Ingénieurs 2000
Génie Logiciel

Projet eMagine

Documentation développeur

FICHE D'APPROBATION

Version	Date	Rédacteurs
1.0	19/01/2006	A. OGIER

Chef de projet	Responsable Documentaire	Responsable Client
J.-B. RENAUDIN	L. BARBISAN	R. FORAX

Fiche de révision

Numéro de version actuel :1.0

Version	Date	Auteur	Libellé
0.1	18/01/06	A. OGIER	Création du document
0.2	18/01/06	A. OGIER	Reprise du CDCT
0.3	18/01/06	A. OGIER	Coupage des parties non importantes pour la doc dev.
0.4	19/01/06	A. OGIER	Relecture du document, correction des fautes
0.5	19/01/06	A. OGIER	Mise à jour de certaines informations non valides actuellement.

Sommaire

OBJET DU DOCUMENT

La documentation développeur d'eMagine est un document qui sert à la transmission des connaissances techniques nécessaires à la compréhension globale de l'application. Il explique à un développeur comment se compose l'application, quel est l'objectif de chaque élément.

Ce document présente les concepts utilisés, le design de l'application, les diagrammes génériques communs à de nombreux éléments, et enfin, présente chaque paquetage et leur utilité.

1. GLOSSAIRE

Le glossaire explique les termes qui seront utilisés tout au long du processus de mise en oeuvre de eMagine.

Acteur du logiciel : Individu géré par le système d'information (apprentis, tuteurs, acteur en entreprises).

Acteur en entreprise : Ces acteurs sont des individus appartenant à une entreprise. Les différents types d'acteurs en entreprise sont : décideur, gestionnaire, ingénieur, ingénieur suppléant et responsable taxe.

Apprenti : il s'agit d'un étudiant d'Ingénieurs 2000 suivant le cursus en alternance. Un apprenti possède un seul et unique tuteur ingénieur ainsi qu'un seul tuteur enseignant.

Basculement : Opération de passage d'un candidat en admis, d'un étudiant en année supérieure. Ce terme n'existera plus dans le logiciel eMagine.

Candidat : Étudiant souhaitant entrer dans une des filières Ingénieurs 2000, il est candidat dès l'envoi d'une demande de candidature jusqu'à son admission.

Courrier : Un courrier est un document papier ou électronique envoyé à un acteur du logiciel.

Courrier type : Courrier pré-rédigé ou seul certains emplacements vides du document sont à compléter (par exemple, nom du destinataire, adresse, etc...). Le logiciel utilisé est Microsoft Word.

CFA : Centre de Formation par Alternance.

CRM (Customer Relationship Management): A pour but de créer et entretenir une relation mutuellement bénéfique entre une entreprise et ses clients.

Diplômé : Apprenti ayant reçu son diplôme de fin d'étude.

Dossier d'inscription : Ce dossier est un document papier, renvoyé par le candidat. Il contient tous les renseignements correspondants à un candidat (état civil, scolarité...).

Dossier de l'apprenti : Ensemble des renseignements de l'apprenti formaté sous forme de fiche.

Évènement : Il s'agit d'une action effectuée par un utilisateur de l'application OU automatisée. Un évènement informe les utilisateurs d'un changement dans les données de l'application (ex: changement tuteur, etc) ou l'envoi d'un courrier. Tous les événements feront l'objet d'un historique.

Enseignant : Il s'agit d'un enseignant de l'école dispensant des cours dans le site académique de l'apprenti et chargé de suivre l'étudiant pendant toute sa période académique.

Entreprise : Il s'agit de l'entité entreprise (enseigne) mais elle fait intervenir plusieurs acteurs distincts du logiciel (décideur, gestionnaire, ingénieur, ingénieur suppléant et responsable taxe).

Entreprise liée : Celle-ci correspond à une entreprise employant au moins un apprenti Ingénieurs 2000.

Extraction : Il s'agit de l'export de données vers un format Excel ou CSV.

Fiche : Une fiche correspond aux détails des informations d'un acteur du logiciel ou d'un événement, par exemple l'état civil d'un apprenti ou le détail d'une absence.

Fonctionnalité : Une fonctionnalité est une action identifiée entre le système et les acteurs UML du système.

Identifiant : Chaîne désignant un utilisateur

Liste : Il s'agit d'une énumération d'acteurs du logiciel ou d'événements apparaissant dans le logiciel. Par exemple, une liste d'apprentis de deuxième année dans la filière Informatique et réseau.

Maîtrise d'oeuvre : équipe eMagine

Maîtrise d'ouvrage : clients

Mailing : Utilisation spécifique du courrier électronique qui permet la diffusion d'informations à un grand nombre d'utilisateurs possédant une adresse électronique.

Notification : Il s'agit d'une demande de modification. Celle-ci est effectuée par un utilisateur n'ayant pas les droits de modification (exemple : secrétaire).

Profil : Permet de définir une liste de modules accessibles (droit de modification, droit d'extraction, etc). Un utilisateur possède un seul et unique profil.

Publipostage : Génère une liste de courrier à partir d'un courrier type et d'une listes de données utilisées pour compléter les emplacements vides du courrier type.

Système d'Information (SI) : application eMagine.

Session : Accès pendant un certain temps de tout ou une partie du logiciel GALA. Cet accès se fait à partir d'un nom et d'un mot de passe.

Super-utilisateur : Utilisateur ayant accès à toutes les fonctionnalités du système.

Tuteur enseignant (TE) : Enseignant de l'école où étudie l'apprenti. Il a le devoir de suivre son apprenti pour toute son activité académique. Un tuteur peut avoir plusieurs tutés de sa filière. Le tuteur en entreprises est considéré comme un acteur en entreprise. Il hérite de ce dernier.

Tuteur ingénieur (TI) : Ingénieur de l'entreprise ou un employé ayant au moins 5 ans d'expérience dans l'entreprise. Il a le devoir de suivre son apprenti pour toute son activité professionnelle. Un tuteur peut avoir plusieurs tutés dans son entreprise.

UML : Unified Modeling Language, traduire par langage de modélisation unifié. Il s'agit d'un type de modélisation des données d'un Système d'Information. La méthode se compose de diagrammes utilisés pour exprimer des règles du système étudié.

Utilisateur : Il s'agit d'une personne interagissant avec le logiciel.

2. PRÉSENTATION DES CONCEPTS

La présentation des concepts permet d'éclaircir le document et d'éviter les contre sens.

2.1. LES ACTEURS UML

Las acteurs UML sont une part importante des concepts UML. Ils établissent des rôles dans l'applicatif. Un acteur ne représente pas forcément une seule personne (deux personnes peuvent se servir des mêmes fonctionnalités de l'application, elles ne forment alors qu'un seul acteur). De même une personnes peut être plusieurs acteurs.

Il existe quatre catégorie d'acteurs :

Acteur primaire : Regroupe les personnes qui utilisent les fonctions principales du système.

Acteur secondaire : Regroupe les personnes qui effectuent des tâches administratives ou de maintenances.

Matériel externe : Dispositif matériel nécessaire au système (ordinateur, etc...).

Autre système : Système externe avec lequel le système doit interagir.

Un acteur peut participer à des relations de généralisation. Les acteurs enfants seront alors capables de communiquer avec les cas d'utilisation des acteurs parents :

Ce schéma signifie que l'acteur 2 est capable de réaliser tout les cas d'utilisation de l'acteur 1.

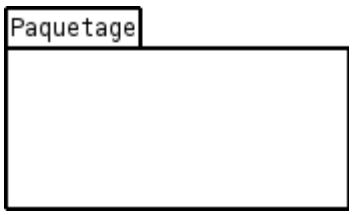
Diagramme des acteurs : Le diagramme des acteurs permet de montrer les relations des acteurs par rapport à leurs fonctionnalités (il y aura généralisation entre acteurs s'ils ont les même fonctionnalités).

Diagramme d'interaction : Il présente les interactions entre le système et les acteurs c'est à dire qu'il définit les fonctionnalités utilisées pour chaque acteurs.

