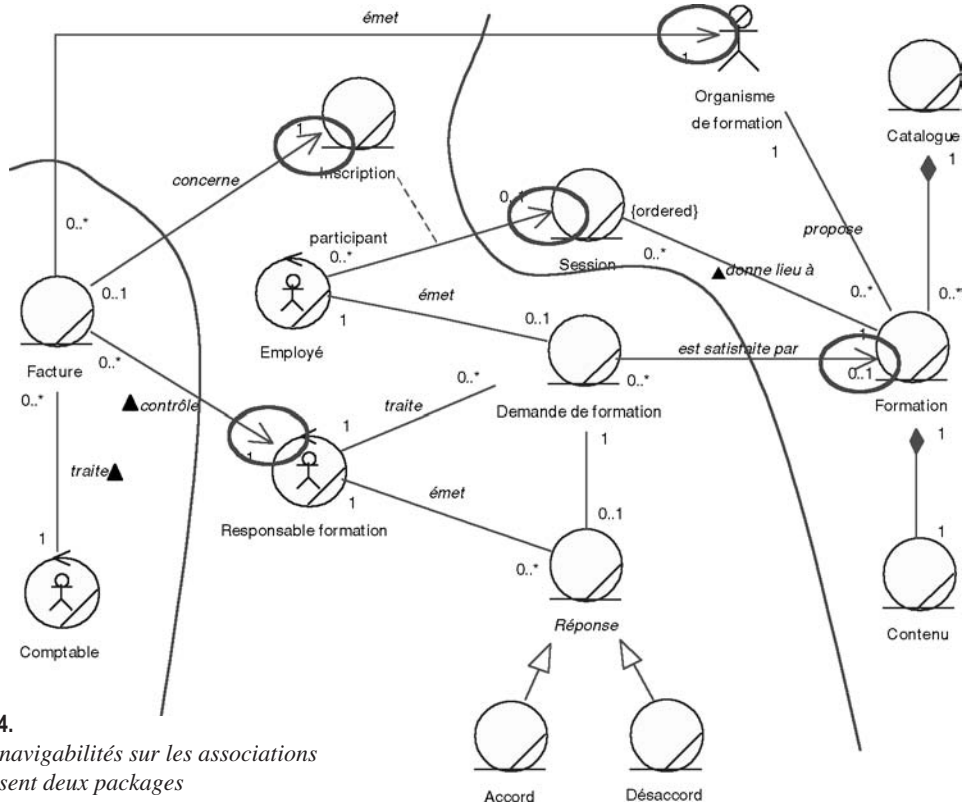


Figure 7-24.
Ajout des navigabilités sur les associations qui traversent deux packages

Nous pouvons maintenant dessiner un diagramme de classes par package en ajoutant quelques attributs métier pertinents. Notez la présence sur les diagrammes des classes reliées appartenant aux autres packages (avec l'indication « (from xxx)[4] »).

4. Cette convention graphique efficace n'est pas standard UML. Elle a été popularisée par l'outil Rational/Rose et reprise par Enterprise Architect.

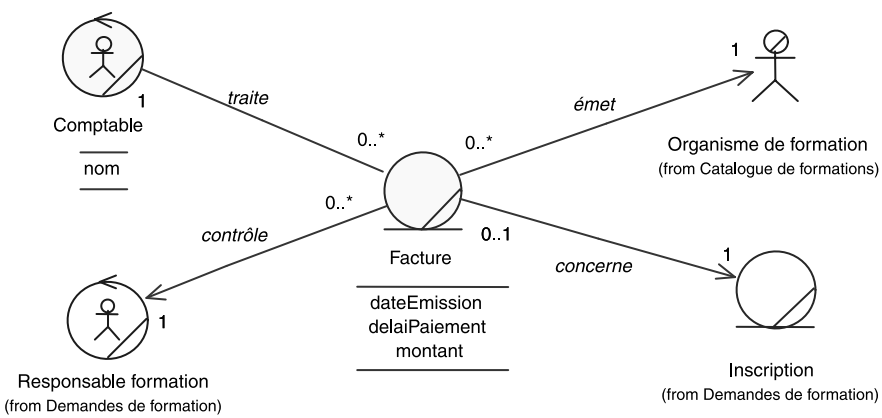


Figure 7-25. *Diagramme de classes du package* Comptabilité

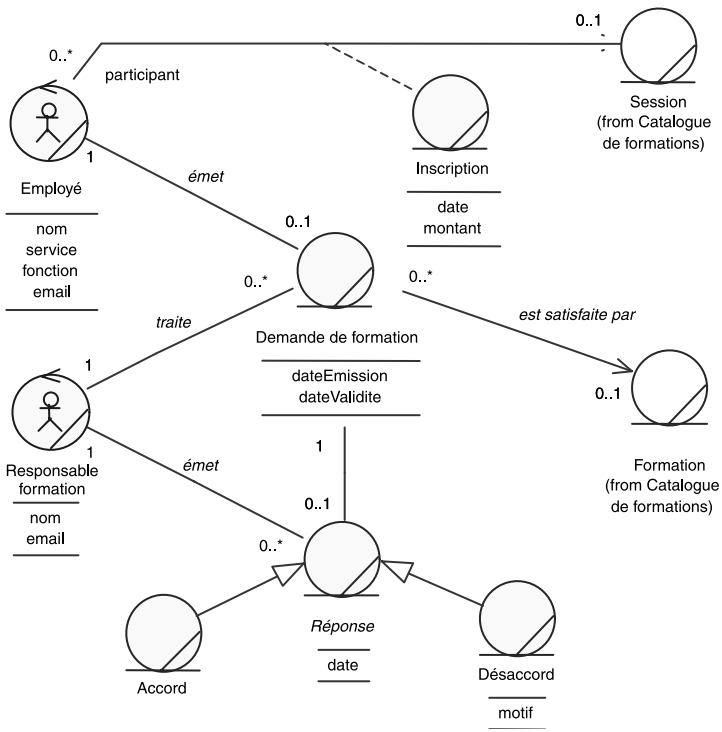
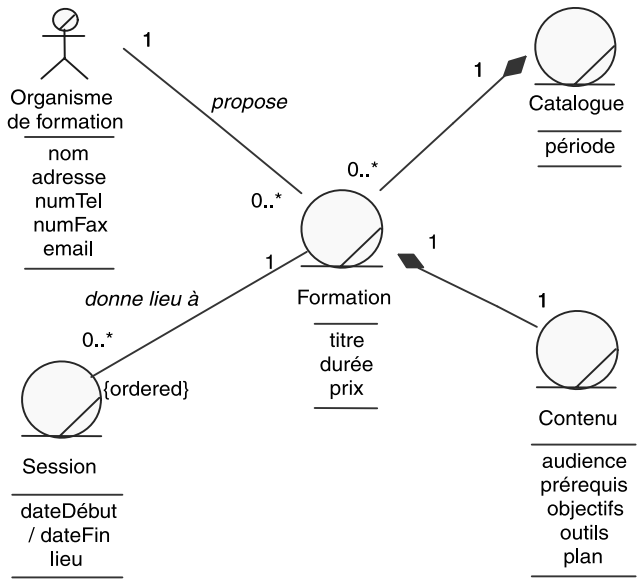


Figure 7-26. *Diagramme de classes du package* Demandes de formation

Figure 7-27.
Diagramme de classes du package Catalogue de formations



On notera l'attribut dérivé */dateFin* de *Session*. La contrainte pourrait s'écrire simplement en OCL[5] : {self.dateFin = self.dateDébut + formation.durée }.

Étape 4 – Analyse du domaine (partie dynamique)

EXERCICE 7-15.
Modèle métier – diagramme d'états

Réalisez le diagramme d'états de la demande de formation.

solution

Quelles informations avons-nous déjà réunies sur la dynamique d'une demande de formation ?

Reprenons les trois premières phrases de l'énoncé :

1. Le processus de formation est initialisé lorsque le responsable formation reçoit une demande de formation de la part d'un employé. Cette demande

5. OCL (Object Constraint Language) est un petit langage à expressions, permettant d'exprimer des contraintes sur les diagrammes de classes UML au moyen d'expressions booléennes qui doivent être vérifiées par le modèle. OCL fait partie intégrante d'UML depuis la version UML 1.1.

est instruite par le responsable qui qualifie la demande et transmet son accord ou son désaccord à l'intéressé.

2. En cas d'accord, le responsable recherche dans le catalogue des formations agréées un stage correspondant à la demande. Il informe l'employé du contenu de la formation et lui propose une liste des prochaines sessions. Lorsque l'employé retourne son choix, le responsable formation inscrit le participant à la session auprès de l'organisme de formation concerné.
3. En cas d'empêchement, l'employé doit informer le responsable de formation au plus tôt pour annuler l'inscription ou la demande.

Nous avions également réalisé un diagramme d'activité du processus de formation montrant les objets métier et leurs changements d'états.

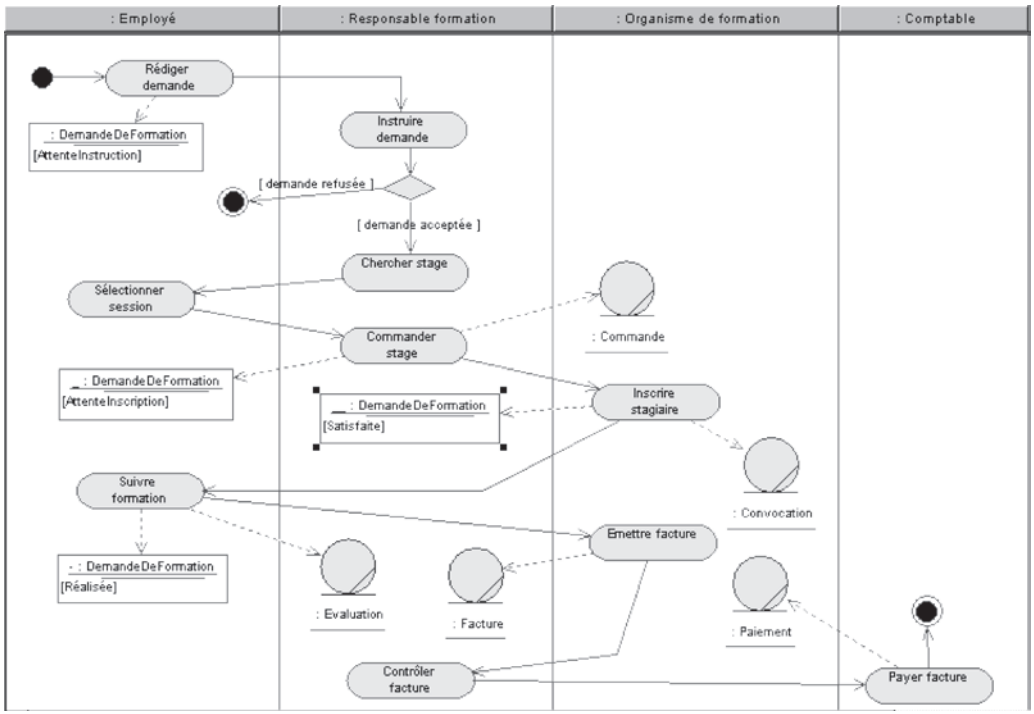
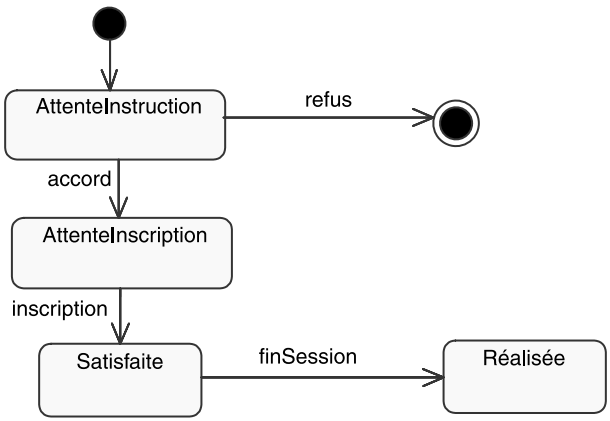


Figure 7-28.

Diagramme d'activité du processus de formation

À partir de ce diagramme d'activité, nous pouvons tout d'abord identifier quatre états principaux de la demande de formation, comme le montre la figure suivante.

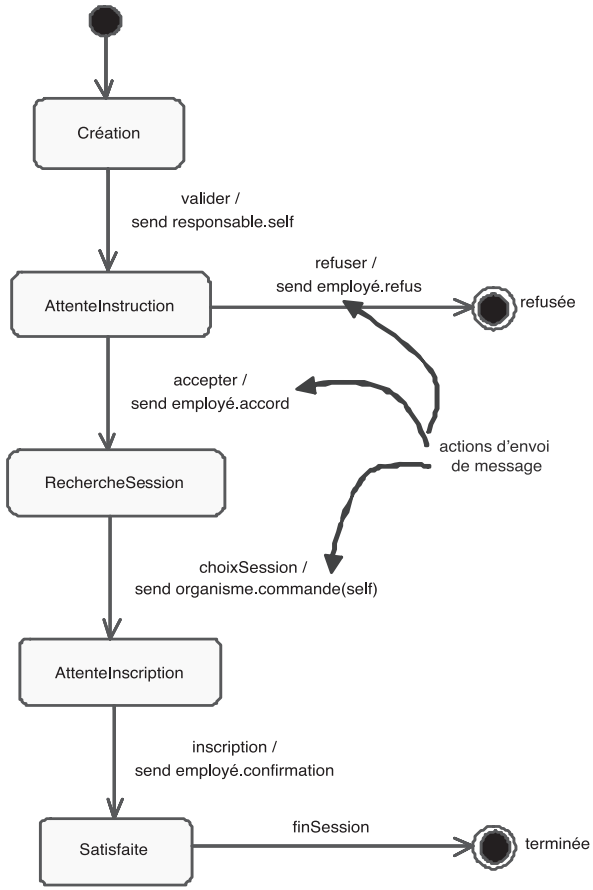
Figure 7-29.
Diagramme d'états initial de la demande de formation



En fait, en relisant attentivement la première phrase, nous nous apercevons que la demande est initiée par l'employé et envoyée au responsable formation, puis instruite par ce dernier qui transmet son accord ou son désaccord à l'intéressé. Pour pouvoir compléter le diagramme d'états, nous allons réfléchir à la suite de la vie d'une demande. Ceci nous amène à ajouter un état avant *Attente Instruction*, puisque c'est la validation de la demande qui déclenche la transmission au responsable formation. La création proprement dite de cette demande n'est pas atomique, car l'employé doit effectuer plusieurs sélections (thème, période, etc.) avant de procéder à la validation. Nous avons également identifié des actions d'envoi de message qui se matérialiseront par le mot-clé « `send` » sur les transitions du diagramme d'états.

Une nouvelle version plus complète du diagramme d'états est représentée sur la figure suivante.

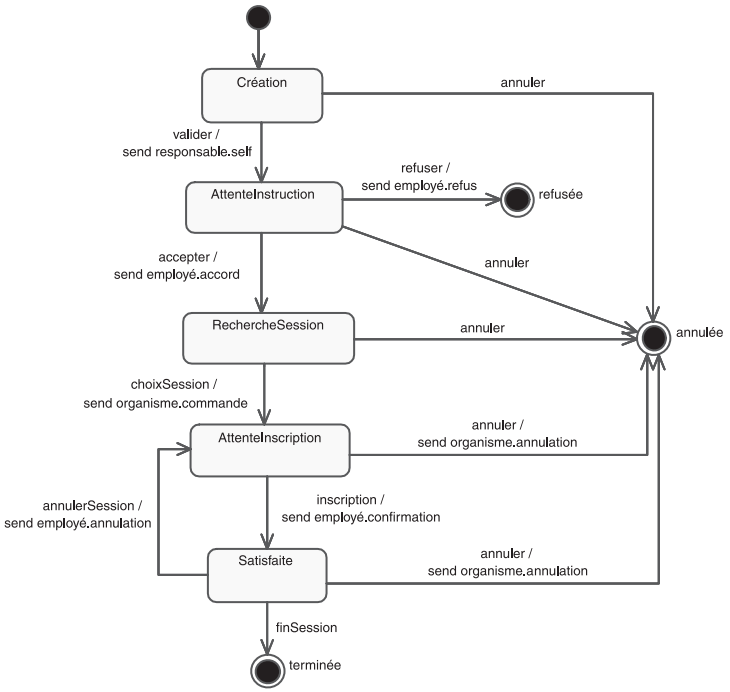
Figure 7-30.
Deuxième version du diagramme d'états de la demande de formation



Que peut-il bien nous manquer pour terminer notre diagramme d'états ? En fait, toutes les transitions d'annulation ou d'erreur. L'employé peut ainsi annuler sa demande à tout moment, l'organisme de formation peut indiquer une annulation de session, etc.

Le diagramme d'états complet est représenté sur la figure ci-après.

Figure 7-31.
Diagramme d'états complet de la demande de formation



Étape 5 – Définition des itérations

Nous allons maintenant définir des itérations à partir du travail déjà effectué et nous fixer comme objectif la conception de la première de ces itérations avec le langage Java comme cible principale.

EXERCICE 7-16.
Planification des itérations

Proposez une découpe du projet en itérations à partir du travail d'analyse précédent (cas d'utilisation et modèle métier statique). Ayez en particulier à l'esprit un des principes majeurs du processus unifié : guidé par les cas d'utilisation…

Découpez le projet en itérations.

Solution

Au vu des dépendances entre les packages métier, ainsi qu'entre les cas d'utilisation, il paraît naturel de commencer par la gestion du catalogue. En effet, les deux autres packages métier dépendent de *Catalogue de formations* et le cas d'utilisation fondamental *Demander une formation* est relié par extension au cas *Consulter le catalogue*. Nous choisissons donc de réaliser les deux cas d'utilisation qui concernent le catalogue dans la première itération.

Pour la deuxième itération, il est indispensable de s'occuper du cas d'utilisation principal du système, à savoir *Demander une formation*.

Dans une troisième itération, nous traiterons les aspects plus administratifs (inscription, etc.) avec *Traiter les demandes*.

Enfin, la dernière itération ser consacrée à *Gérer ses demandes*, un peu moins important fonctionnellement.

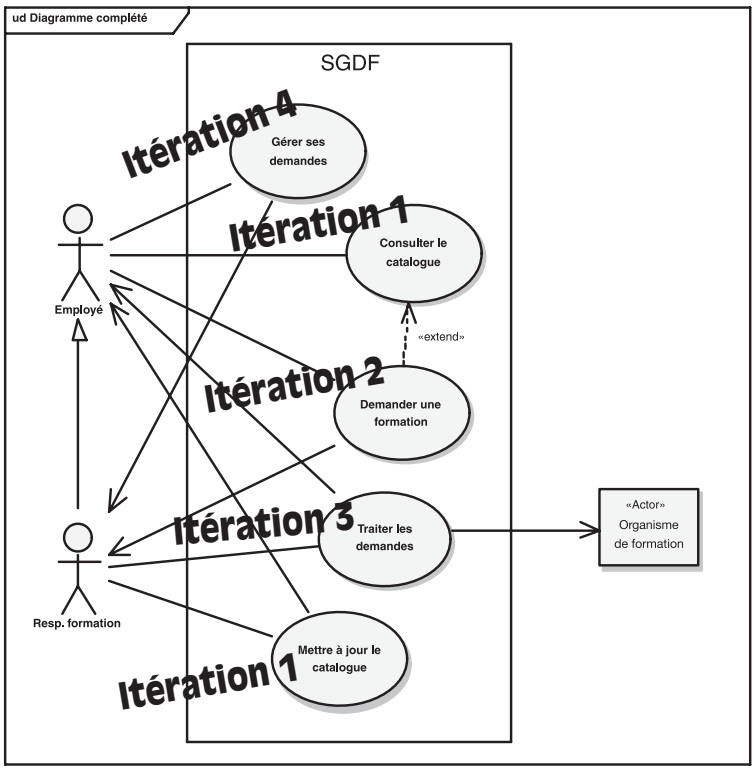


Figure 7-32.
Proposition de répartition des cas d'utilisation en itérations

Il est à noter que le service plus technique d'authentification de l'employé ou du responsable formation sur l'intranet peut être réalisé parallèlement aux cas d'utilisation fonctionnels.

Étape 6 – Définition de l'architecture système

Les systèmes informatiques modernes sont organisés en couches horizontales, elles-mêmes découpées en partitions verticales. Cette découpe est d'abord logique, puis éventuellement physique en termes de machines.

Le problème général de l'architecture des systèmes informatiques n'est pas le sujet de ce livre. Néanmoins, nous profiterons de cette quatrième partie pour faire passer quelques idées fondamentales sur les architectures en couches dites « n-tiers », ainsi que sur les diagrammes UML qui sont utiles pour cette activité.

À retenir

ARCHITECTURE EN TROIS NIVEAUX

L'architecture à trois niveaux, devenue maintenant classique, était bien, au départ, une division logique, mais elle fut interprétée à tort comme pouvant impliquer des nœuds d'exécution physiquement séparés.

Le principal objectif de cette séparation en trois couches (3-tiers) est d'isoler la logique métier des classes de présentation (IHM), ainsi que d'interdire un accès direct aux données stockées par ces classes de présentation. Le souci premier est de répondre au critère d'évolutivité : pouvoir modifier l'interface de l'application sans devoir modifier les règles métier, et pouvoir changer de mécanisme de stockage sans avoir à retoucher l'interface, ni les règles métier.

En voici l'illustration, basée sur l'étude de cas du chapitre 2 : la caisse enregistreuse de supermarché.

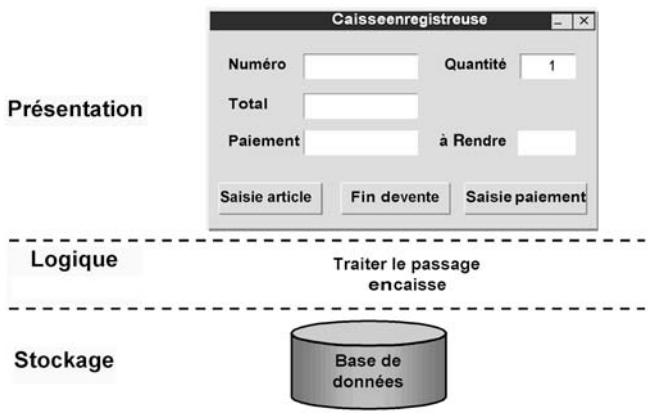
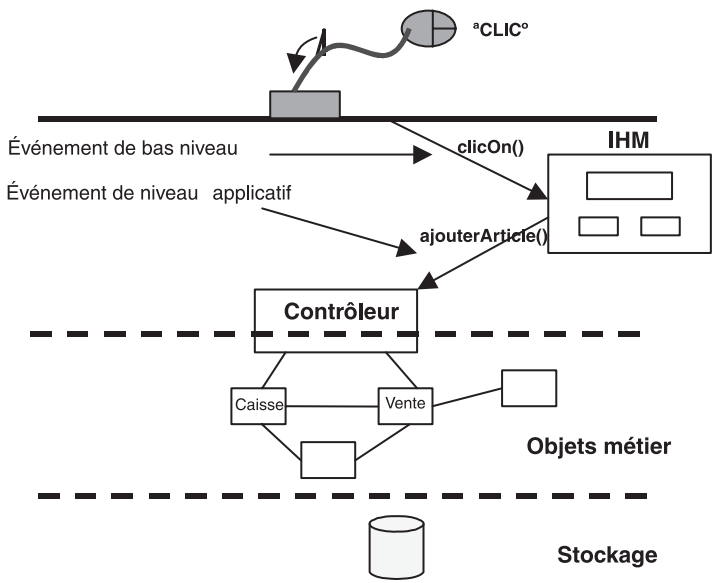


Figure 7-33.
Architecture en trois couches de la caisse enregistreuse

On ne considère plus aujourd'hui que cette décomposition en trois couches est suffisante si l'on a des objectifs importants de modularité et de réutilisation. En effet, elle conduit les objets graphiques de présentation à connaître le détail de la couche logique, ce qui nuit à leur maintenabilité et à leur réutilisabilité.

Pour améliorer cet état de fait, l'idée consiste à introduire un objet artificiel, souvent appelé « contrôleur »[6], entre les objets graphiques et les objets métier. C'est l'objet de conception *contrôleur* qui connaît maintenant l'interface des objets de la couche métier et qui joue le rôle de « façade » vis-à-vis de la couche présentation, comme cela est illustré sur la figure suivante.

Figure 7-34.
Diagramme illustrant l'ajout de l'objet contrôleur



Tous ces nouveaux objets contrôleurs, introduits en conception, vont être rassemblés dans une nouvelle couche appelée « logique applicative », qui concourt à réaliser les cas d'utilisation du système, et à isoler la couche présentation des objets métier, souvent persistants et très réutilisables.

L'architecture en trois couches peut ainsi être complétée avec deux couches supplémentaires qui représentent pour la première ces objets contrôleurs

6. Il s'agit du deuxième GRASP Pattern proposé par Larman dans [Larman 05]. Le nom de contrôleur fait également référence au pattern bien connu MVC (Model – View – Controller), ainsi qu'aux classes « control » de Jacobson, dont nous parlons plus loin dans ce chapitre.

découplant la présentation du métier et pour la seconde des services techniques généraux tels que l'accès aux bases de données, la génération de rapports, etc.

Nous allons utiliser ces principes d'architecture multicouches dans la suite du chapitre dans le cadre du système de gestion des demandes de formation.

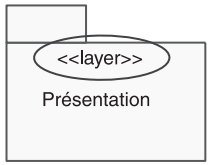
À retenir

PACKAGES, COUCHES ET PARTITIONS

En UML, le seul mécanisme de regroupement de classes disponible est le package. Par conséquent, les couches horizontales et les partitions verticales se traduisent également par des packages.

Ainsi, une architecture en couches se décrit par un diagramme statique qui ne montre que des packages et leurs dépendances. UML 2 a reconnu l'importance de ce type de diagramme de haut niveau en officialisant le diagramme de packages comme un type de diagramme UML à part entière. Vous pouvez utiliser le mot-clé « `layer` » pour distinguer les packages qui représentent les couches.

Figure 7-35.
Représentation UML d'une couche logicielle



EXERCICE 7-17.
Architecture en couches préliminaire

Proposez un diagramme d'architecture préliminaire du projet sur la base des recommandations précédentes.

Élaborez un diagramme de packages d'architecture.

solution

Nous décrivons donc un package stéréotypé « `layer` » par couche logicielle. À l'intérieur de chaque couche, nous donnons une structure préliminaire en partitions.

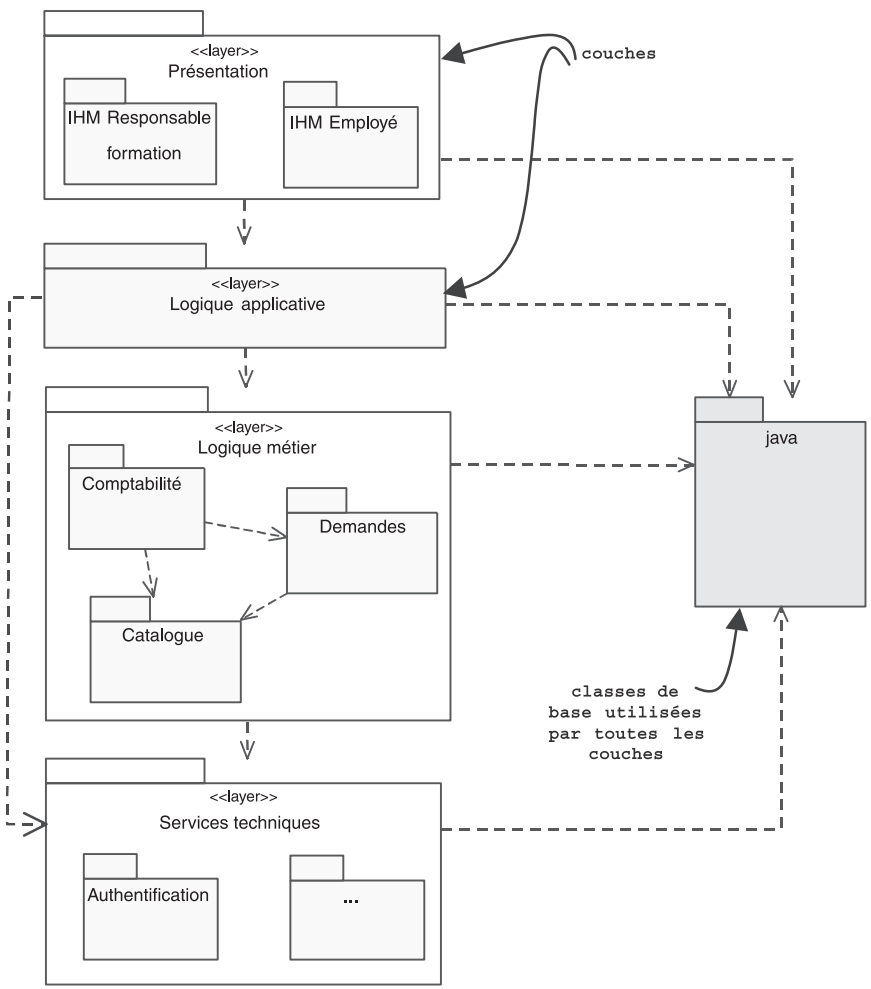


Figure 7-36. *Architecture en couches du système de gestion des demandes de formation*

La couche métier comprend *a priori* les trois packages identifiés à l'étape 3 : *Comptabilité*[7], *Demandes* et *Catalogue*. Cette découpe pourra être affinée par la suite dans notre étude ; il ne s'agit que d'une structuration préliminaire.

La couche applicative n'est pas détaillée sur le schéma. Elle peut être structurée soit à l'identique de la couche métier, soit au contraire d'un point de vue fonctionnel en calquant les packages de cas d'utilisation.

La couche présentation regroupe plutôt les classes graphiques des interfaces respectives du responsable et de l'employé.

7. La partie relative à la comptabilité ne fait en toute rigueur pas partie du système informatique, comme indiqué lors de l'étude des cas d'utilisation. Nous la conservons néanmoins dans la suite de l'exercice pour travailler sur une architecture logique de taille conséquente.

La couche des services techniques comprend au moins un package pour gérer le service technique d'authentification, identifié dès l'étape 1.

Enfin, il ne faut pas oublier les classes de base Java fournies par le JDK et qui sont utilisées par toutes les couches. La couche présentation par exemple va utiliser les classes graphiques. La couche des services techniques quant à elle va en particulier utiliser les classes JDBC d'accès aux bases de données relationnelles. Toutes les couches vont se servir des classes de base telles que les conteneurs, les dates, etc.

Il faut toutefois bien considérer que cette architecture préliminaire pourra être affinée ou modifiée (principalement au niveau des partitions à l'intérieur de chaque couche) par le travail de conception qui va suivre. N'oubliez pas que le processus d'analyse/ conception est fondamentalement itératif.

Étape 7 – Définition des opérations système (itération #1)

EXERCICE 7-18.
Opérations système

La première itération correspond aux cas d'utilisation *Consulter le catalogue* et *Mettre à jour le catalogue*. Les concernant, on a procédé à une description de haut niveau à l'étape 1 (exercice 7-3). Nous citons pour mémoire :
« Le responsable formation peut introduire une nouvelle formation dans le catalogue, modifier une formation existante ou supprimer une formation supprimée par un organisme. Il peut également modifier les regroupements de formations appelés thèmes. Il a aussi la possibilité de mettre à jour les dates et lieux des sessions.

Pour demander une formation et pour maintenir le catalogue, le système doit proposer une fonctionnalité de base de consultation du catalogue.

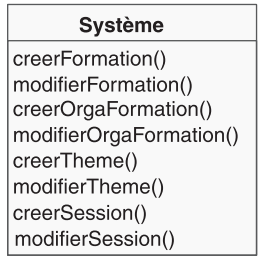
Répertoriez les opérations système pour le cas d'utilisation METTRE À JOUR LE CATALOGUE.

solution

Les opérations système pour le cas d'utilisation *Mettre à jour le catalogue* se déduisent facilement de sa description de haut niveau. Il faut néanmoins penser à la création et à la maintenance des organismes de formation, ce qui n'apparaît pas clairement dans le texte.

Les opérations système sont rassemblées sur le schéma suivant, où une classe symbolise le système vu comme une boîte noire, avec ses opérations.

Figure 7-37.
Opérations système pour le cas d'utilisation Mettre à jour le catalogue



Pour simplifier, nous avons considéré que l'action de modification inclut toujours la suppression et nous avons omis les opérations de consultation pure.

EXERCICE 7-19.
Contrat d'opération système

Nous avons identifié les opérations système à l'étape précédente. Mais comment pouvons-nous spécifier le résultat de l'exécution d'une opération système ?

À retenir

CONTRAT D'OPÉRATIONS

Larman a proposé dans [Larman 05] d'établir un « contrat » pour chaque opération système.

Un contrat d'opération décrit les changements d'état du système quand une opération système est effectuée. Ces modifications sont exprimées en termes de « post-conditions » qui détaillent le nouvel état du système après l'exécution de l'opération.

Les principales post-conditions concernent la création (ou la destruction) d'objets et de liens issus du modèle statique d'analyse, ainsi que la modification de valeurs d'attributs. Les contrats d'opérations permettent ainsi de faire le lien entre le point de vue fonctionnel/dynamique des cas d'utilisation et le point de vue statique d'analyse.