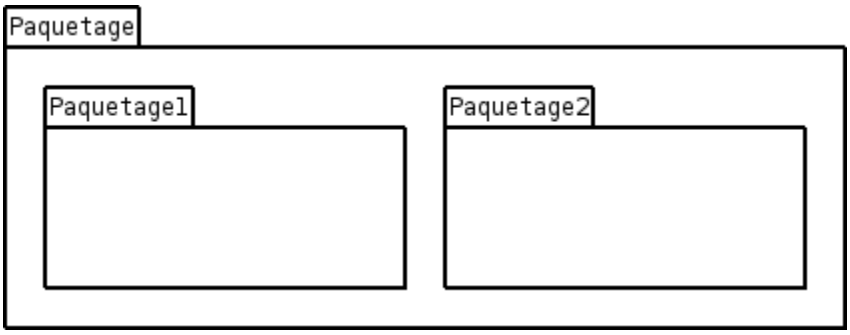
2.2. PAQUETAGES UML

a. Présentation

Les paquetages offrent un mécanisme général pour la partition des modèles et le regroupement des éléments de modélisation. Chaque paquetage est représenté graphiquement par un dossier :



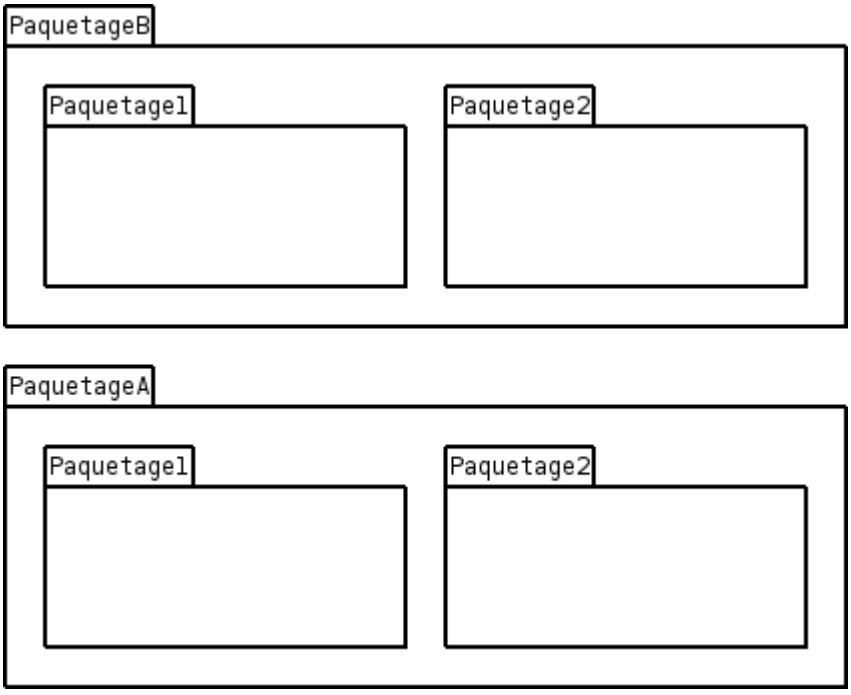
Les paquetages peuvent contenir ou référencer d'autres paquetages et d'autres éléments UML. La décomposition en paquetage n'est pas l'amorce d'une décomposition fonctionnelle. Ces regroupements sont faits selon des critères purement logiques. L'objectif d'une décomposition en paquetage est d'avoir une cohérence forte entre les éléments d'un paquetage.



Cela signifie que le paquetage1 est inclut dans le « Paquetage ».

b. Espace de nommage

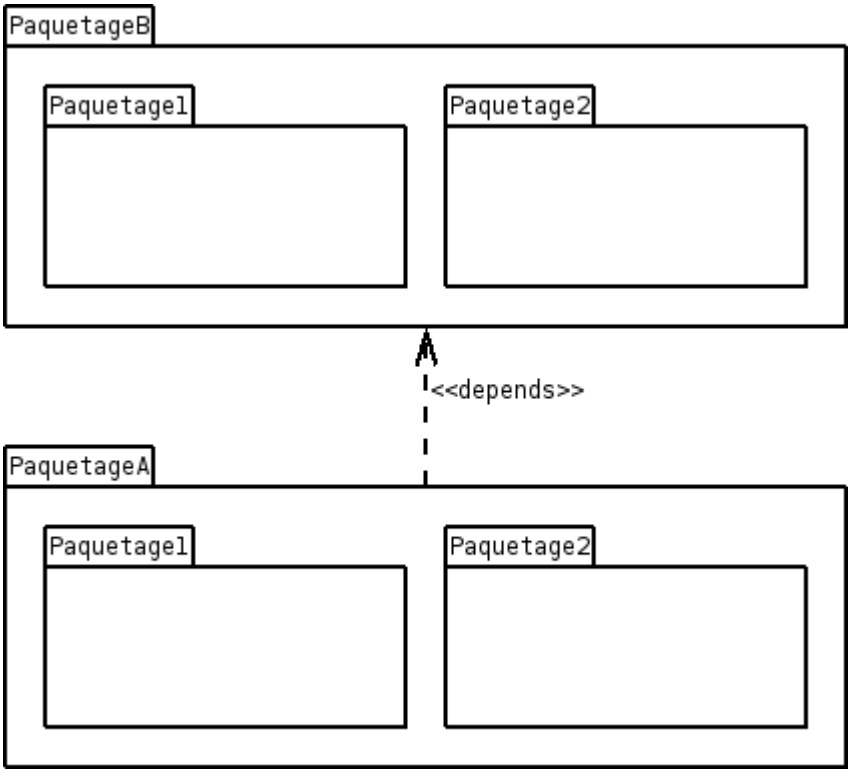
Chaque paquetage définit un espace de dommage. Cela signifie que tous les éléments contenus dans un paquetage se distinguent par leur appartenance au paquetage englobant.



paquetage1 dans paquetageB (paquetageB::paquetage1) n'est pas le même que paquetage1 dans paquetageA (paquetageB::paquetage1).

c. Dépendances entre paquetages

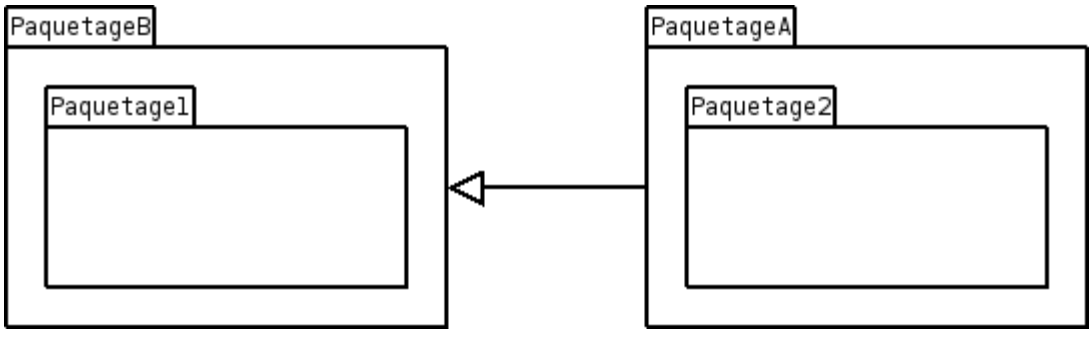
Par défaut les éléments contenus dans un paquetage emboîté voient les éléments contenus dans leur paquetage ou dans les paquetages englobants. Pour avoir accès aux éléments qui ne sont pas accessibles par défaut, il faut définir une relation de dépendance entre les paquetages :



Cela signifie que le paquetageB référence des éléments du paquetageA ou de paquetageA::paquetage1 ou de paquetageA::paquetage2

d. Généralisation

Un paquetage peut participer à une relation de généralisation. La généralisation est comparable à celle des classes. Le paquetage enfant décrit une spécialisation du paquetage parent.