Un plan type de description textuelle de contrat d'opération est donné ci-après :

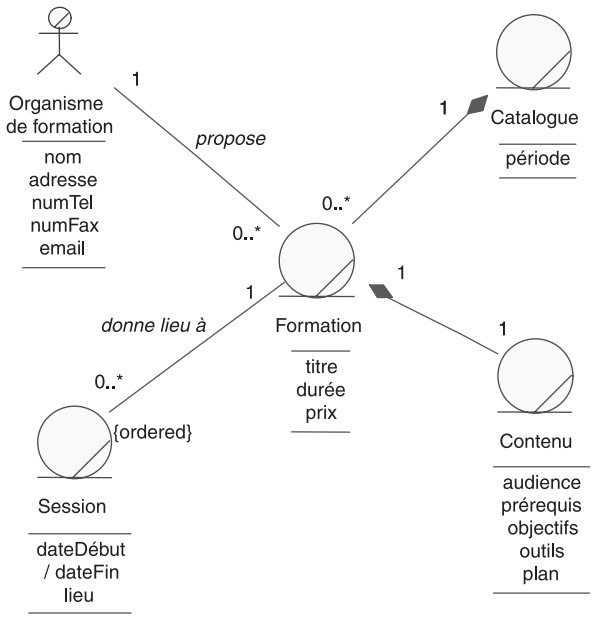
- nom
- responsabilités
- références
- pré-conditions
- post-conditions
- exceptions (optionnel)
- notes (optionnel)

Rédigez le contrat de l'opération système CREERFORMATION.

solution

En premier lieu, nous allons extraire du diagramme de classes du package *Catalogue de formations* (voir figure 7-27) la partie concernée par notre question. En effet, les opérations système *creerFormation* et *creerTheme* vont agir sur des objets et des liens provenant du diagramme suivant :

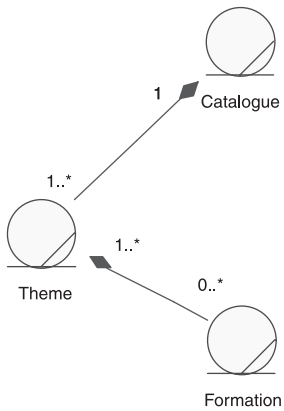
Figure 7-38.
Diagramme de classes du package Catalogue de formations



Cependant, faisait défaut la notion de thème dans notre modèle métier. Ce concept de thème est purement applicatif : il facilite le travail de l'employé lors d'une demande de formation en lui permettant de rester volontairement imprécis et de ne pas choisir une formation particulière, mais plutôt un ensemble de formations sur un sujet donné.

Nous supposerons que les thèmes structurent le catalogue, mais qu'ils ne le partitionnent pas : une formation appartient à au moins un thème. La figure suivante montre les modifications apportées par l'introduction du concept de thème.

Figure 7-39.
Introduction du concept de thème



Nous pouvons maintenant décrire le contrat de l'opération *creerFormation* :

- Nom
  creerFormation.
- Responsabilités
  Créer une nouvelle formation d'après la description fournie par l'organisme de formation concerné et la classer dans au moins un des thèmes existants.
- Références
  Cas d'utilisation *Mettre à jour le catalogue*.
- Pré-conditions
  – le catalogue de formations existe ;
  – il y a au moins un thème dans le catalogue ;
  – l'organisme fournisseur de la formation existe déjà dans le catalogue ;
  – le responsable est connecté sur l'intranet.
- Post-conditions
  – une formation f a été créée avec ses attributs ;
  – un objet contenu c a été créé avec ses attributs ;
  – c a été lié à f ;
  – f a été liée à l'organisme fournisseur ;
  – d'éventuels objets sessions ont été créés avec leurs attributs ;
  – ces objets sessions ont été liés avec f ;
  – f a été liée à au moins un thème.

Étape 8 – Diagrammes d'interaction (itération #1)

Les contrats d'opérations constituent le dernier livrable en matière d'analyse. En effet, s'ils décrivent ce que fait une opération en termes de changements d'état, ils ne doivent pas encore décrire comment elle y procède.

C'est justement le travail du concepteur de choisir comment les objets logiciels vont interagir entre eux pour réaliser telle ou telle opération. Jacobson[8] a proposé le premier (encore lui !) des stéréotypes de classes pour décrire la réalisation d'un cas d'utilisation. Nous allons nous inspirer de ses travaux pour remplacer le système vu comme une boîte noire (du point de vue de l'analyse) par des objets logiciels (du point de vue de la conception), comme cela est illustré par les diagrammes de séquence présentés ci-après.

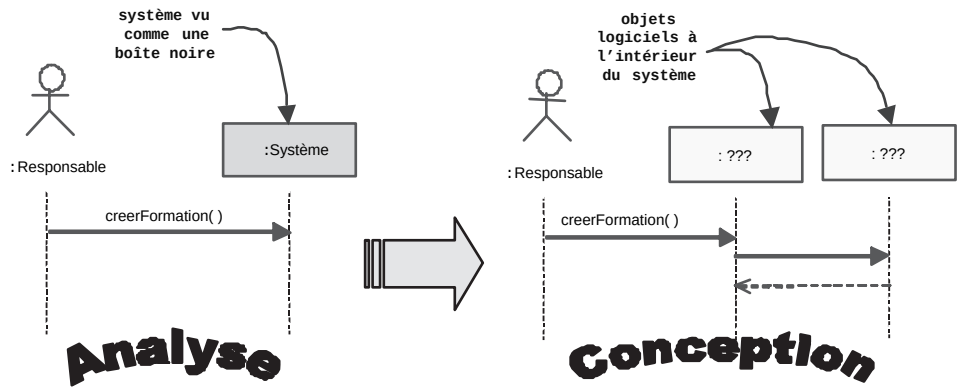


Figure 7-40.
Passage de l'analyse à la conception

À retenir

STÉRÉOTYPES DE JACOBSON

À l'intérieur du système, Jacobson distingue les trois stéréotypes suivants :

- <<boundary>> : classes qui servent à modéliser les interactions entre le système et ses acteurs ;
- <<control>> : classes utilisées pour représenter la coordination, l'enchaînement et le contrôle d'autres objets – elles sont en général reliées à un cas d'utilisation particulier ;
- <<entity>> : classes qui servent à modéliser des informations durables et souvent persistantes

8. Voir en particulier *The Unified Software Development Process*, I. Jacobson *et al.*, 1999, Addison Wesley, p. 44.

Nous utiliserons ces trois stéréotypes (avec leurs symboles graphiques associés, dans les diagrammes de séquence) pour montrer graphiquement comment un message émis par un acteur traverse les couches présentation, application et métier[9].

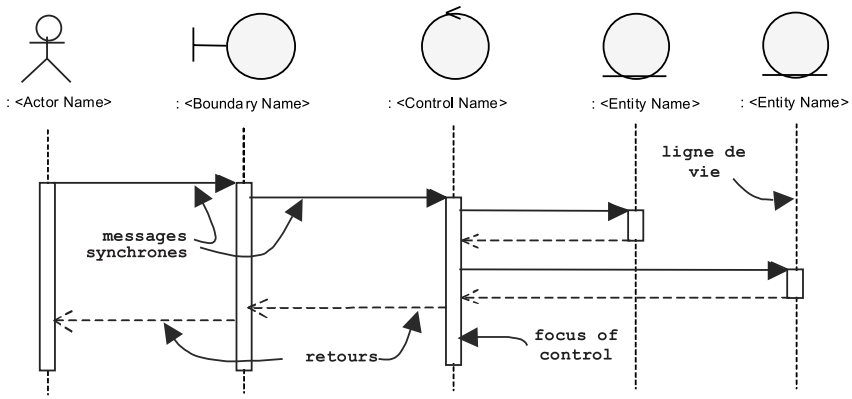


Figure 7-41.
Illustration des trois stéréotypes de Jacobson sur un diagramme de séquence

Notez la représentation des « focus of control » – bandes blanches qui représentent les périodes d'activité sur les lignes de vie des objets –, ainsi que les flèches en pointillé de retour d'invocation d'une opération.

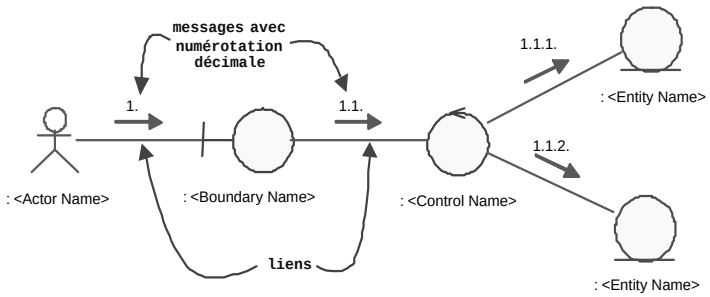


Figure 7-42.
Illustration des trois stéréotypes de Jacobson sur un diagramme de communication

Notez la numérotation décimale qui permet de montrer l'imbrication des messages, d'une façon comparable à la représentation des « focus of control » sur le diagramme de séquence précédent.

9. Même si Jacobson (et le RUP de Rational) présente ses stéréotypes comme des types de classes d'analyse, nos préférons parler pour notre part de conception préliminaire. Nous détaillerons ensuite cette conception logique en fonction de la plate-forme de développement choisie (J2EE, .NET, etc.) et remplacerons par exemple les « boundary » par des JSP (J2EE) ou des ASP (.Net), etc.

EXERCICE 7-20.
Diagrammes d'interaction de conception

Réalisez un diagramme de séquence ou un diagramme de communication qui montre la réalisation de l'opération système *creerFormation*.

Réalisez un diagramme d'interaction pour CREERFORMATION.

solution

Que devons-nous faire ? Pour le savoir, il nous faut reprendre toutes les post-conditions répertoriées lors de l'étape précédente :

- une formation f a été créée avec ses attributs ;
- un objet contenu c a été créé avec ses attributs ;
- c a été lié à f ;
- f a été liée à l'organisme fournisseur ;
- d'éventuels objets sessions ont été créés avec leurs attributs ;
- ces objets sessions ont été liés avec f ;
- f a été liée à au moins un thème.

N'oubliez pas que les post-conditions ne représentent que le nouvel état du système à la fin de l'exécution de l'opération système. Elles ne sont absolument pas ordonnées : c'est le rôle du concepteur de choisir maintenant quel objet doit réaliser chaque action, et dans quel ordre.

La post-condition fondamentale concerne bien la création de l'objet formation, avec son contenu et ses sessions, puis la définition de ses liens avec les autres objets du catalogue tels que les thèmes et les organismes. Il est raisonnable de penser que la création de l'objet formation f va se faire en quatre étapes :

1. initialisation de l'objet f et de ses attributs ;
2. création de son contenu ;
3. création des sessions ;
4. validation de f.

Voyons par le détail une solution possible pour la première étape, mettant en jeu deux objets <<`boundary`>>, un <<`control`>> et un <<`entity`>>.

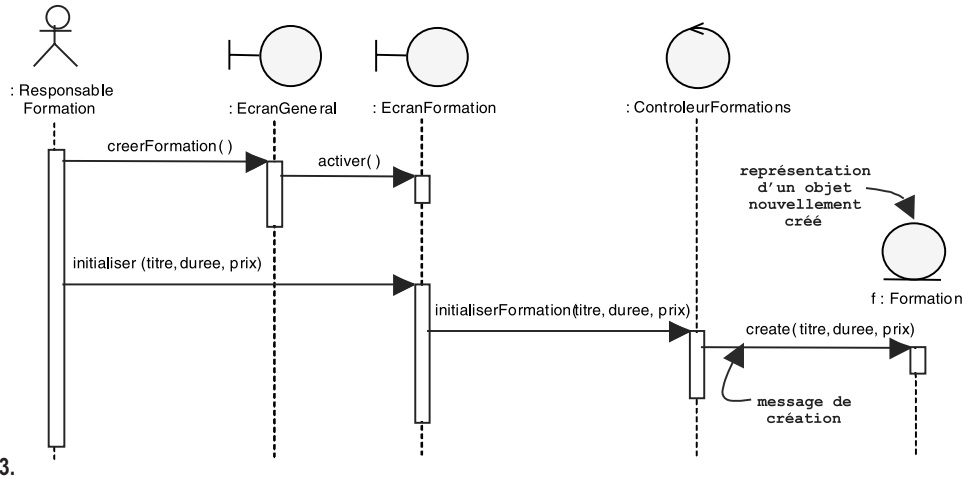


Figure 7-43.
Diagramme de séquence de l'initialisation de f

Le même scénario peut se représenter par un diagramme de communication, comme ceci :

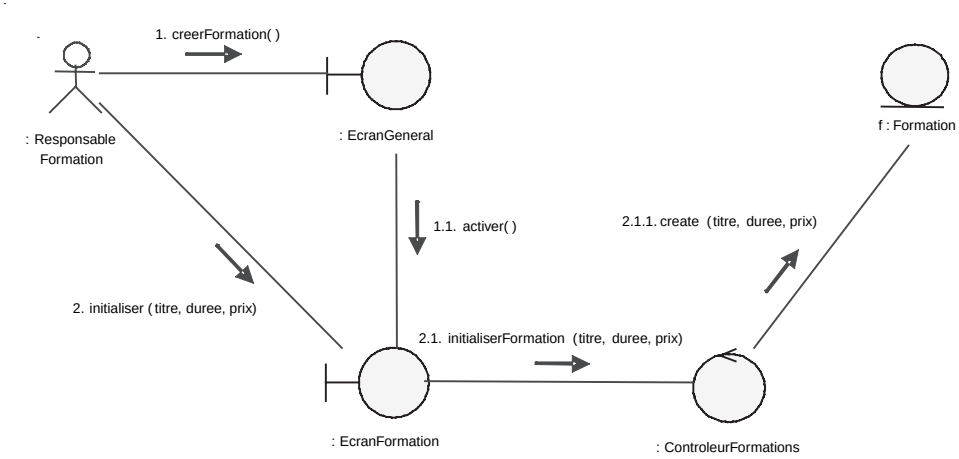


Figure 7-44.
Diagramme de communication de l'initialisation de f

Poursuivons par la création du contenu. Le diagramme de séquence complété devient alors :

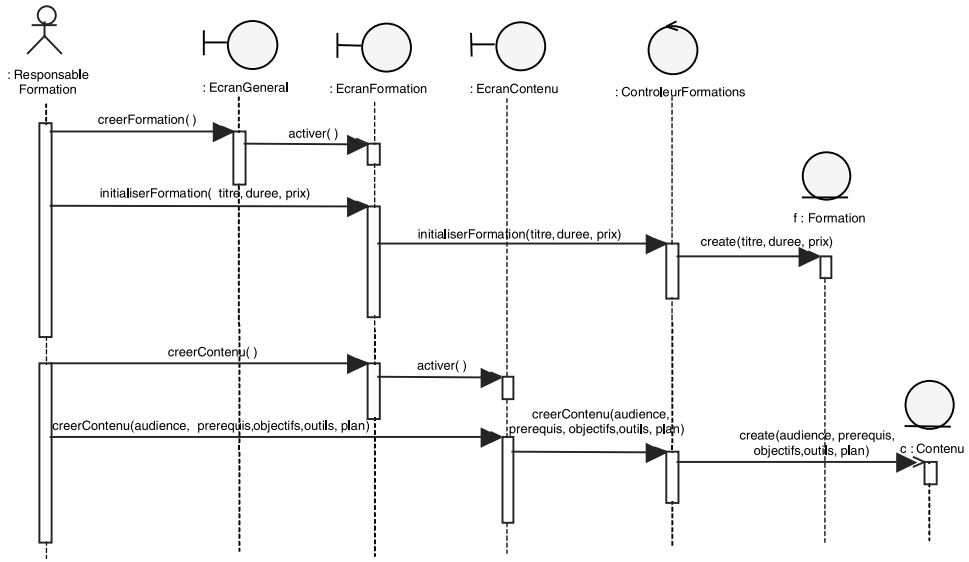


Figure 7-45.
Diagramme de séquence de l'initialisation de f et de la création de son contenu

On notera que le diagramme de séquence devient de moins en moins lisible au fur et à mesure que nous ajoutons des objets… C'est pour cette raison simple que le diagramme de communication est intéressant en conception : il nous permet de disposer nos objets dans les deux dimensions afin d'améliorer la lisibilité du schéma.

Le diagramme de communication qui correspond au diagramme de séquence précédent est donné sur la figure suivante, à titre de comparaison.

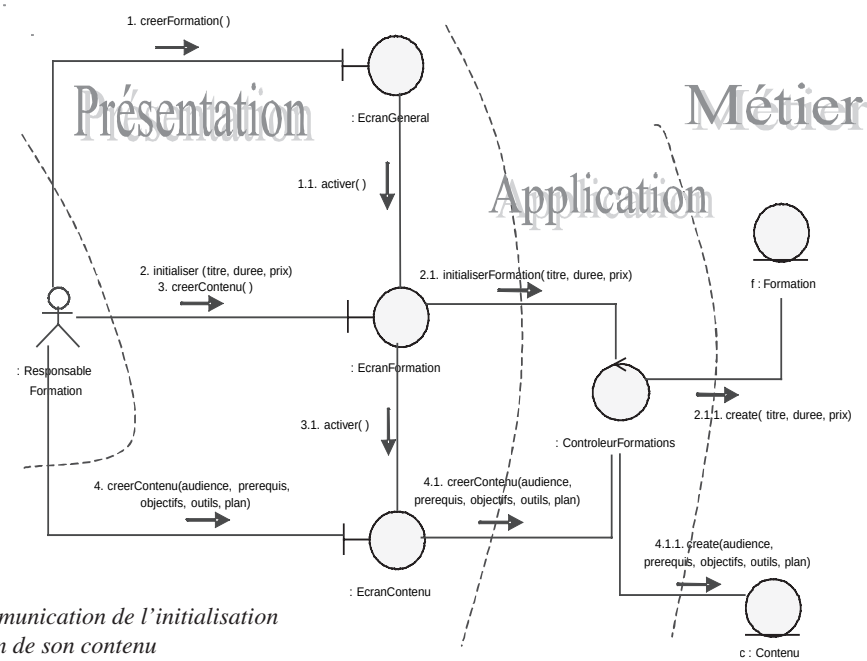


Figure 7-46.
Diagramme de communication de l'initialisation de f et de la création de son contenu

On notera que le diagramme de *communication* présenté ci-dessus permet de bien différencier visuellement les couches d'objets.

Poursuivons maintenant par la création des sessions.

À retenir

MULTI-OBJET

Pour indiquer que la formation f sera liée à une collection de sessions, nous utilisons un *multi-objet*. Le *multi-objet* est une construction UML 1 qui représente en un seul symbole plusieurs objets de la même classe[b]. Cela permet de ne pas ajouter trop tôt de classes de conception détaillée liées au langage de programmation (comme Vector de la STL C++ ou *ArrayList* en Java, etc.). Un multi-objet peut également représenter l'abstraction entière d'une connexion à une base de données.

Nous avions d'ailleurs oublié dans le diagramme 7-45 de créer la collection vide de sessions lors de la création de f. Il suffit ensuite après la création de chaque session de l'ajouter à la collection. Nous utilisons pour cela une opération générique *add()* : voir figure 7-47.

b. Cette notation très répandue se trouve avoir disparu dans la version 2.0 du langage UML. Nous continuons cependant à l'utiliser dans le reste du chapitre pour des raisons de lisibilité et de clarté pédagogique. Le lecteur curieux se référera au chapitre suivant pour voir par quoi cette notation UML 1 a été remplacée en UML 2.

Le diagramme de *communication* présente un autre avantage sur le diagramme de séquence : il permet de représenter les relations structurelles entre les objets. Nous avons par exemple fait figurer les liens de composition autour de l'objet formation f, pour mieux préparer la traçabilité avec notre futur diagramme de classes de conception.

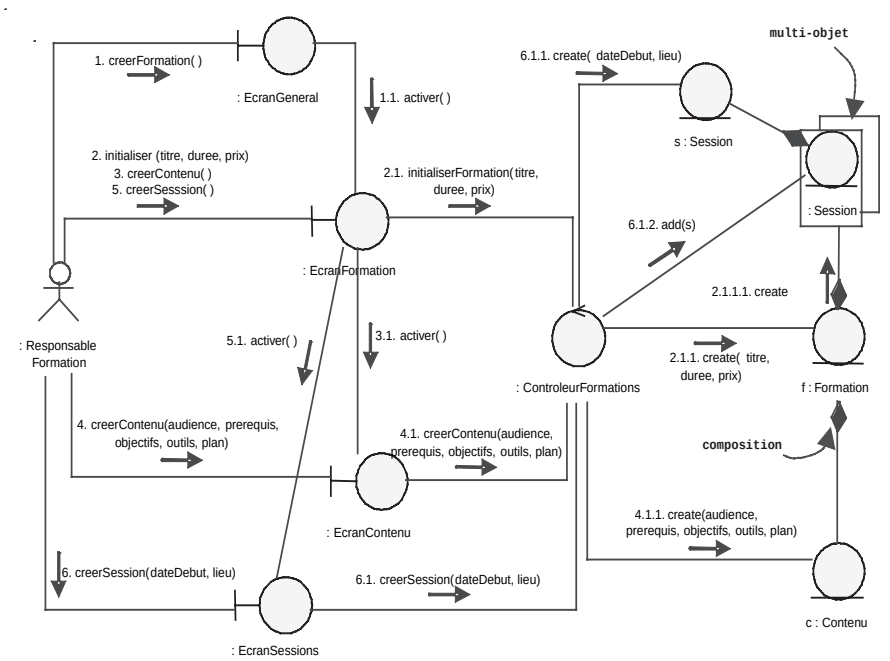


Figure 7-47.
Diagramme de communication de l'initialisation de f, de la création de son contenu et d'une session

Il ne nous reste plus qu'à lier la formation f à un thème existant et à valider la création. En comparant le travail réalisé aux post-conditions qui étaient souhaitées au début de la réponse, on peut constater que la suivante n'a pas été prise en compte : « f a été liée à l'organisme fournisseur ». Nous l'ajoutons simplement aux responsabilités du contrôleur, lors de la création de la formation.

Le diagramme de communication complet de l'opération système *creerFormation* se trouve sur la figure suivante. Remarquez la quantité d'information, assez considérable, qui arrive à être représentée de façon encore à peu près lisible sur une seule page. Néanmoins, nous atteignons là les limites du diagramme de communication.

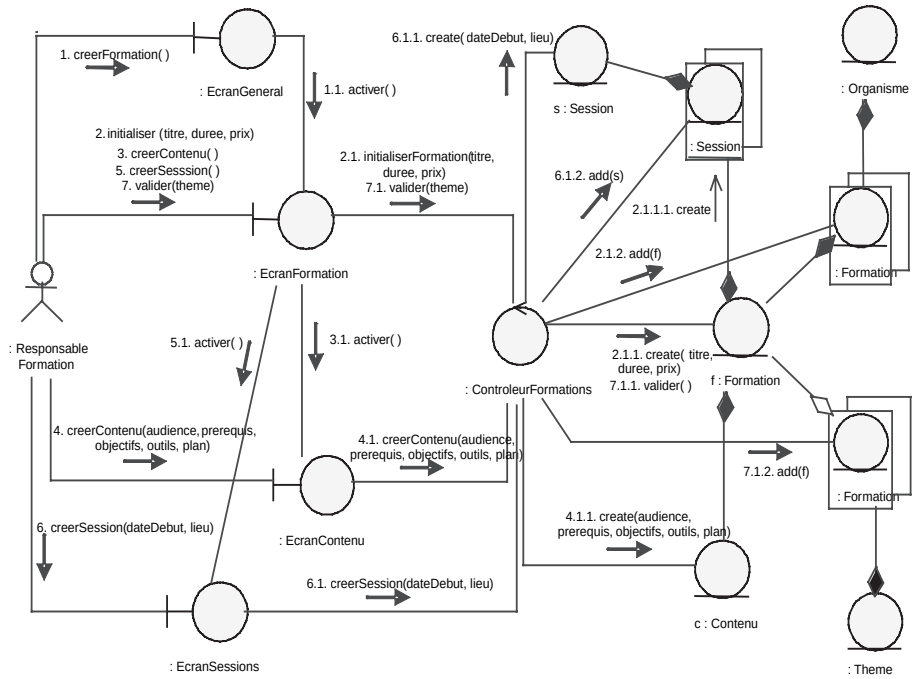


Figure 7-48.
Diagramme de communication complet de l'opération système creerFormation

Une idée intéressante pour améliorer la lisibilité du diagramme consiste à le découper en deux en prenant l'objet contrôleur comme charnière :

- une première partie afin de spécifier la cinématique de l'interface homme-machine avec les acteurs, les objets <<boundary>> et l'objet <<control>> ;
- une seconde partie afin de spécifier la dynamique des couches applicatives et métier avec l'objet <<control>> et les objets <<entity>>.

Les diagrammes de communication partiels ainsi obtenus sont montrés sur les deux figures qui suivent.

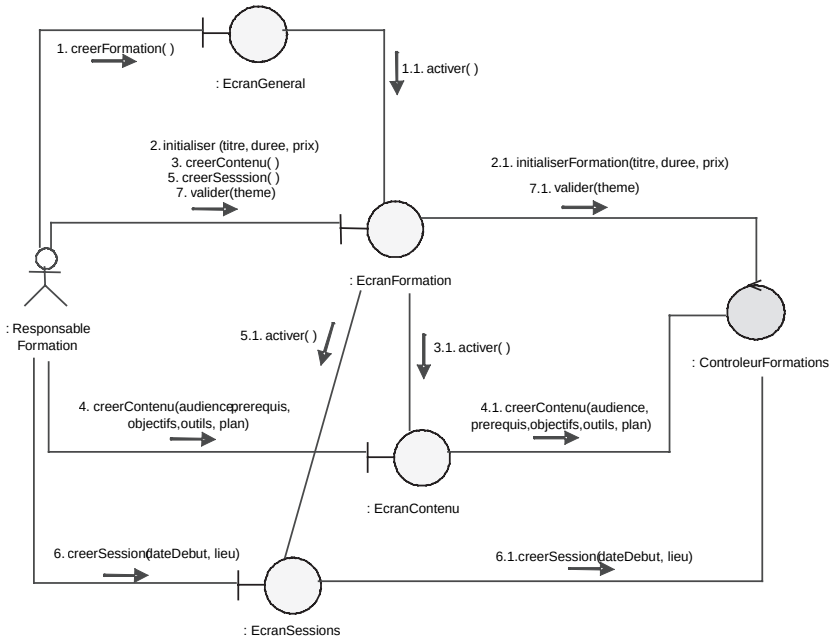


Figure 7-49.
Diagramme de communication partiel de l'opération système creerFormation *: couche présentation et lien avec la couche applicative*

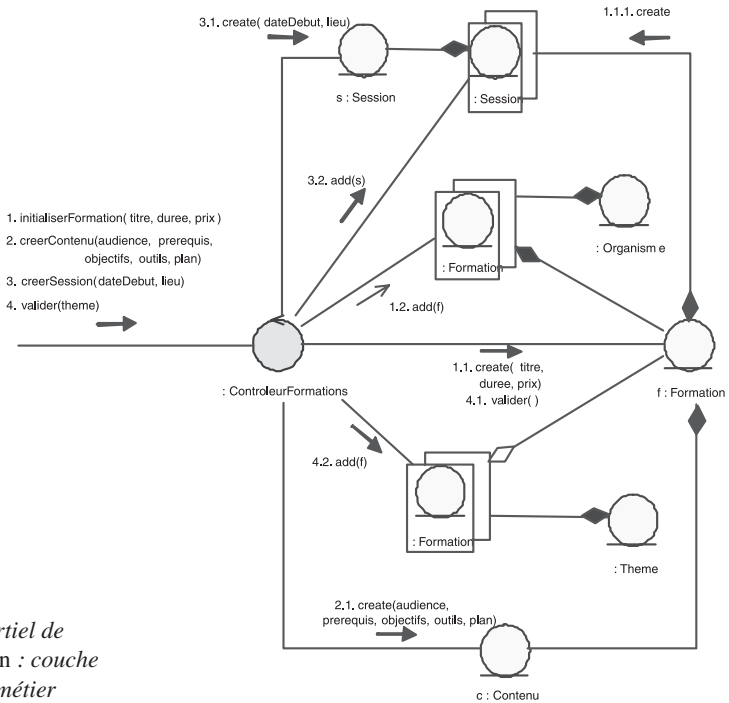


Figure 7-50.
Diagramme de communication partiel de l'opération système creerFormation *: couche applicative et lien avec la couche métier*

Étape 9 – Diagrammes de classes de conception (itération #1)

Chaque opération système va donner lieu à une étude dynamique sous la forme d'un diagramme d'interaction, comme cela a été le cas pour l'opération *creerFormation* lors de l'exercice précédent.

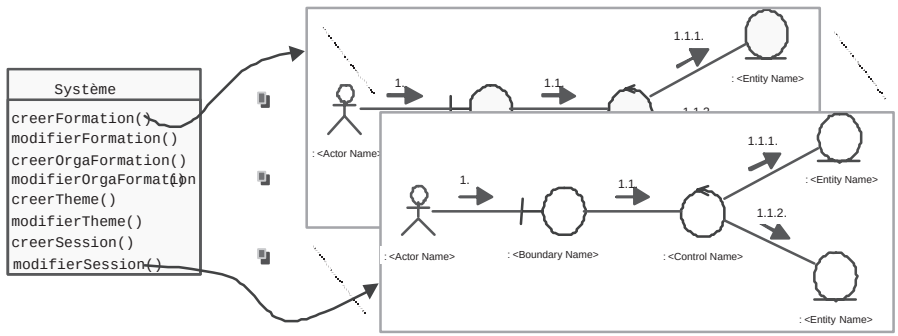
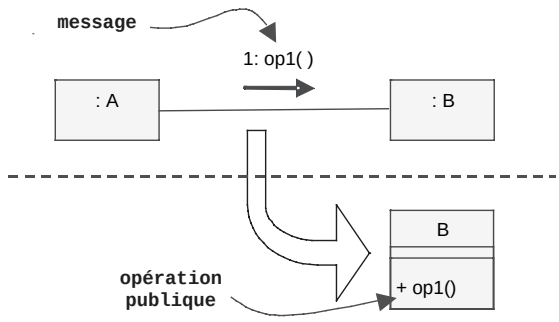


Figure 7-51.
Démarche de conception initialisée par les opérations système

Les diagrammes d'interaction ainsi réalisés vont permettre d'élaborer des diagrammes de classes de conception, et ce en ajoutant principalement les informations suivantes aux classes issues du modèle d'analyse :

- les opérations : un message ne peut être reçu par un objet que si sa classe a déclaré l'opération publique correspondante ;

Figure 7-52.
Rapport entre message et opération



- la navigabilité des associations ou les dépendances entre classes, suivant que les liens entre objets sont durables ou temporaires, et en fonction du sens de circulation des messages.

À retenir

LIENS DURABLES OU TEMPORAIRES

Un lien durable entre objets va donner lieu à une association navigable entre les classes correspondantes ; un lien temporaire (par paramètre : « `parameter` », ou variable locale : « `local` ») va donner lieu à une relation de dépendance.

Sur l'exemple présenté ci-après, le lien entre l'objet :A et l'objet :B devient une association navigable entre les classes correspondantes. Le fait que l'objet :A reçoive en paramètre d'un message une référence sur un objet de la classe C induit une dépendance entre les classes concernées.

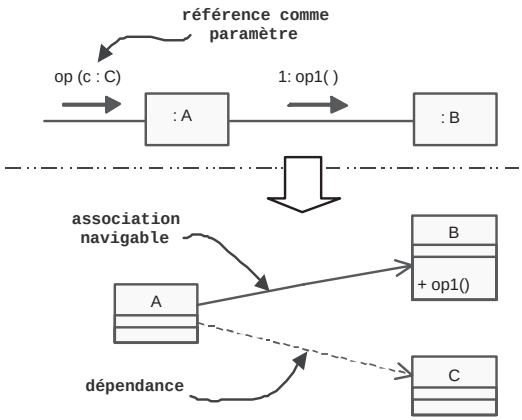


Figure 7-53.
Rapport entre les liens entre objets et les relations entre classes

Notez enfin que nous recommandons de ne pas ajouter les classes qui correspondent aux multi-objets dans le diagramme de classes de conception de façon à rester le plus longtemps possible indépendant du langage de programmation cible.

EXERCICE 7-21.
Diagramme de classes de conception

En appliquant les règles énoncées plus haut, Réalisez un fragment de diagramme de classes de conception à partir du diagramme de communication partiel 7-50 (*creerFormation*).

Commencez un diagramme de classes de conception pour CREERFORMATION.

solution

Le diagramme de communication 7-50 nous permet tout d'abord d'ajouter les opérations dans les classes comme cela est montré sur le schéma suivant.