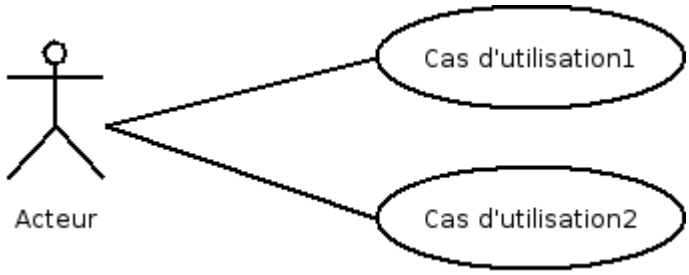


Cela signifie que la paquetageA contient tous les éléments du paquetageB.

2.3. Cas d'utilisation

a. Principe

Un cas d'utilisation est un classificateur qui modélise une fonctionnalité d'un système. L'instanciation d'un cas d'utilisation se traduit par l'échange de messages entre le système et ses acteurs.



Ce schéma explique que l'acteur utilise le système pour faire des choses.

Les cas d'utilisation se déterminent en observant et en précisant, acteur par acteur, les séquences d'interaction – les scénari – du point de vue de l'utilisateur. Ils décrivent en terme d'informations échangées et d'étapes dans la manière d'utiliser le système. Un cas d'utilisation regroupe une famille de scénari d'utilisation selon un critère fonctionnel. Les cas d'utilisations possèdent de nombreuses caractéristiques.

b. Priorité

Une priorité : il s'agit d'une valeur comprise entre 1 et 5, où 1 représente la priorité de développement la plus forte. Elle détermine les fonctionnalités prioritaires à traiter. Elle est calculée en fonction de la difficulté de développement et de l'importance de la fonctionnalité par rapport aux besoins du client.

L'échelle des priorités est attribuée de la façon suivante :

- **1 = Très importante** : Développement difficile d'une fonction indispensable,
- **2 = Importante** : Développement moyennement difficile d'une fonction indispensable,
- **3 = Élevée** : Développement difficile d'une fonction utile,
- **4 = Moyenne** : Développement moyennement difficile d'une fonction utile,
- **5 = Optionnelle** : Fonction non indispensable à l'application.

c. Pré-requis du système

Les pré-requis sont les conditions nécessaires au démarrage du cas d'utilisation.

d. Post-conditions

Les post-conditions représentent l'état du système après la réalisation du cas d'utilisation.

e. Besoins IHM

Les besoins IHM sont les besoins nécessaires pour l'interaction homme-machine.

f. Cas d'exceptions

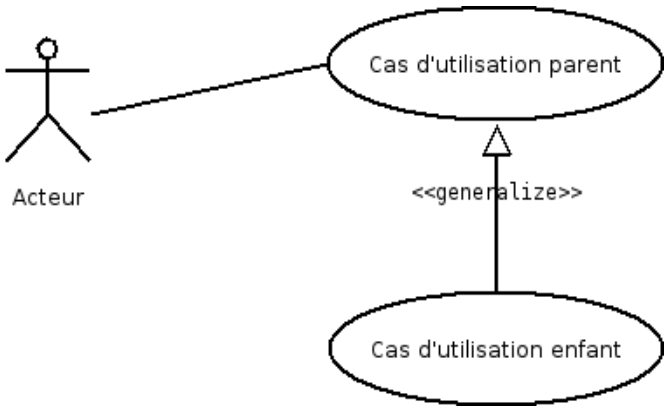
Les cas d'exceptions sont les conditions nécessaires au bon déroulement du cas d'utilisations.

g. Relation entre les cas d'utilisation

Il existe trois relations entre les cas d'utilisation.

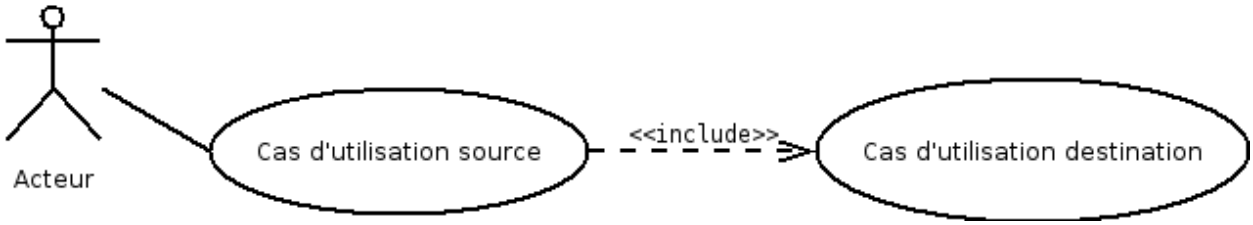
Généralisation

Dans une relation de généralisation entre deux cas d'utilisation, le cas d'utilisation enfant est une spécialisation du cas d'utilisation parent. Le cas d'utilisation parent peut être abstrait.



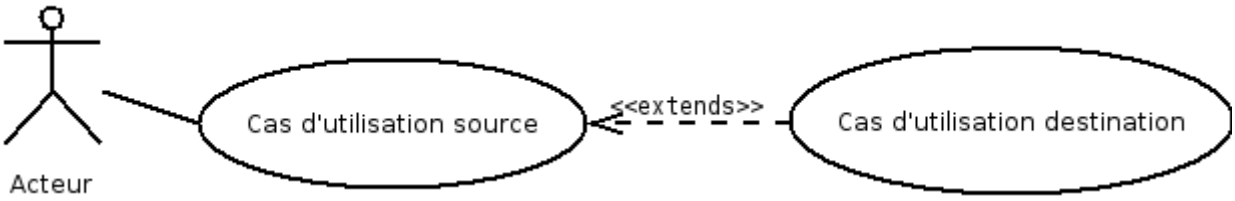
Inclusion

Dans une relation d'inclusion entre cas d'utilisation, une instance du cas d'utilisation source comprend également le comportement décrit par le cas d'utilisation destination. L'inclusion a un caractère obligatoire. La source spécifiant à quel endroit le cas d'utilisation cible doit être inclus.



➢ Extension

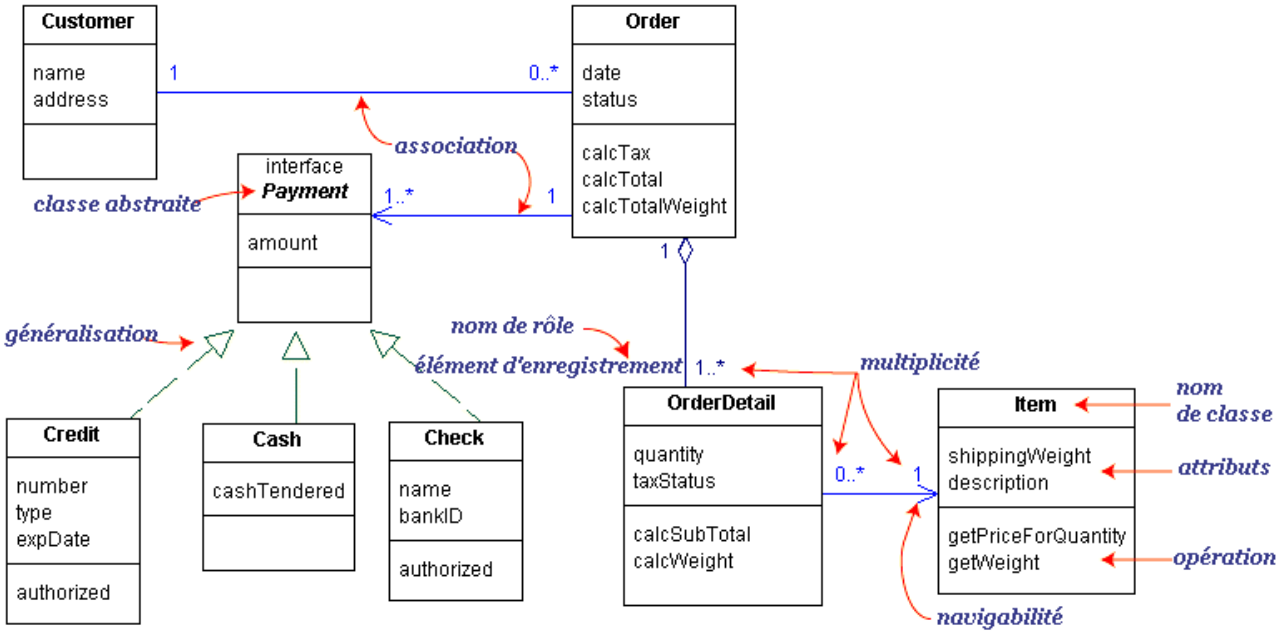
Dans une relation d'extension entre cas d'utilisation, le cas d'utilisation source ajoute son comportement au cas d'utilisation destination. L'extension peut être soumise à une condition. Cette relation permet de modéliser des variantes de comportement d'un cas d'utilisation.