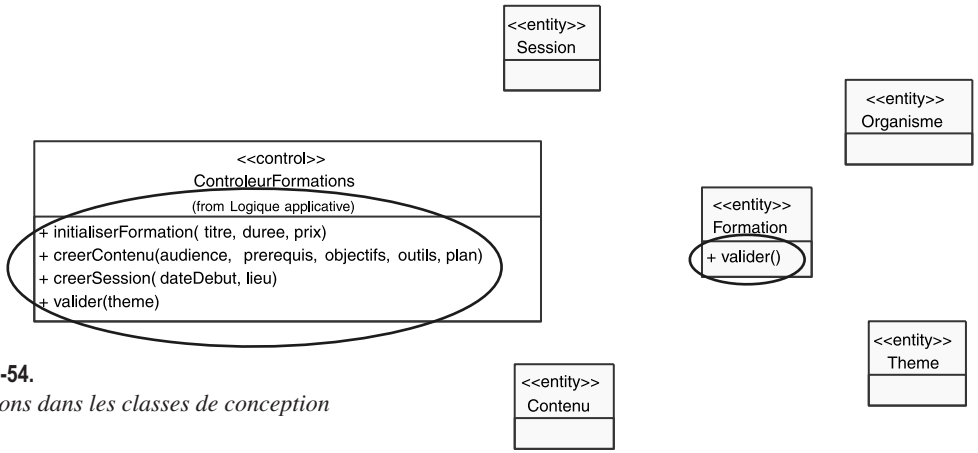


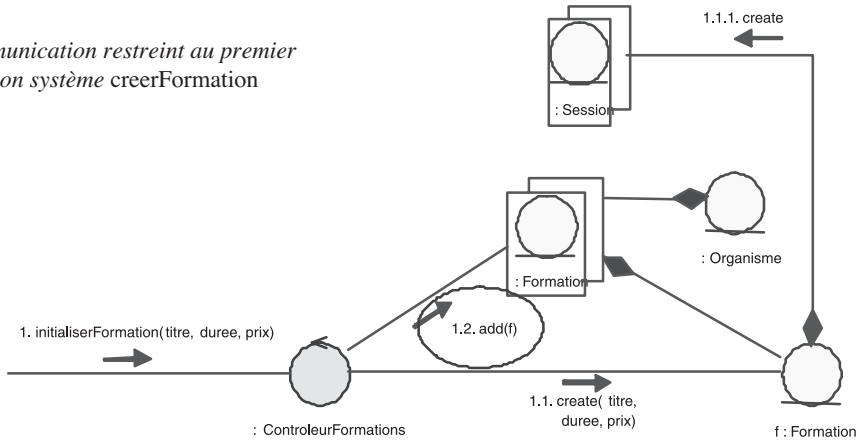
Figure 7-54.
Opérations dans les classes de conception

On notera qu'il y a peu d'opérations puisque nous ne faisons pas apparaître :

- Les opérations de création (message `create`),
- Les opérations génériques sur les classes conteneurs (*add*(), etc.).

Néanmoins, on peut remarquer un premier problème : comment l'objet *controleurFormations* peut-il ajouter la nouvelle formation f au multi-objets de l'organisme correspondant sans posséder une référence sur cet organisme ?

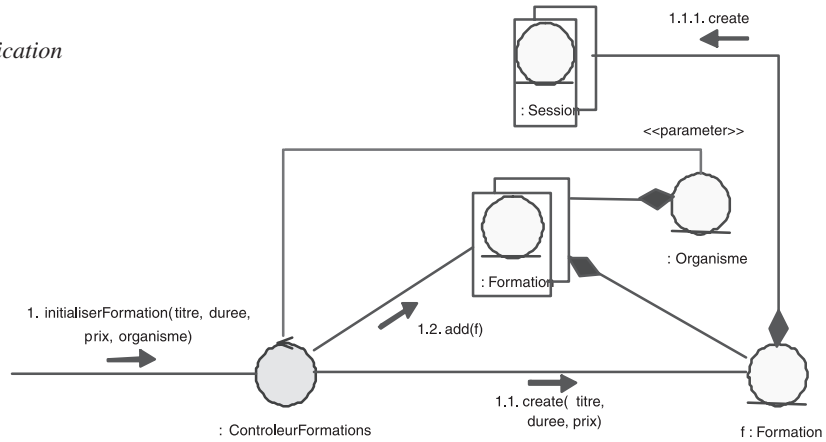
Figure 7-55.
Diagramme de communication restreint au premier message de l'opération système creerFormation



Cela signifie que nous devons ajouter un paramètre à l'opération *initialiserFormation* : une référence vers un organisme existant.

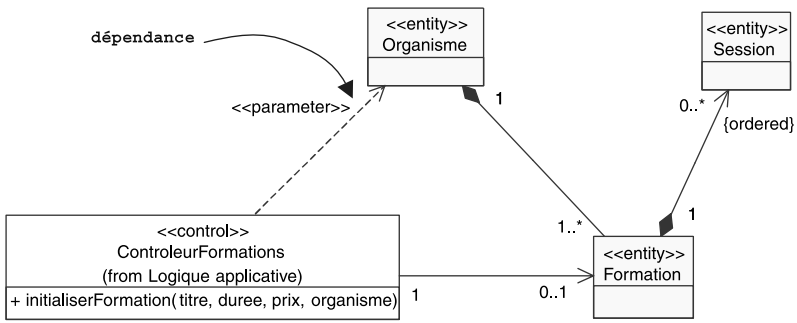
Si nous utilisons également les mots-clés « `parameter` » et « `local` » pour indiquer les liens temporaires entre objets, le diagramme de communication précédent est modifié comme suit :

Figure 7-56.
Diagramme de communication complété



Nous allons maintenant compléter le diagramme de classes en ajoutant les relations entre classes : association (avec ses variantes : agrégation ou composition) et dépendance. Le travail est facilité en ceci que nous avions déjà indiqué les liens de composition et d'agrégation sur le diagramme de communication.

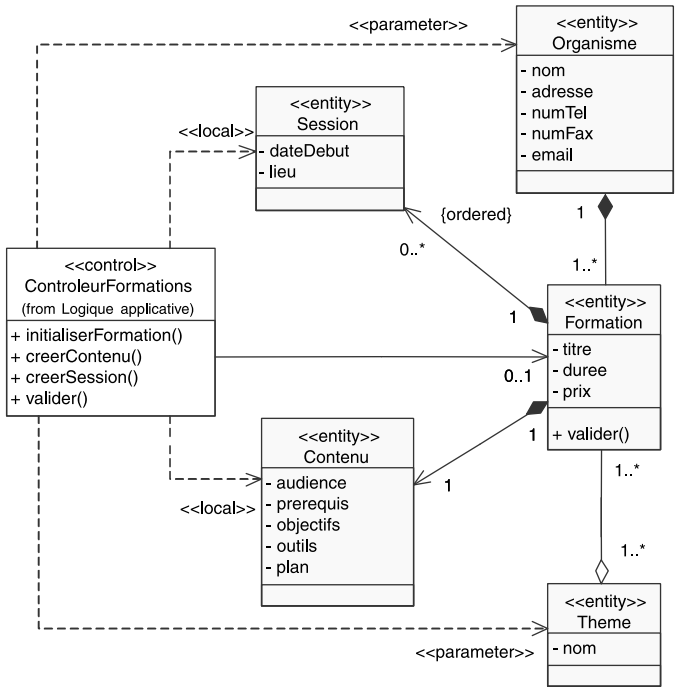
Figure 7-57.
Diagramme de classes complété d'après le diagramme de communication précédent



On notera l'utilisation du mot-clé « `parameter` » sur la dépendance entre les classes *ControleurFormations* et *Organisme*, pour refléter le type de lien temporaire qui existe entre les objets correspondants dans le diagramme de communication.

Si nous appliquons maintenant la même démarche sur l'ensemble du diagramme 7-50, nous obtenons le diagramme de classes de conception ci-après. Il faut noter que nous avons fait figurer les attributs dans les classes, mais pas les paramètres des opérations (pour simplifier le schéma).

Figure 7-58.
Diagramme de classes de conception complété



Bien sûr, ce diagramme est encore dans un état tout à fait provisoire :

- les choix de navigabilité des associations sont loin d'être définitifs – ils pourront être modifiés par l'étude des autres opérations système ;
- les dépendances se transformeront peut-être en associations si les objets requièrent un lien durable, et non un simple lien temporaire, dans le cadre d'autres opérations système.

EXERCICE 7-22.
Amélioration de la conception

À partir des diagrammes réalisés lors de la question précédente, proposez des améliorations à la conception objet qu'ils illustrent.

Améliorez les diagrammes de conception précédents.

solution

Le diagramme de classes 7-58 présente une classe *ControleurFormations* couplée à toutes les autres classes ! Cette propriété est tout à fait contraire à un principe fondamental de la conception objet, appelé couramment « couplage faible ».

À retenir

COUPLAGE FAIBLE

Le « couplage » représente une mesure de la quantité d'autres classes auxquelles une classe donnée est connectée , dont elle a connaissance, ou dont elle dépend.

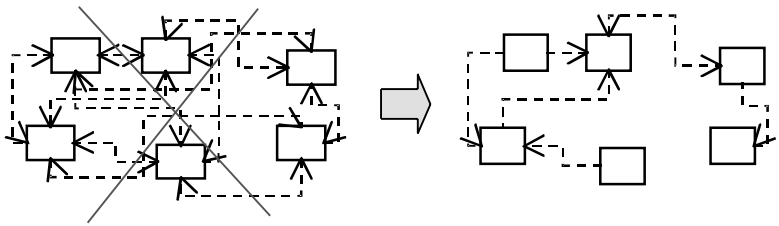


Figure 7-59.
Couplage entre classes

Conserver un couplage faible est un principe qu'il faut bien avoir à l'esprit pour toutes les décisions de conception ; c'est un objectif sous-jacent à évaluer d'une façon continue. En effet, en y pourvoyant, on obtient en général une application plus évolutive et plus facile à maintenir.

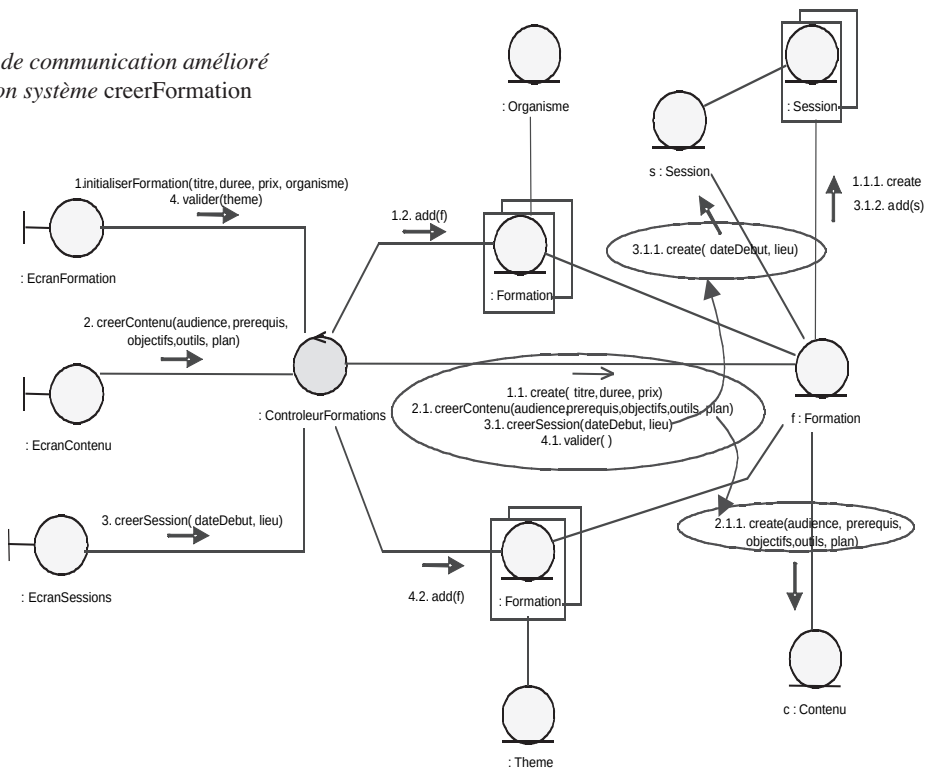
La notion d'objet « contrôleur » que nous avons détaillée précédemment (voir figure 7-34) est un bon exemple de moyen mis en œuvre pour minimiser le couplage entre les couches logicielles.

Essayons de voir s'il n'existe pas un moyen simple de réduire le couplage de la classe *ControleurFormations* sans augmenter pour autant celui des autres classes.

Reprenons le diagramme de communication 7-50. L'objet *ControleurFormations* est-il vraiment le mieux placé pour créer les objets *Contenu* et *Session* ? Ne pourrait-il pas plutôt déléguer cette responsabilité de création à l'objet *Formation* qui va de toute façon être ensuite lié d'une façon durable à son contenu et à ses sessions ? De cette manière, nous enlevons les deux dépendances entre *ControleurFormations* et *Contenu* et *Session*, sans en ajouter, puisque *Formation* est déjà couplée à *Contenu* et *Session* par des relations fortes de composition.

Le diagramme de communication peut donc être modifié comme le montre la figure suivante.

Figure 7-60.
Diagramme de communication amélioré de l'opération système creerFormation



Le diagramme de classes de conception est ainsi allégé de deux dépendances, simplement parce que l'objet *ControleurFormations* a su déléguer une partie de ses responsabilités à l'objet *Formation*. En fait, cet exemple simple est tout à fait représentatif du travail itératif d'évaluation et d'amélioration que doit faire tout concepteur en matière de conception objet.

Pour terminer, nous allons compléter le diagramme de classes amélioré par les types des attributs, et procéder à la signature complète des opérations (paramètres avec leur type). Il faut bien noter que nous utilisons des types simples du langage Java (comme *int* et *short*), des classes de base Java (comme *String* et *Date*), des classes « primitives » utilisateur (comme *Numero* et *Email*), qu'il faudra définir précisément, et enfin des classes du modèle (comme *Theme* et *Organisme*).

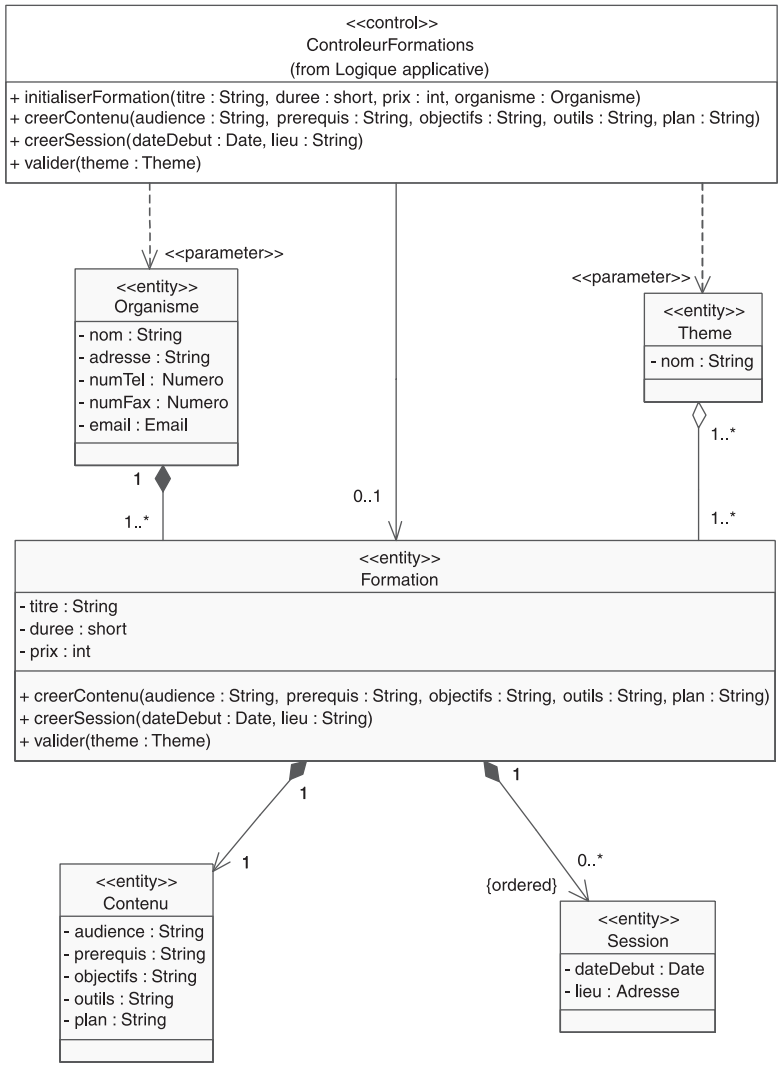


Figure 7-61.
Diagramme de classes de conception amélioré

Étape 10 – Définition des opérations système (itérations #2 et #3)

À ce stade, nous tenons pour acquis que l'itération 1 a été réalisée avec succès. Les cas d'utilisation *Consulter le catalogue* et *Mettre à jour* ont été conçus, implémentés et testés. Le package métier *Catalogue de formations* a été affiné et enrichi en conséquence. Un état possible de son diagramme de classes de conception (ne montrant que les classes `<<entity>>`) est présenté sur la figure suivante.

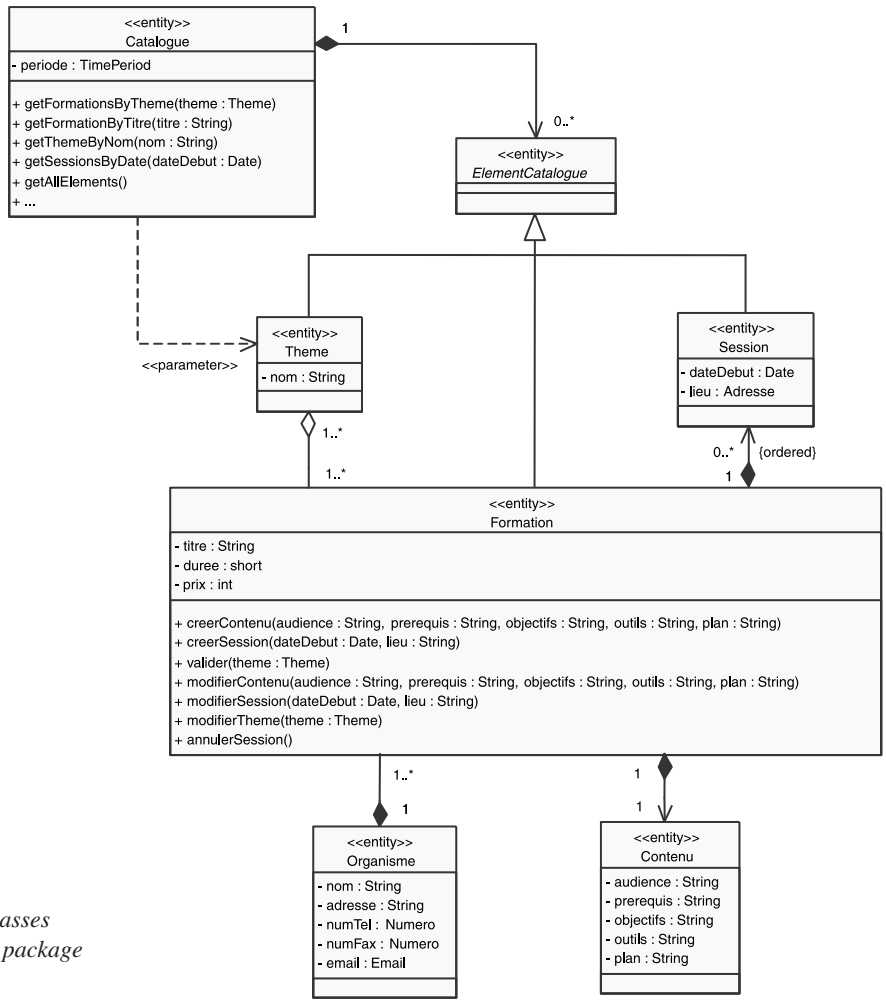


Figure 7-62.
Diagramme de classes de conception du package Catalogue

On notera que de nombreuses opérations ont été ajoutées, ainsi qu'une classe abstraite *ElementCatalogue* qui englobe les thèmes, les formations et les sessions, dans l'optique

de la création d'une demande de formation par un employé à partir d'un élément quelconque du catalogue de formations.

Le stockage du catalogue dans une base de données relationnelle est opérationnel, ainsi que l'interface homme-machine des deux cas d'utilisation.

Le mécanisme d'authentification est également disponible dans une version qui va nous permettre de continuer notre travail.

EXERCICE 7-23.
Opérations système

Nous allons maintenant concevoir et implémenter les deuxième et troisième itérations[10]. Commençons donc par le cas d'utilisation *Demander une formation*. Sa description de haut niveau a été réalisée à l'étape 1 (exercice 7-3). Nous la citons pour mémoire :

« L'employé peut consulter le catalogue, et sélectionner un thème, ou une formation, ou même une session particulière. La demande est automatiquement enregistrée par le système et transmise au responsable formation par e-mail. »

Pour le second cas, *Traiter les demandes*, la description était la suivante :

« Le responsable formation va utiliser le système pour indiquer aux employés sa décision (accord ou refus). En cas d'accord sur une session précise, le système va envoyer automatiquement par fax une demande d'inscription sous forme de bon de commande à l'organisme concerné. »

Listez les opérations système pour les cas d'utilisation DEMANDER UNE FORMATION et TRAITER LES DEMANDES.

solution

Nous allons tout d'abord réaliser un diagramme de séquence système du scénario nominal du cas d'utilisation *Demander une formation*.

Nous supposons que l'employé peut effectuer plusieurs demandes, d'où le cadre `loop`. Les sélections d'une formation et d'une session sont optionnelles, d'où les deux cadres `opt` imbriqués. Notez également la possibilité qu'offre UML 2 de représenter graphiquement les préconditions en haut de la ligne de vie du système boîte noire.

10. Nous groupons les itérations par deux uniquement à des fins pédagogiques, afin que les différents modèles réalisés soient le plus intéressant possible.

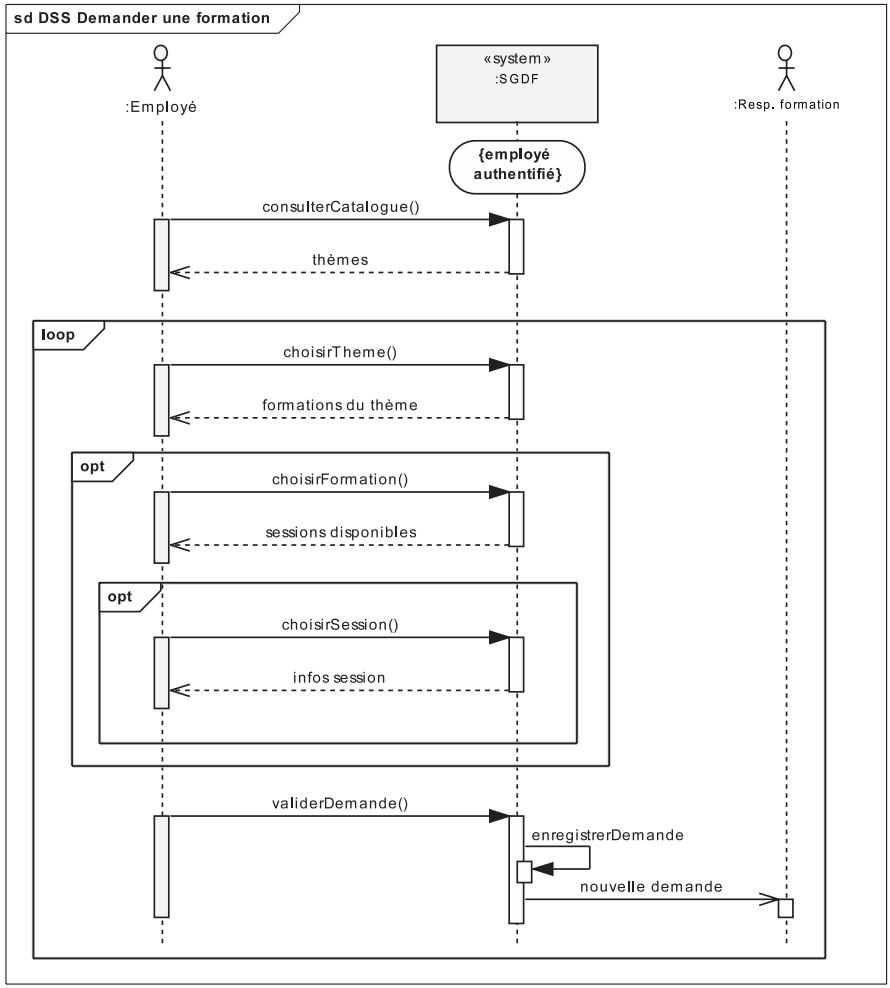


Figure 7-63. *Diagramme de séquence système de* Demander une formation

Nous allons ensuite réaliser un diagramme de séquence système simplifié (sans traiter le cas d'acceptation de demande incomplète) du cas d'utilisation *Traiter les demandes*.

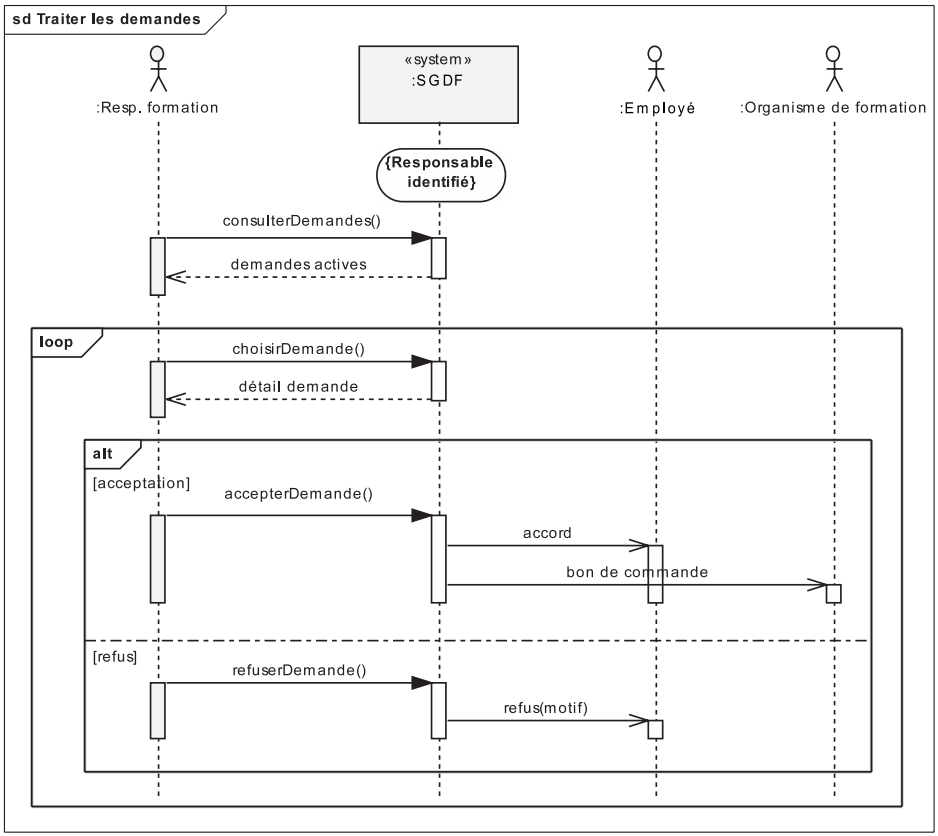


Figure 7-64.
Diagramme de séquence système de Traiter les demandes

Les principales opérations système pour les cas d'utilisation *Demander une formation* et *Traiter les demandes* sont donc listées sur les deux schémas précédents (flèches entrant dans le SGDF).

Étape 11 – Contrats d'opérations (itérations #2 et #3)

EXERCICE 7-24.
Contrats d'opérations système

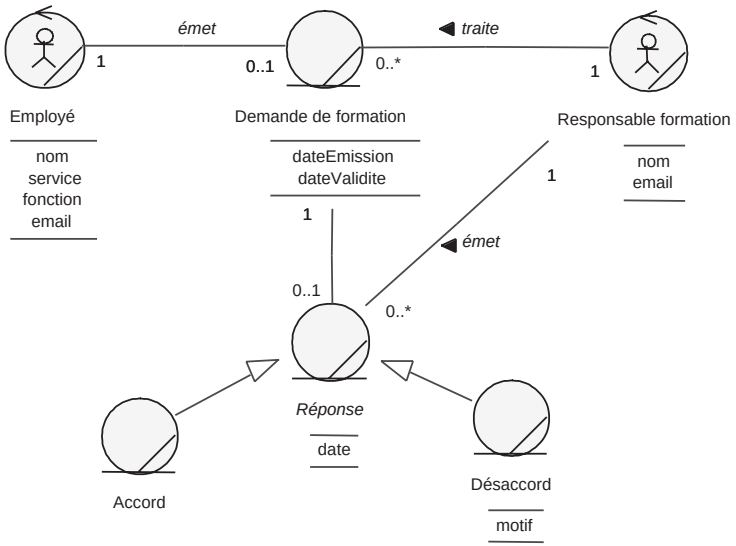
Utilisez le plan type précédent de contrat d'opération système.

Rédigez les contrats de VALIDERDEMANDE et REFUSERDEMANDE.

Solution

En premier lieu, nous allons extraire du diagramme de classes du package *Demandes de formation* (voir figure 7-26) la partie concernée par notre question. En effet, les opérations système *validerDemande* et *refuserDemande* vont agir sur des objets et des liens émanant du diagramme ci-après.

Figure 7-65.
Extrait du diagramme de classes déduit de la modélisation métier



Établissons tout d'abord le contrat de l'opération *validerDemande* :

- Nom
  validerDemande.
- Responsabilités
  Créer une demande initiale d'après les éléments du catalogue et la transmettre au responsable formation pour instruction.
- Références
  Cas d'utilisation *Demander une formation*.
- Pré-conditions
  – le catalogue de formations existe ;
  – l'employé est connecté sur l'intranet ;
  – un objet e représentant l'employé existe dans l'application.
- Post-conditions
  – une demande de formation ddf a été créée ;
  – les attributs *dateValidite* et *dateEmission* de ddf ont été initialisés ;
  – ddf a été liée à l'employé e ;
  – ddf a été liée à un élément du catalogue de formation (c'est un aspect qui faisait défaut dans le diagramme de la modélisation métier) ;
  – un e-mail contenant ddf a été transmis au responsable formation.

- Exceptions
  - L'employé peut annuler sa création de demande à tout moment avant de valider.

Poursuivons par le contrat de l'opération *refuserDemande* :

- Nom
  refuserDemande.
- Responsabilités
  Refuser une demande transmise par un employé et lui retourner le motif du refus.
- Références
  Cas d'utilisation *Traiter les demandes*.
- Pré-conditions
  - une demande de formation ddf existe ;
  - le responsable est connecté sur l'intranet ;
  - un objet e représentant l'employé existe dans l'application et est relié à ddf.
- Post-conditions
  - la demande de formation ddf a été détruite ;
  - un objet Désaccord d a été créé ;
  - les attributs *date* et *motif* de d ont été initialisés ;
  - un e-mail contenant d a été transmis à l'employé e.
- Exceptions
  - Néant.

Étape 12 – Diagrammes d'interaction (itérations #2 et #3)

EXERCICE 7-25.
Diagramme de communication

Comme à l'étape 8 pour les itérations 1 et 2, nous allons poursuivre notre travail de conception en réalisant un diagramme de communication.

Élaborez un diagramme de communication qui réalise VALIDERDEMANDE.

solution

La démarche est analogue à celle que nous avons adoptée pour l'exercice 7-20. Le diagramme de communication représentant l'initialisation de la demande de formation par l'employé est tout à fait similaire à celui de la figure 7-44.

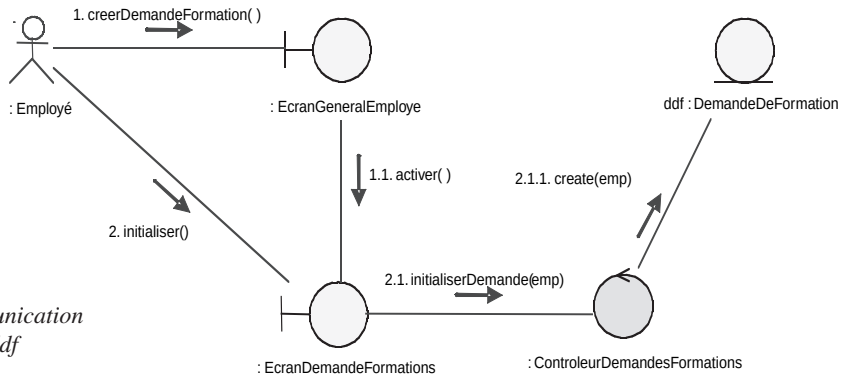


Figure 7-66.
Diagramme de communication de l'initialisation de ddf

Continuons par l'établissement du lien avec un élément du catalogue de formation, puis par le positionnement des attributs *dateValidite* et *dateEmission*, et enfin l'envoi du message au responsable.

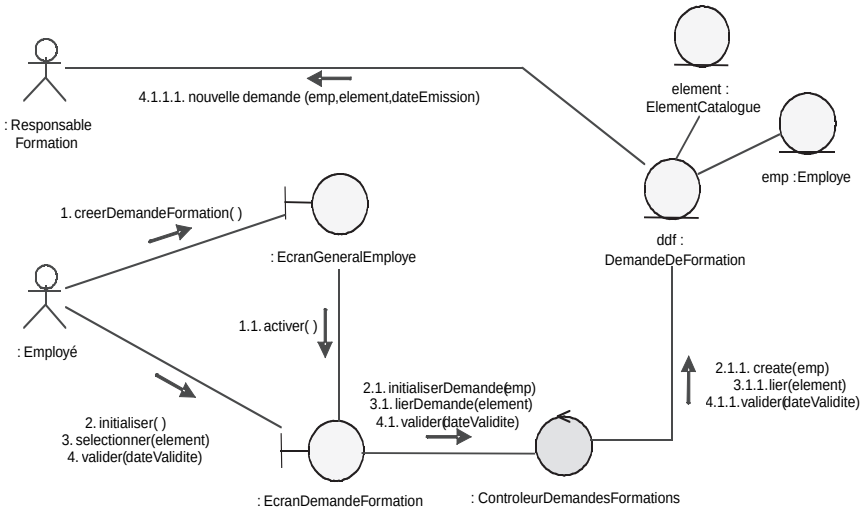


Figure 7-67.
Diagramme de communication complet de l'opération système validerDemande

Étape 13 – Diagrammes de classes de conception (itérations #2 et #3)

EXERCICE 7-26.
Diagramme de classes de conception

Sur le modèle de la figure 7-62 et en extrapolant à partir de la réponse précédente mais aussi en vous appuyant sur votre connaissance du sujet, réalisez un diagramme de classes de conception du package *Demandes*.

Vous pouvez vous référer également au diagramme d'états de la question 7-15.