2.4. Diagrammes de classes

Un diagramme de classes présente une vue globale d'un système en affichant ses classes et les relations entre elles. Les diagrammes de classes sont statiques : ils affichent ce qui interagit mais pas ce qui se passe pendant l'interaction.

Le diagramme de classes ci-après modélise la commande d'un client à partir d'un catalogue de détail. La classe centrale est la commande (Order). Customer, qui lui est associée, effectue l'achat et le paiement. Il existe trois types de paiements : en espèces, par chèque ou à crédit. La commande contient des OrderDetails (les lignes), chacun avec son article associé.



a. Eléments des diagrammes de classes

- **Classe**

  Une classe représente un concept dans le système en cours de modélisation. C'est une abstraction d'un ensemble d'objets ayant les mêmes attributs, opérations, relations et sémantiques.

- **Interface**

  Une interface est un spécificateur pour les opérations visibles de l'extérieur relatives à une classe, un composant ou un sous-système. Souvent, les interfaces ne spécifient qu'une partie limitée du comportement d'une classe réelle.

- **Opération**

  Une opération, désignée également sous le terme méthode, est un service qu'une instance de la classe peut être amené à exécuter.

- **Attribut**

  Un attribut est une propriété nommée d'une classe définissant une étendue de valeurs qu'un objet peut contenir.

- **Package**

  Les diagrammes de classes qui présentent des packages sont désignés également sous le nom diagrammes de package. En particulier, les diagrammes de package contiennent des icônes de package pour chaque sous-package physique sous le répertoire de projet.

- **Objet**

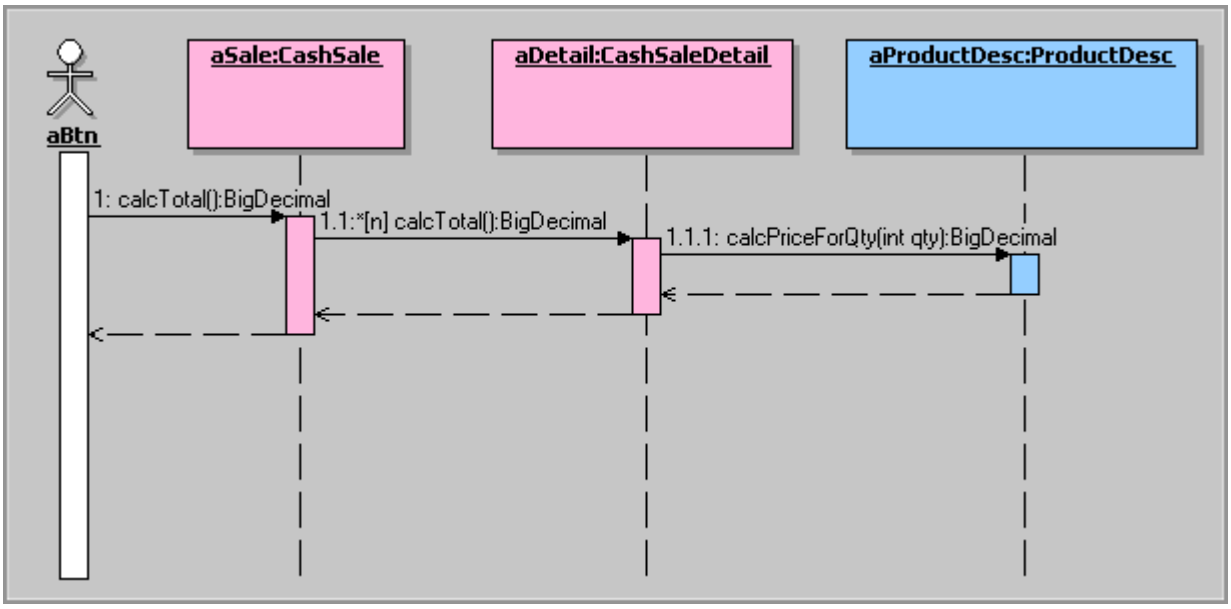
  Un objet représente une instance d'une classe.
  Pour définir les fonctionnalités d'un objet, vous pouvez insérer des **emplacements** dans l'élément d'objet, associer les emplacements aux attributs des classificateurs instanciés et définir les valeurs.

- **Liens**

  - Un lien d'association établit la connexion entre deux éléments de même type, entre les classes et les interfaces et vice-versa, ou entre deux objets.
  - Une relation d'agrégation signifie qu'un élément contient un ou plusieurs autres éléments. Vous pouvez dessiner un lien d'agrégation entre deux éléments de même type, entre les classes et les interfaces et vice-versa, ou entre deux objets.
  - Une relation de composition indique une forme intense d'agrégation. Vous pouvez dessiner un lien de composition entre deux éléments de même type, entre les classes et les interfaces et vice-versa, ou entre deux objets.
  - Un lien de dépendance peut être dessiné entre deux éléments quelconques.
  - Un lien de relation de généralisation/implémentation peut être dessiné entre deux éléments du même type, par exemple entre deux classes ou deux interfaces. Ce lien de relation peut

également être dessiné entre une classe et une interface, où la classe est le client et l'interface est le fournisseur.

2.5. Diagrammes de séquence



Les diagrammes de séquence, aussi appelés diagrammes d'interaction, peuvent être utilisés pour modéliser les aspects dynamiques d'un système ou sous-système. Les diagrammes de séquence représentent le comportement du système en tant que série d'interactions parmi un groupe d'objets. Chaque objet est représenté sous la forme d'une colonne dans le diagramme. Chaque interaction est représentée sous la forme d'une flèche entre deux colonnes.

a. Eléments des diagrammes de séquence

- **Acteurs**

Un acteur est utilisé pour échanger des informations avec un système. Les acteurs définissent un ensemble logique de rôles que les utilisateurs d'une entité peuvent jouer lors de l'interaction avec l'entité.

- **Objet**

Un objet représente une instance d'une classe.

- **Liens**

  - Lien de message : Un lien de message peut être dessiné entre la ligne de vie des objets sur le diagramme de séquence. Un objet demande l'exécution d'une opération à partir d'un autre objet en lui envoyant un message.
  - Lien de message avec délai de propagation : Un lien de message avec délai de propagation peut être dessiné entre la ligne de vie des objets sur le diagramme de séquence.

- Lien de message réflexif : Un lien de message réflexif peut être dessiné d'une ligne de vie d'un objet avec retour sur elle-même.
- Lien de retour : Un lien de retour peut être dessiné entre la ligne de vie des objets sur le diagramme de séquence.

3. DESCRIPTION DES ACTEURS

Les acteurs permettent de créer les Use case du chapitre 6.

3.1. ACTEURS

Administrateur : Il s'agit du technicien capable de gérer des profils et de les associer aux utilisateurs.

CFA pôle entreprise : Cet acteur est autorisé à modifier et à valider les demandes de modifications concernant les données du logiciel relatives à l'entreprise.

CFA pôle école : Cet acteur est autorisé à modifier et à valider les demandes de modifications des données du logiciel relatives aux enseignants.

CFA pôle apprenti : C'est à lui de gérer les apprentis lorsque cela concerne ses données personnelles ou l'ajout de ses tuteurs.

CFA pôle recrutement : Celui-ci sera à même de gérer tous le processus de candidature (entreprise ou apprenti).

Secrétaires : En plus des fonctionnalités minimales du logiciel, elles accèdent à la gestion des absences, ainsi qu'à la gestion des tuteurs enseignants.