Élaborez le diagramme de classes de conception du package DEMANDES.

solution

Nous avons déjà passé en revue les subtilités du diagramme de classes aux chapitres 3 et 4. La classe d'association *Inscription* ne vous surprendra donc pas. On notera la façon dont nous avons complété le compartiment opérations de la classe *DemandeDeFormation*, en particulier avec les opérations privées, nécessaires pour l'envoi des messages aux acteurs.

On notera encore que nous avons fait figurer dans le diagramme les classes *Session* et *ElementCatalogue* bien qu'elles n'appartiennent pas au package courant. En effet, il est important de montrer leurs relations avec des classes du package *Demandes* pour justifier ensuite le sens des dépendances entre les packages englobants. Précisément, il ne faut représenter que les associations navigables, les dépendances ou les généralisations qui pointent vers les classes externes à celles du package concerné.

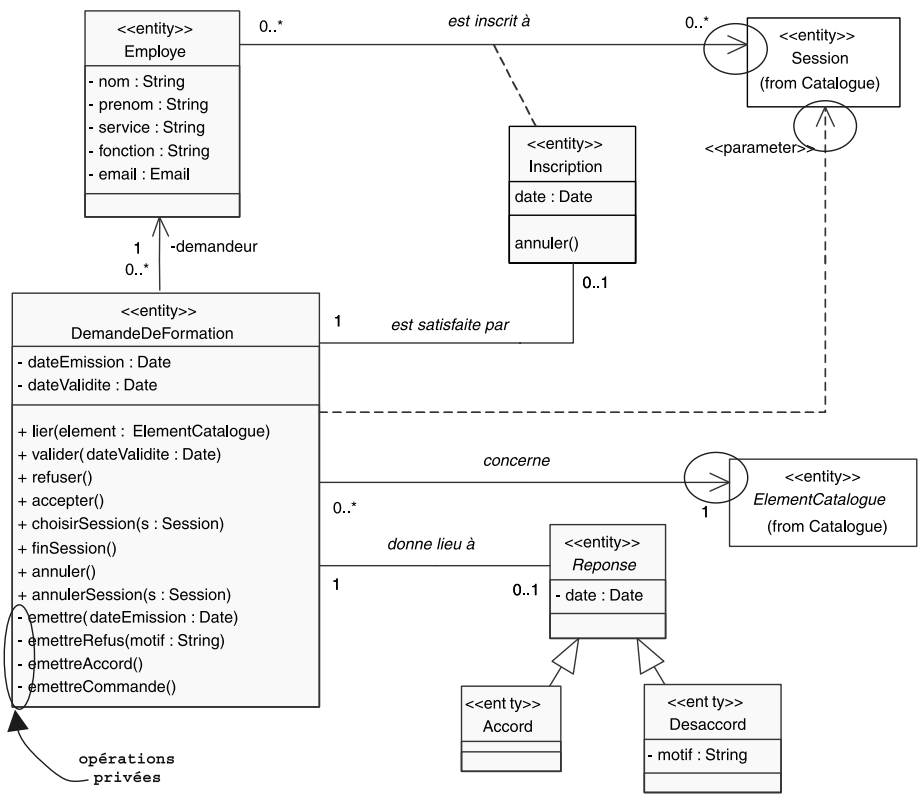


Figure 7-68. *Diagramme de classes de conception du package* Demandes

Étape 14 – Retour sur l'architecture

EXERCICE 7-27.
Diagramme de packages

Reprenez la figure 7-36 qui représentait l'architecture en couches du système et montrez toutes les classes que nous avons identifiées à l'intérieur des packages correspondants. Ne tenez pas compte de la couche services techniques, pas plus que des classes de base Java.

Complétez le diagramme de packages d'architecture.

solution

Il suffit de recenser toutes les classes que nous avons utilisées dans nos différents diagrammes, et de les représenter à l'intérieur du package adéquat.

L'architecture logique détaillée des trois premières couches est montrée sur la figure suivante.

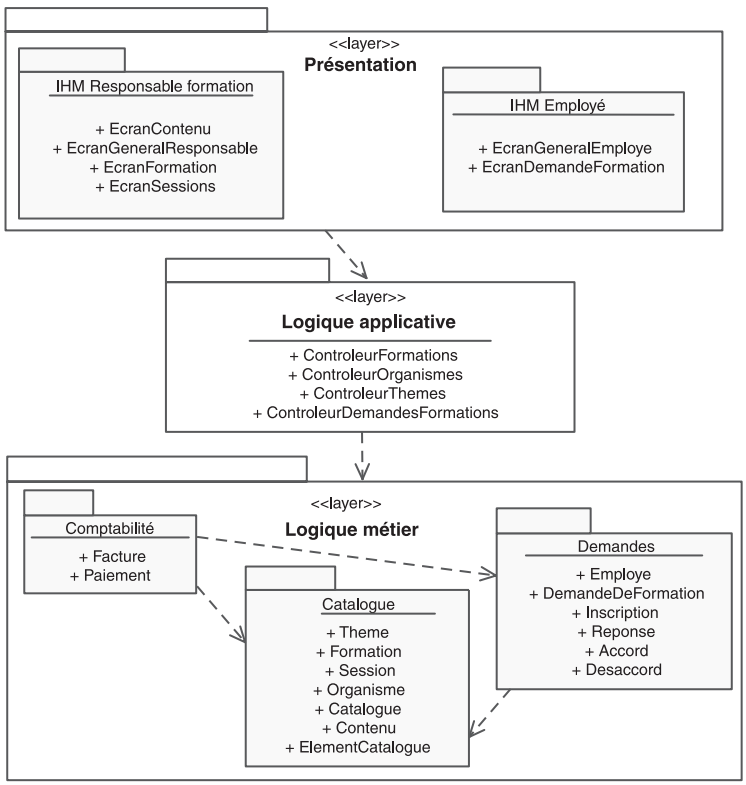


Figure 7-69.
Détail de l'architecture en couches des trois premières itérations

Étape 15 – Passage au code objet

Les modèles de conception que nous avons réalisés permettent de produire, d'une manière aisée, du code dans un langage de programmation objet tel que Java ou C# :

- Les diagrammes de classes permettent de décrire le squelette du code, à savoir toutes les déclarations.

À retenir

PRODUCTION DU SQUELETTE DE CODE JAVA (OU C#) À PARTIR DES DIAGRAMMES DE CLASSES

En première approche :

– la classe UML devient une classe Java (idem en C#) ;

– les attributs UML deviennent des variables d'instances Java (idem en C#[c]) ;

– les opérations UML deviennent des méthodes Java (idem en C#).

On notera que les rôles navigables produisent des variables d'instances, tout comme les attributs, mais avec un type utilisateur au lieu d'un type simple. Le constructeur par défaut est implicite.

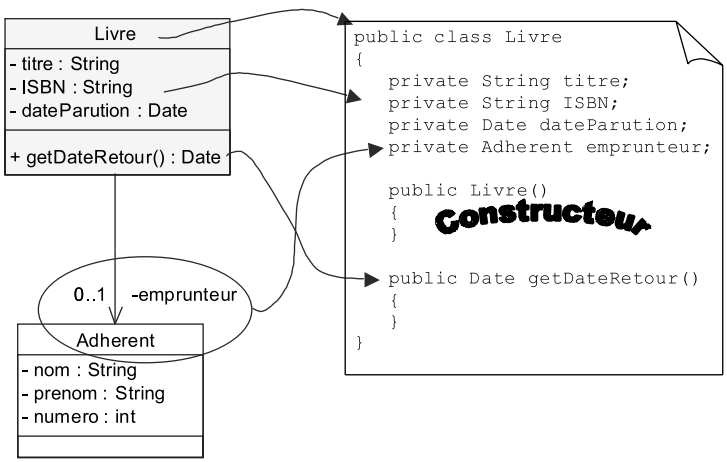


Figure 7-70.
Squelette du code Java de la classe Livre

c. On peut également utiliser le concept C# de *property*, qui permet une meilleure encapsulation. Attention, le concept d'attribut existe en C#, mais ne signifie pas du tout la même chose qu'en UML !

- Avec les diagrammes d'interaction, il est facile d'écrire le corps des méthodes, en particulier la séquence d'appels de méthodes sur les objets qui collaborent.

À retenir

PRODUCTION DU CORPS DES MÉTHODES
À PARTIR DES DIAGRAMMES D'INTERACTION

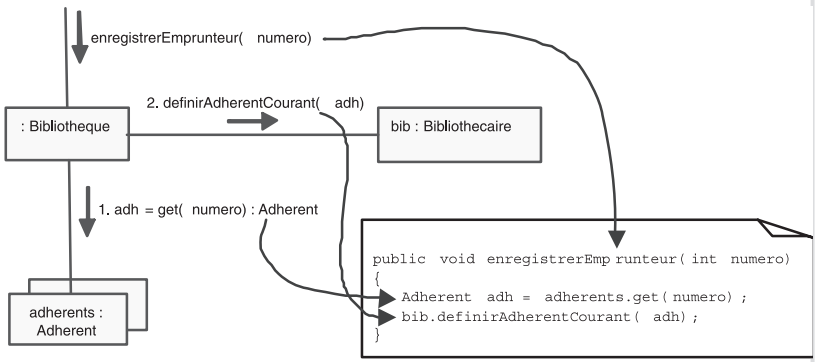


Figure 7-71.
Corps de la méthode enregistrerEmprunteur *en Java (idem en C# !)*

EXERCICE 7-28.
Production d'un squelette de code Java

À partir de la figure 7-68, proposez un squelette de code Java pour la classe *DemandeDeFormation*.

Écrivez le squelette Java de la classe DEMANDEDEFORMATION.

solution

Reprenons le schéma 7-68 en enlevant ce qui ne concerne pas la classe *DemandeDeFormation*. Les règles précédentes suffisent pour produire le squelette de la classe en Java. La seule difficulté tient à ce qu'il ne faut pas oublier la directive d'importation pour les relations avec les classes qui appartiennent à d'autres packages, ainsi que pour les classes de base Java.

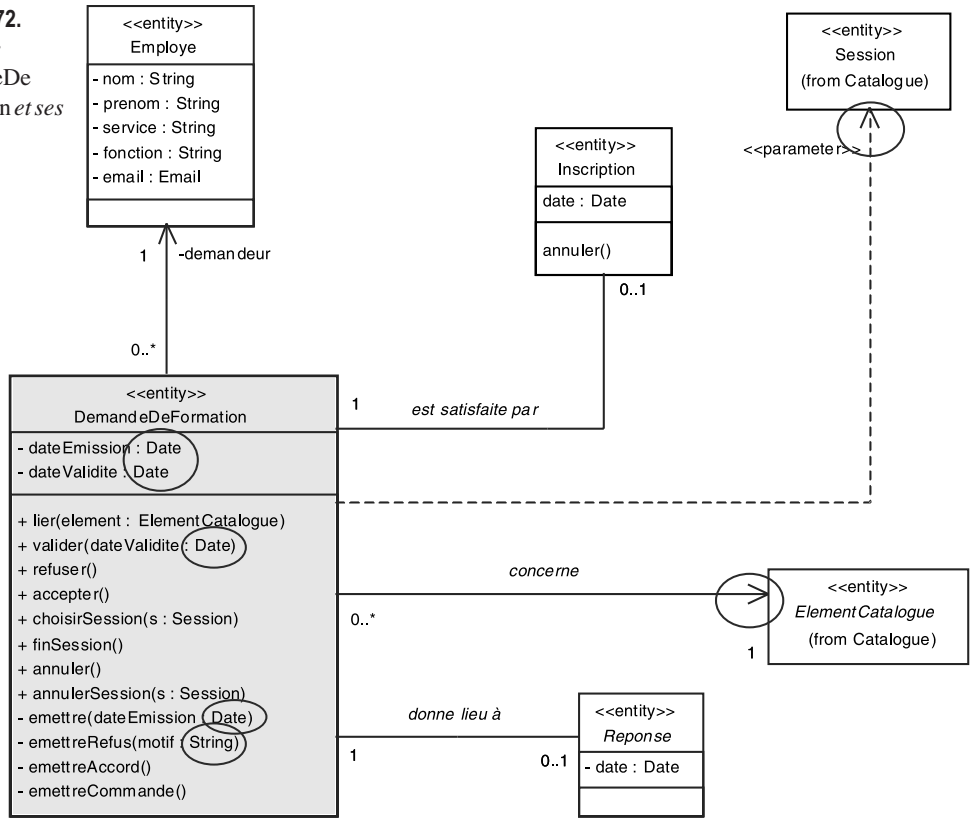


Figure 7-72. *La classe* DemandeDe Formation *et ses relations*

Le code Java correspondant est montré sur le schéma suivant.

On remarquera les directives d'importation ainsi que les quatre dernières méthodes qui permettent l'accès en lecture (*get*) et en écriture (*set*) aux attributs, pour respecter le principe d'encapsulation.

```
package demandes;

import java.util.*;
import catalogue.Session;
import catalogue.ElementCatalogue;

public class DemandeDeFormation
{
   private Date dateEmission;
   private Date dateValidite;
   private Employe demandeur;
   private Inscription inscription;
   private ElementCatalogue elementCatalogue;
   private Reponse reponse;

   public DemandeDeFormation()
   {
   }

   public void lier( ElementCatalogue element)
   {
   }

   public void valider(Date dateValidite)
   {
   }

   public void refuser()
   {
   }

   public void accepter()
   {
   }

   public void choisirSession(Session s)
   {
   }

   public void finSession()
   {
   }

   public void annuler()
   {
   }

   public void annulerSession(Session s)
   {
   }

   private void emettre(Date dateEmission)
   {
   }

   private void emettreRefus(String motif)
   {
   }

   private void emettreAccord()
   {
   }

   private void emettreCommande()
   {
   }

   public Date getDateEmission()
   {
      return dateEmission;
   }

   public void setDateEmission(Date de)
   {
      dateEmission = de;
   }

   public Date getDateValidite()
   {
      return dateValidite;
   }

   public void setDateValidite(Date dv)
   {
      dateValidite = dv;
   }
}
```

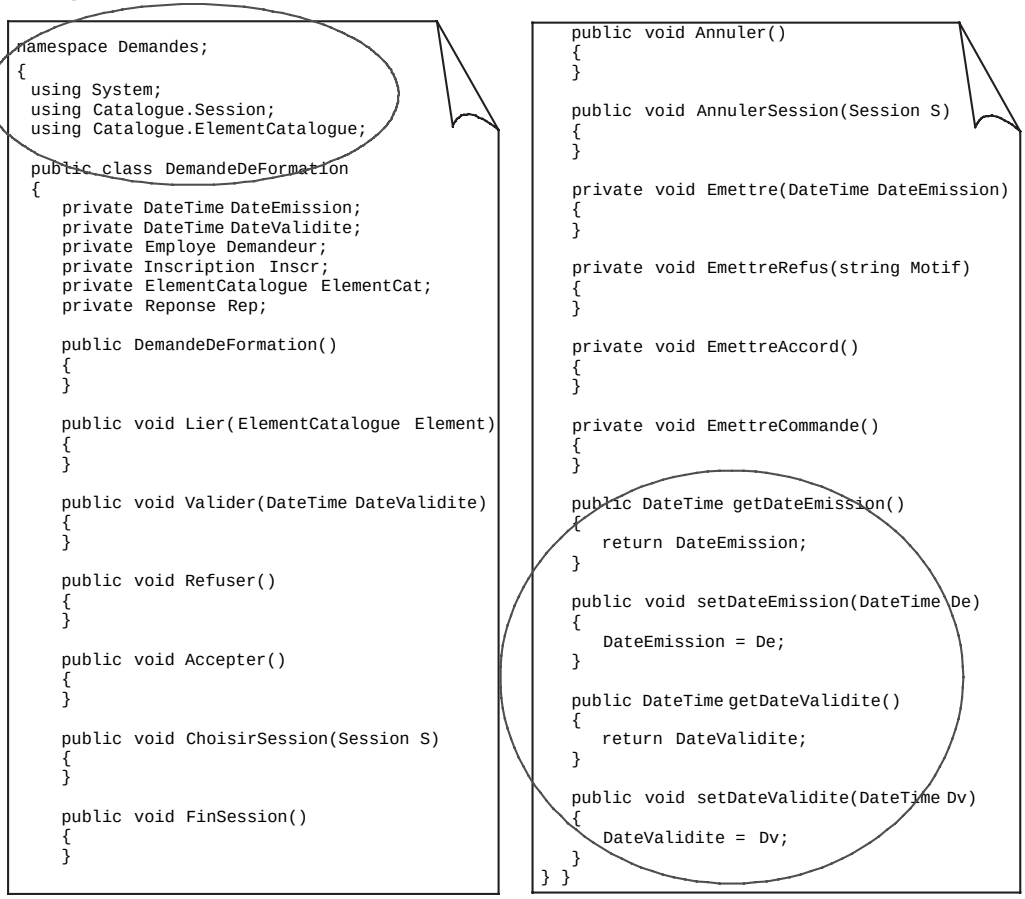
Figure 7-73.
Squelette de code Java de la classe DemandeDeFormation

EXERCICE 7-29.
Production d'un squelette de code C#

Pour bien montrer que notre conception permet ensuite facilement de dériver du code dans n'importe quel langage objet, faites le même exercice que précédemment, mais en utilisant le langage C# (de la plate-forme .NET) plutôt que Java.

Écrivez le squelette C# de la classe DEMANDEDEFORMATION.

solution



```
namespace Demandes;
{
 using System;
 using Catalogue.Session;
 using Catalogue.ElementCatalogue;

 public class DemandeDeFormation
 {
    private DateTime DateEmission;
    private DateTime DateValidite;
    private Employe Demandeur;
    private Inscription Inscr;
    private ElementCatalogue ElementCat;
    private Reponse Rep;

    public DemandeDeFormation()
    {
    }

    public void Lier(ElementCatalogue Element)
    {
    }

    public void Valider(DateTime DateValidite)
    {
    }

    public void Refuser()
    {
    }

    public void Accepter()
    {
    }

    public void ChoisirSession(Session S)
    {
    }

    public void FinSession()
    {
    }

    public void Annuler()
    {
    }

    public void AnnulerSession(Session S)
    {
    }

    private void Emettre(DateTime DateEmission)
    {
    }

    private void EmettreRefus(string Motif)
    {
    }

    private void EmettreAccord()
    {
    }

    private void EmettreCommande()
    {
    }

    public DateTime getDateEmission()
    {
       return DateEmission;
    }

    public void setDateEmission(DateTime De)
    {
       DateEmission = De;
    }

    public DateTime getDateValidite()
    {
       return DateValidite;
    }

    public void setDateValidite(DateTime Dv)
    {
       DateValidite = Dv;
    }
} }
```

Figure 7-74.
Squelette de code C# de la classe DemandeDeFormation

Notez qu'en C#, on utiliserait probablement la notion de propriété (*property*) pour coder les accesseurs de façon plus élégante, par exemple :

```
public DateTime DateValidite
  {
    get { return m_DateValidite; }
    set { m_DateValidite=value; }
  }
```

EXERCICE 7-30.
Production d'un squelette de code Java (bis)

À partir de la figure 7-62, proposez un squelette de code Java pour la classe *Formation*.

Écrivez le squelette Java de la classe FORMATION.

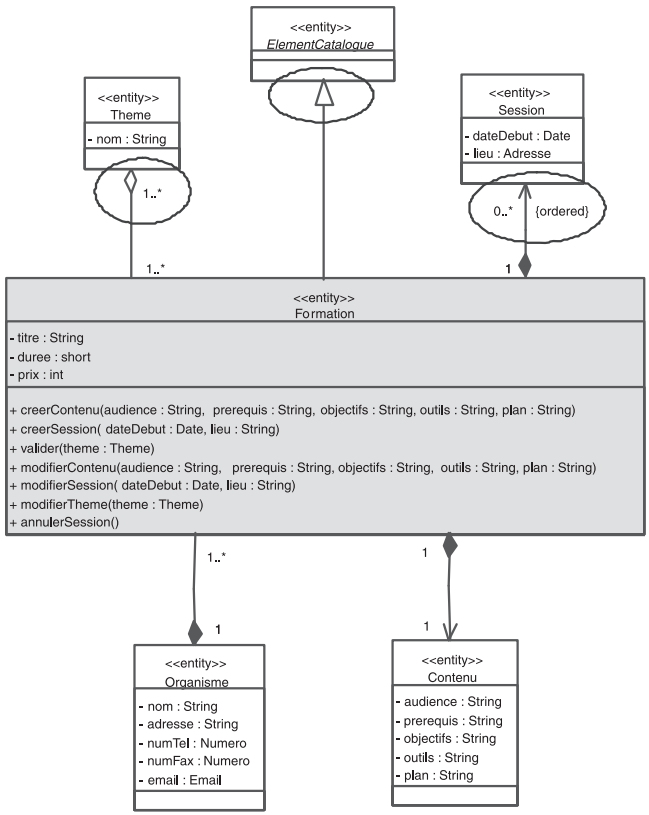
solution

Reprenons le schéma 7-62 en enlevant ce qui ne concerne pas la classe *Formation*.

Quelques difficultés supplémentaires apparaissent, par rapport à la question précédente :

- la relation de généralisation avec *ElementCatalogue*,
- les multiplicités « 1..* » avec *Theme* et « 0..* {ordered} » avec *Session*.

Figure 7-75.
La classe Formation
avec ses relations



Les règles précédentes ne suffisent plus. Nous avons vu un exemple de transformation d'une association navigable de multiplicité « 1 » (ou « 0..1 »), mais comment traduire les associations navigables de multiplicité « * » ?

À retenir

TRADUCTION EN JAVA DES ASSOCIATIONS AVEC MULTIPLICITÉ "*"

Le principe en est relativement simple : une multiplicité « * » va se traduire par un attribut de type collection de références d'objets au lieu d'une simple référence sur un objet.

La difficulté consiste à choisir la bonne collection parmi les très nombreuses classes de base que propose Java. Bien qu'il soit possible de créer des tableaux d'objets en Java, ce n'est pas forcément la bonne solution. En la matière, on préfère plutôt recourir à des collections, parmi lesquelles les plus utilisées sont *ArrayList* (anciennement *Vector*) et *HashMap* (anciennement *HashTable*). Utilisez *ArrayList* si vous devez respecter un ordre et récupérer les objets à partir d'un indice entier ; utilisez *HashMap* si vous souhaitez récupérer les objets à partir d'une clé arbitraire.

Attention, le JDK 1.5 introduit les collections typées appelées « *generics* »[d]. Nous allons pouvoir déclarer dorénavant des listes de sessions et des *maps* de thèmes, comme illustré à la question suivante.

Voici quelques exemples des solutions à retenir pour effectuer un choix pertinent :

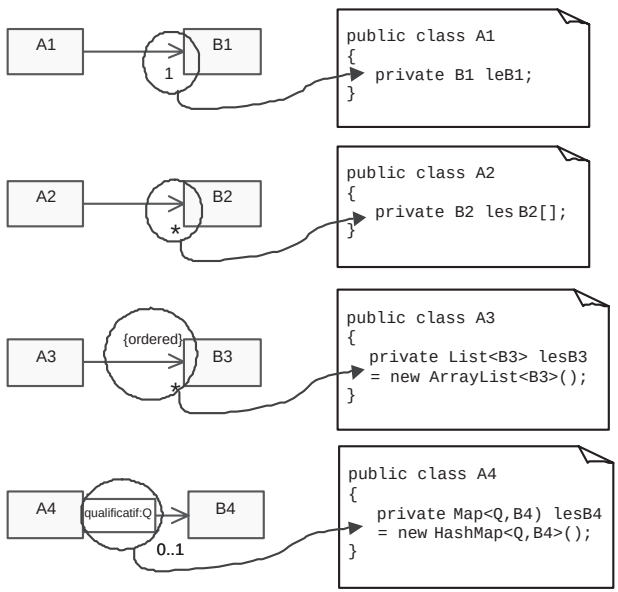


Figure 7-76.
Traductions possibles des associations en Java 1.5

d. Comme on pouvait s'y attendre, le langage C# ne sera pas en reste dans la version 2.0 du Framework .NET !

Pour la classe *Formation*, nous allons utiliser :

- une *ArrayList* pour l'association ordonnée avec la classe *Session* ;
- une *HashMap* pour l'association avec la classe *Theme*, plutôt qu'un simple tableau : nous nous servirons du nom de thème comme qualificatif.

Toutes ces explications nous conduisent au code suivant pour la classe *Formation*.

Figure 7-77.
Squelette de code Java 1.5 de la classe Formation

```
package catalogue;

import java.util.*;

public class Formation  extends ElementCatalogue

{
   private String titre;
   private short  duree;
   private  int prix;
   private List<Session> sessions = new ArrayList<Session>();
   private  Map<String,Theme> themes = new HashMap<String,Theme>();
   private Contenu  contenu;
   private Organisme  organisme;

   public Formation()
   {
   }

   public  void creerContenu(String audience, String   prerequis,
     String objectifs, String outils, String plan)
   {
   }

   public  void creerSession(Date  dateDebut, String lieu)
   {
   }

   public  void valider( Theme theme)
   {
   }

   public  void modifierContenu(String audience, String   prerequis,
     String objectifs, String   outils,String plan)
   {
   }

   public  void modifierSession(Date  dateDebut, String lieu)
   {
   }

   public  void modifier Theme( Theme  theme)
   {
   }

   public  void annulerSession()
   {
   }

   public String  getTitre() {return titre;}

   public  void setTitre(String t) {titre = t;}

   public short  getDuree() {return  duree;}

   public  void setDu ree(short d) { duree = d;}

   public  int getPrix() {return prix;}

   public  void setPrix( int p) {prix = p;}

}
```

EXERCICE 7-31.
Production d'un squelette de code C# (bis)

Écrivez le squelette C# de la classe FORMATION.

solution

À retenir

TRADUCTION EN C# DES ASSOCIATIONS AVEC MULTIPLICITÉ "*"

Le principe est identique à Java : une multiplicité « * » va se traduire par un attribut de type collection de références d'objets au lieu d'une simple référence sur un objet.

Comme en Java, on préfère recourir à des collections, parmi lesquelles les plus utilisées sont *ArrayList*, *SortedList* et *HashTable*. Utilisez *ArrayList* si vous devez respecter un ordre et récupérer les objets à partir d'un indice entier ; utilisez HashTable ou *SortedList* si vous souhaitez récupérer les objets à partir d'une clé arbitraire.

Voici quelques exemples des solutions à retenir pour effectuer un choix pertinent :

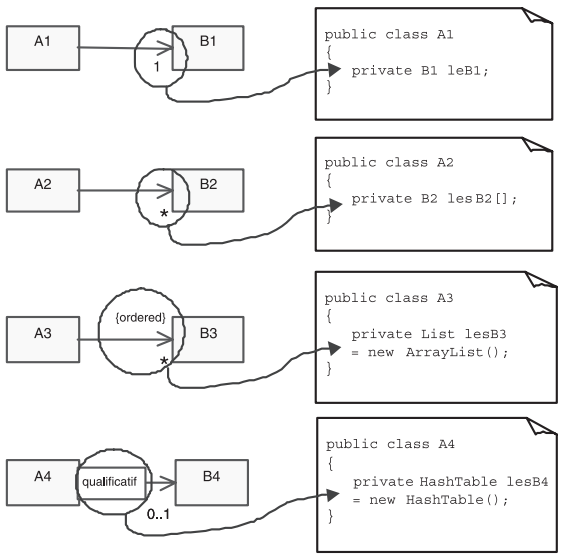


Figure 7-78. *Traductions possibles des associations en C#*

Il est à noter que les collections paramétrées sont annoncées également dans la prochaine version de C# (.NET 2.0). Le troisième cas donnerait par exemple une liste générique comme suit :

```
public class A3
{
private List<B3> lesB3 = new ArrayList<B3>() ;
}
```

Pour la classe *Formation*, nous allons utiliser :

- une *ArrayList* pour l'association ordonnée avec la classe *Session* ;
- une *HashTable* pour l'association avec la classe *Theme*, plutôt qu'un simple tableau : nous nous servirons du nom de thème comme qualificatif.

Toutes ces explications nous conduisent au code suivant pour la classe *Formation*.

```
namespace Catalogue;
{
  using System;

  public class Formation : ElementCatalogue

  {
     private string Titre;
     private short Duree;
     private int Prix;
     private List Sessions = new  ArrayList();
     private HashTable Themes = new HashTable();
     private Contenu  Cont;
     private Organisme  Org;

     public Formation()
     {
     }

     public void CreerContenu(string A udience, string Prerequis,
       string Objectifs, string Outils, string Plan)
     {
     }

     public void CreerSession(DateTime  DateDebut, string Lieu)
     {
     }

     public void Valider( Theme T)
     {
     }

     public void ModifierCont enu(string Audience, string Prerequis,
       string Objectifs, string  Outils, string Plan)
     {
     }

     public void ModifierSession(DateTime  DateDebut, string Lieu)
     {
     }

     public void ModifierTheme( Theme T)
     {
     }

     public void AnnulerSession()
     {
     }

     public string getTitre() {return Titre;}

     public void setTitre(string T) {Titre = T;}

     public short getDuree() {return Duree;}

     public void setDuree(short D) {Duree = D;}

     public int getPrix() {return Prix;}

     public void setPrix(int P) {Prix = P;}
} }
```

Figure 7-79.
Squelette de code C# de la classe Formation

Étape 16 – Déploiement de l'application

Nous allons finalement décrire l'implantation physique de notre application de gestion des demandes de formation grâce à un dernier type de diagramme proposé par UML : le diagramme de déploiement.

À retenir

DIAGRAMME DE DÉPLOIEMENT

Le diagramme de déploiement montre la configuration physique des différents matériels qui participent à l'exécution du système, ainsi que les artefacts qu'ils supportent.

Ce diagramme est constitué de « nœuds » connectés par des liens physiques. Les symboles des nœuds peuvent contenir des artéfacts (et non plus des composants comme en UML 1).

ARTÉFACT

Un artéfact modélise une entité physique comme un fichier. On le représente par un rectangle contenant le mot-clé « `artifact` ». On explicite la présence d'un artéfact sur un nœud de déploiement en imbriquant son symbole dans celui du nœud englobant.

Si un artéfact implémente un composant ou une classe, on dessine une flèche en pointillés avec le mot-clé « `manifest` » allant du symbole de l'artéfact au symbole du composant qu'il implémente[e].

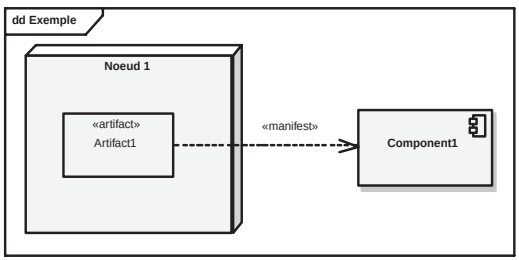


Figure 7-80.
Notation du diagramme de déploiement en UML 2

e. Le diagramme de déploiement a été notablement modifié avec la nouvelle version UML 2. Ceci vient principalement de la redéfinition du concept de composant, plus logique en UML 2 et donc moins physique. L'introduction du nouveau concept d'artéfact (*artifact*) permet de mieux séparer la manifestation physique du composant de sa représentation logique.

EXERCICE 7-32.
Diagramme de déploiement

Proposez un diagramme de déploiement réaliste pour les trois premières itérations du système de gestion des demandes de formation.

Élaborez un diagramme de déploiement pour les trois premières itérations.

solution

Chaque acteur a son propre poste client qui est un « PC » connecté au serveur intranet de l'entreprise, lequel est lui-même un PC serveur NT. Ce serveur intranet contient en particulier l'application d'authentification.

Chacun des deux acteurs a en outre sa propre interface homme-machine, matérialisée par une page *JSP*. Ces deux *JSP* utilisent un même service d'authentification général, contenu par le serveur intranet. Le catalogue est stocké dans une base de données spécifique, de même que les employés.

Le serveur métier héberge pour sa part les autres applications ainsi que les bases de données. Il s'agit là d'une machine Unix et ce pour des raisons historiques…

Le dessin, réalisé avec l'outil Enterprise Architect qui fournit ses propres icônes pour les différents types de nœud, est donné à titre d'exemple sur la figure suivante.

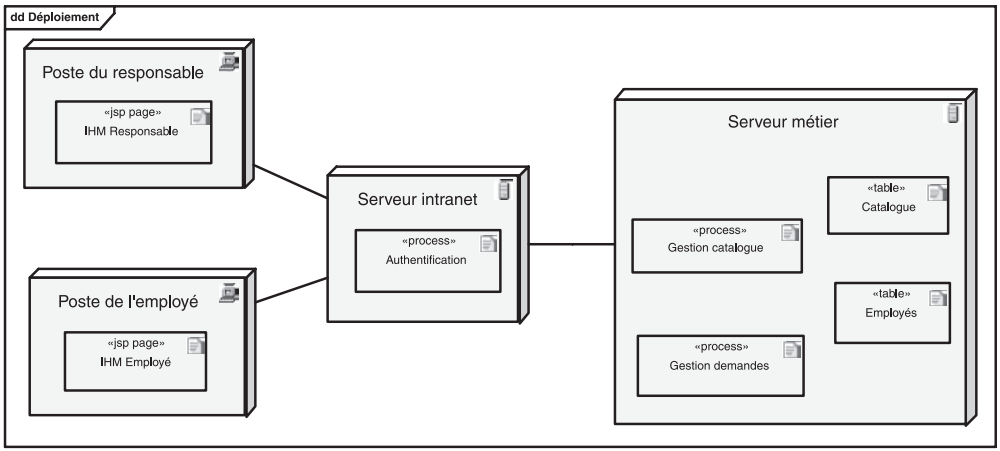


Figure 7-81.
Diagramme de déploiement correspondant aux trois premières itérations