Gestionnaire statistiques : sont assimilés à l'acteur « Gestionnaire statistiques », les responsables de filières ainsi que le responsable statistique du CFAI. Cet acteur est capable d'effectuer et de calculer des statistiques sur les différents acteurs et/ou les différents événements du logiciel.

Utilisateur : Ce sont des « utilisateurs basiques » . Il s'agit des utilisateurs du logiciel capables de consulter les informations du logiciel, de créer des mailings, d'effectuer des extractions et d'envoyer des demandes de modifications.

3.2. Diagramme des acteurs

Le diagrammes des acteurs montrent les liens d'héritage des acteurs.

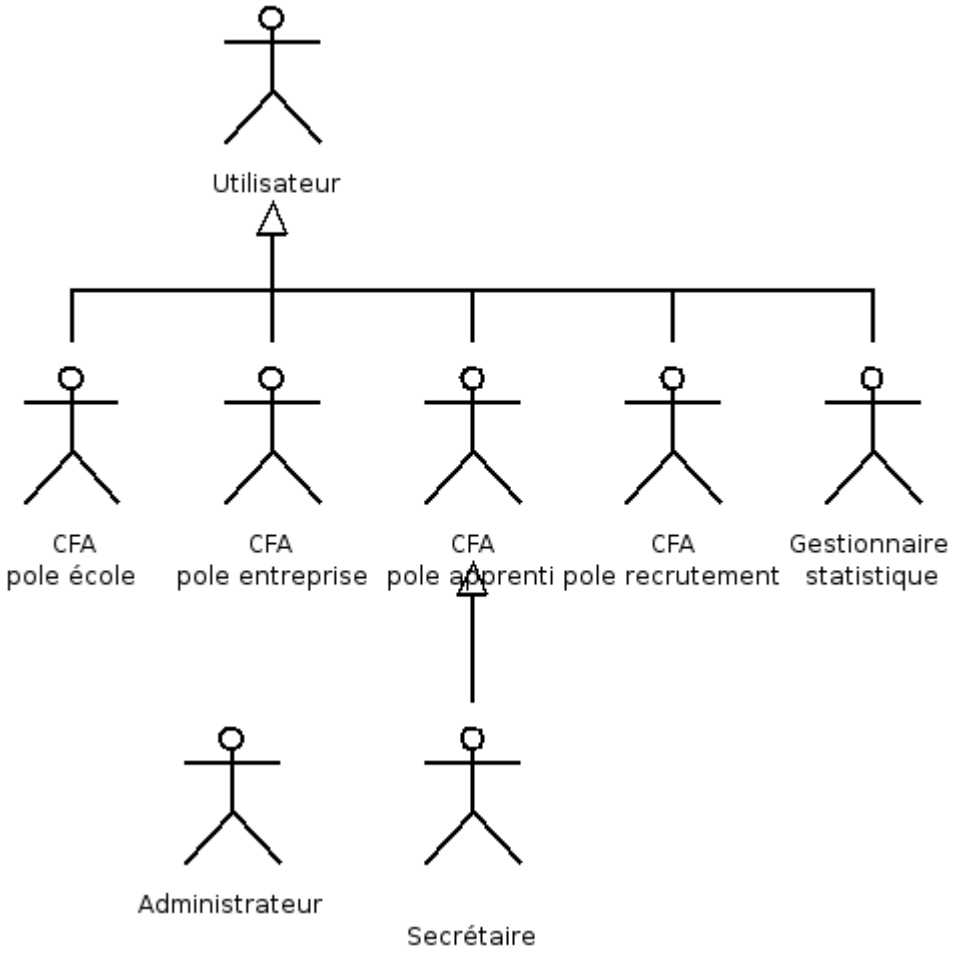


***Fig 3** : Diagramme des acteurs du nouveau système*

Tous les responsables, les secrétaires et le gestionnaire statistique bénéficient des mêmes fonctionnalités que l'utilisateur de base.

3.3. Diagramme des fonctionnalités

Le diagramme permet de répartir les fonctionnalités par acteur.

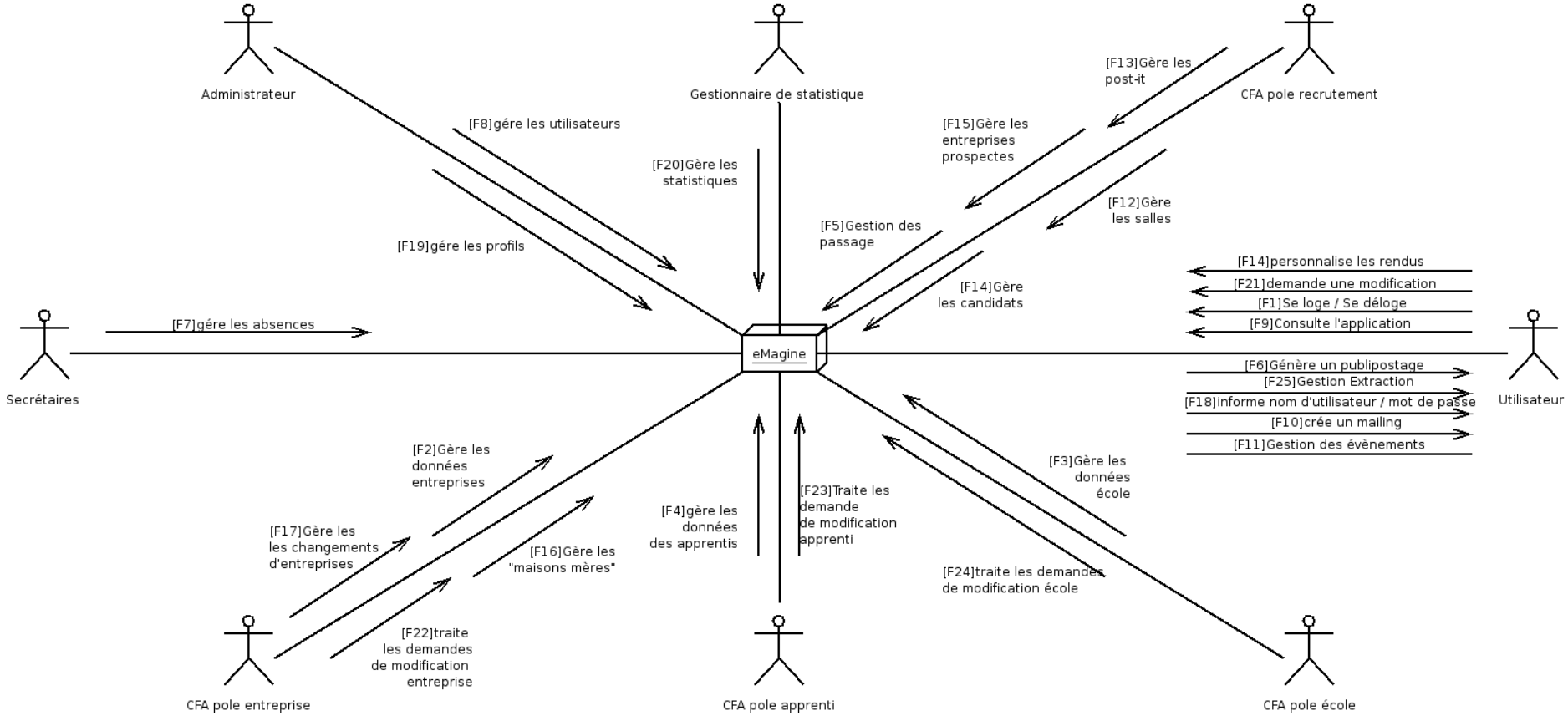


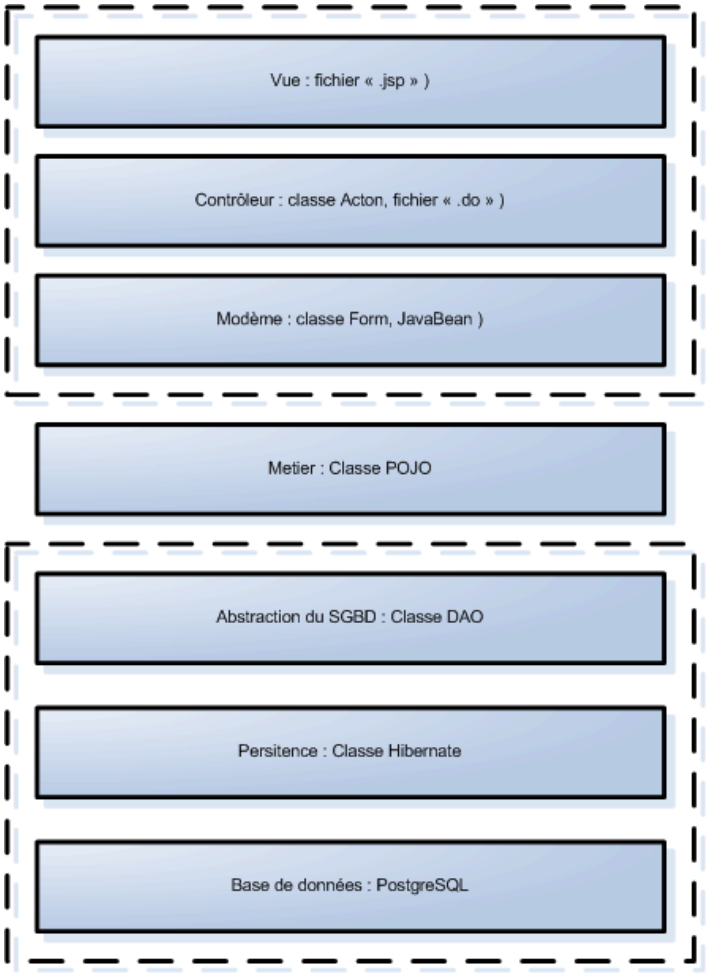
Fig 4 : Diagramme d'interaction du nouveau système

4. DESIGN

Ce chapitre explique les différents concepts et design patterns utilisés pour l'implémentation technique. L'application eMagine est décomposé en couche.

4.1. COUCHES APPLICATIVES

L'application peut être découpée en couche, les couches permettent d'atteindre un niveau d'abstraction optimal entre les différents composants de l'application. Cela permet l'implémentation des couches séparément.



Nous allons à présent analyser chaque couche de l'application. Les trois première couches sont le design pattern MVC (Modèle-Vue-Contrôleur, cf. Design Pattern MVC)

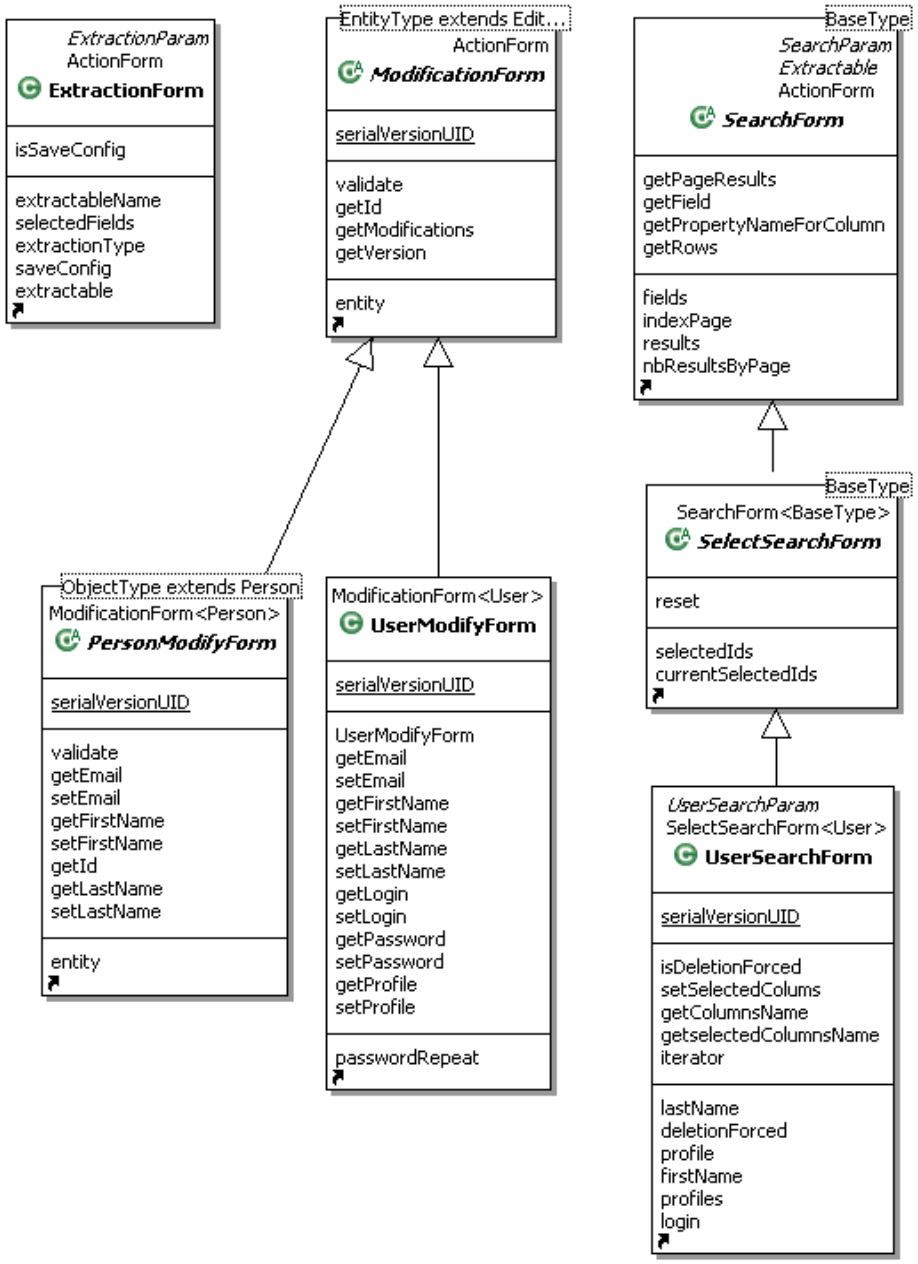
a. Vue

La vue est implémentée par Struts, c'est un framwork qui évite de mélanger les données avec l'affichage. En effet en html il arrive souvent que les données soit directement dans le code source de la page, struts permet d'éviter cela en utilisant le design pattern MVC. Les fichiers Jsp sont les vues, ce sont elle qui mettent en forme les donnée. Pour permettre une maintenance efficace des page html (.jsp) un template a été utilisé (un fihcier .css). Cela permet de centraliser les styles graphique des page html de façon a avoir toutes les page html avec la même mise en forme :

b. Modèle :

Le modèle est représenté par les classes Form, une classe forme ne contient exclusivement que des propriété. Les classes Form représente les données à mettre en forme et communique avec la page JSP.

Voici le diagramme général (extrait de leur contexte paquetage) des classes ActionForm pour l'applicatif eMagine :



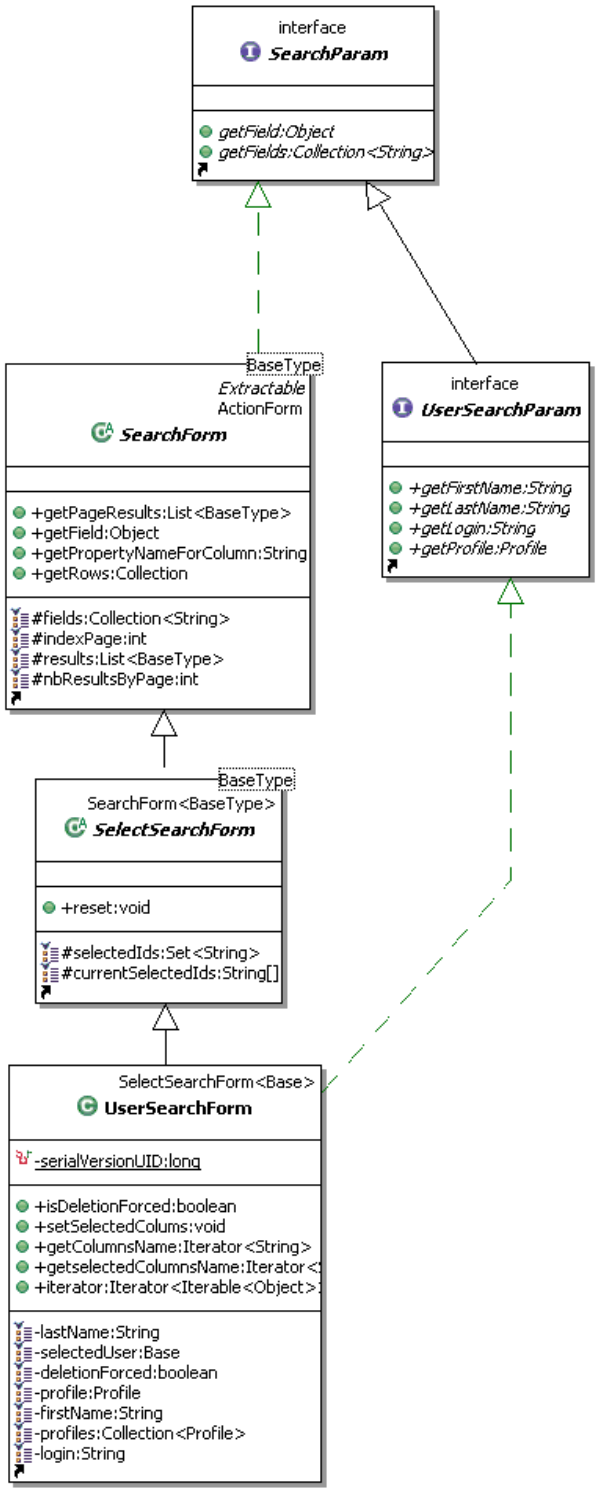
Les formulaires étant en générale de deux types :

- Recherche
- Modifications

Il a été décidé de créer deux formulaires de base : ModifyForm et SearchForm respectivement un formulaire de modification/création et un formulaire de recherche. Chaque nouveau formulaire (associé à une classe métier ex: User, Profil, etc...) hérite d'un de ces deux formulaire. Quant aux formulaires spécifiques ils héritent directement de ActionForm.

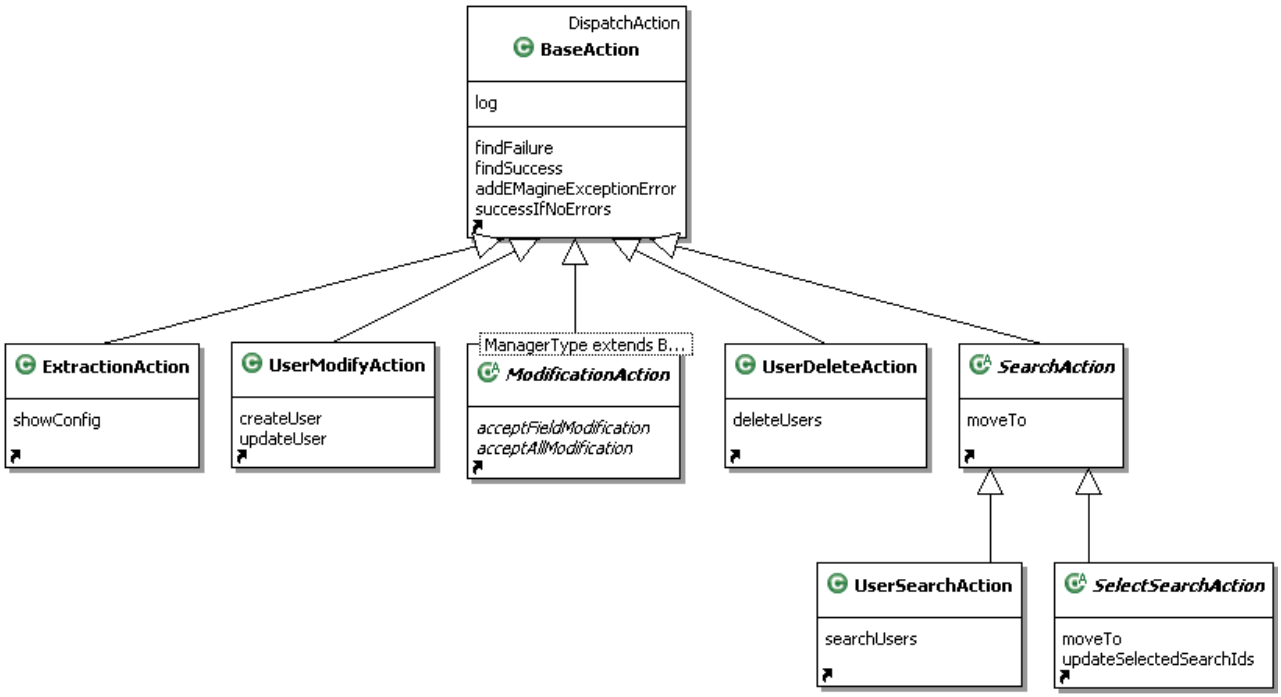
ModifyForm permet de récupérer tout les champs modifiés par l'utilisateur, ainsi que toutes les classes héritant de cette méthode. De même SearchForm permet d'afficher les résultats d'une recherche sous forme de page.

Les classes héritant de SearchForm implémentent aussi SearchParam, qui permet de spécifier les champs de recherche pour les recherche :



c. Contrôleur

Le contrôleur est représenté par la classe action, lors d'une validation d'un formulaire par l'utilisateur, une méthode est appelée sur une classe action. C'est lui qui fait le lien entre les données et l'affichage. Les classes implémentant les contrôleurs sont les classes héritant de Action. Voici le diagramme général (extrait de leur contexte paquetage) des différentes action de l'application. Les classe action peuvent être vue comme des Uses Cases. Chaque classe Action représenterais un UseCase, le schéma suivant représente les classes de base, toutes ne sont par représentées pour permettre une compréhension rapide :



Comme pour les classe Form, le principe d'héritage est conservé.

d. Manager

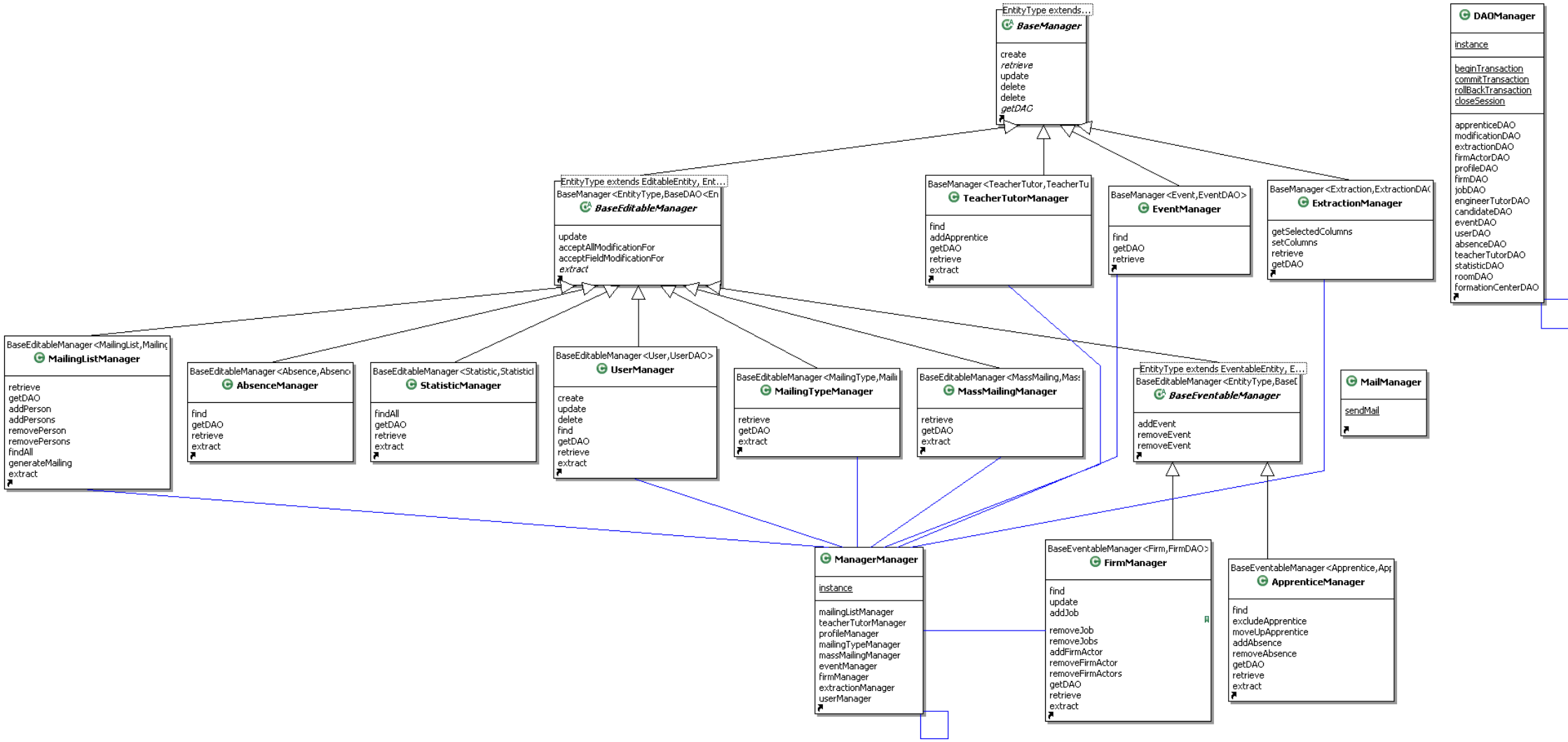
Les managers permettent de déplacer la responsabilité de gestion des classes métier (cf. Métier) vers un objet extérieur à la classe métier. De plus le manager permet de réaliser toutes les actions qui n'ont pas de rapport avec la base de données, et ne peuvent donc pas être dans les objets DAO (cf. DAO), et qui n'ont pas non plus de rapport avec la représentation graphique (classe Action). Ces actions sont donc mis dans les managers (par exemple l'envoie de mail).

Les managers héritent tous de BaseManager, le manager de base permet de faire les actions de base sur les entités :

- C : create
- R : retrieve
- U: update
- D: delete

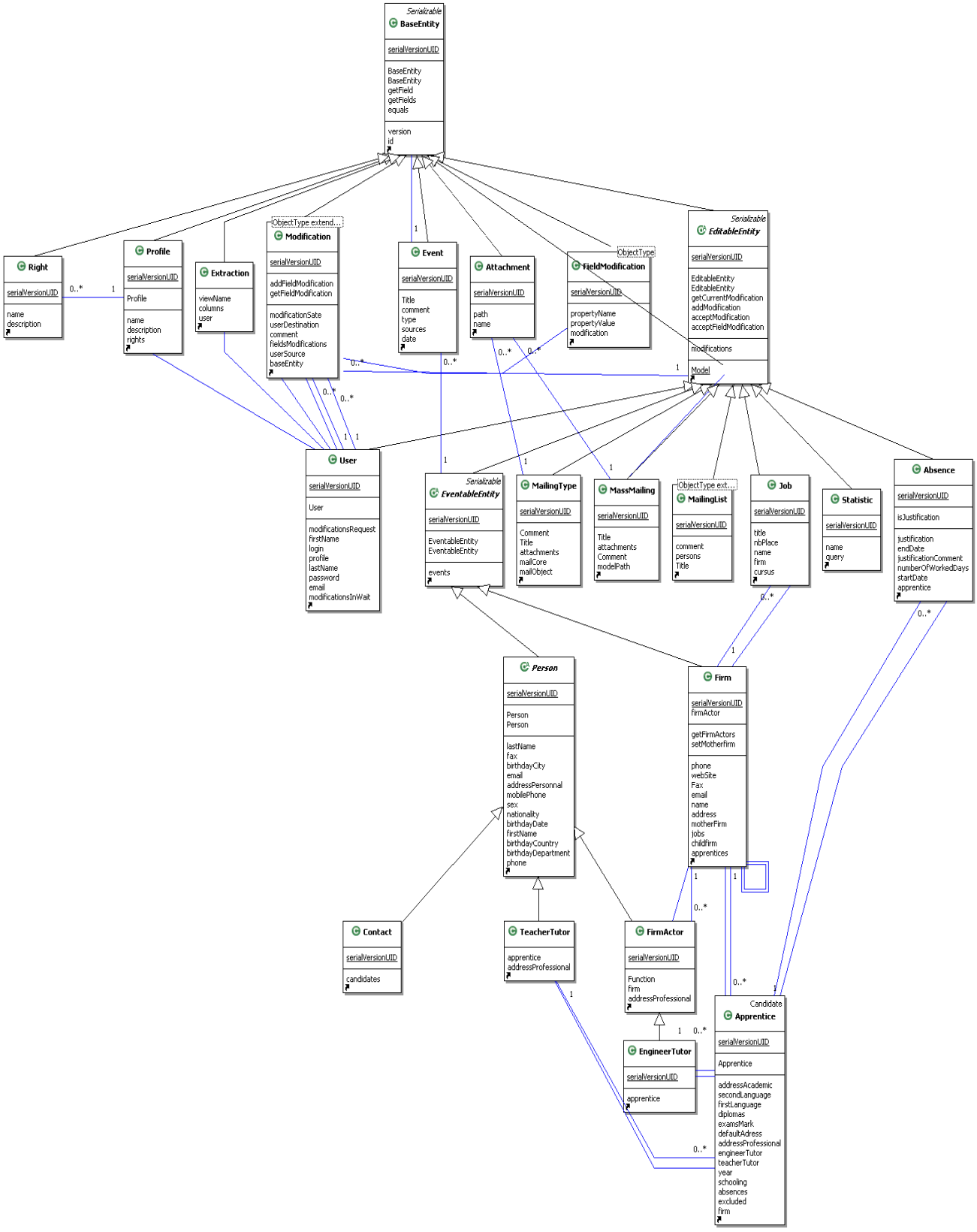
Ce design pattern est appelé CRUD (Create, Retrieve, Update, Delete, cf. Design CRUD)

Le baseManager est paramétré par le type de données qu'il gère, cela évite les cast intempestif et permet d'utiliser les méthodes de BaseManager quelque soit la classe qui hérite de BaseManager. La classe ManagerManager est un singleton (cf. Design Pattern Singleton) et instancie les managers.


EntityType extends...
BaseManager
create
retrieve
update
delete
delete
getDAO
DAOManager
instance
beginTransaction
commitTransaction
rollBackTransaction
closeSession
apprenticeDAO
modificationDAO
extractionDAO
firmActorDAO
profileDAO
firmDAO
jobDAO
engineerTutorDAO
candidateDAO
eventDAO
userDAO
absenceDAO
teacherTutorDAO
statisticDAO
roomDAO
formationCenterDAO
EntityType extends EditableEntity, Ent...
BaseManager <EntityType,BaseDAO <En
BaseEditableManager
update
acceptAllModificationFor
acceptFieldModificationFor
extract
BaseManager <TeacherTutor,TeacherTu
TeacherTutorManager
find
addApprentice
getDAO
retrieve
extract
BaseManager <Event,EventDAO>
EventManager
find
getDAO
retrieve
BaseManager <Extraction,ExtractionDA(
ExtractionManager
getSelectedColumns
setColumns
retrieve
getDAO
BaseEditableManager <MailingList,Mailing
MailingListManager
retrieve
getDAO
addPersons
addPersons
removePerson
removePersons
findAll
generateMailing
extract
BaseEditableManager <Absence,Absenc
AbsenceManager
find
getDAO
retrieve
extract
BaseEditableManager <Statistic,Statistic
StatisticManager
findAll
getDAO
retrieve
extract
BaseEditableManager <User,UserDAO>
UserManager
create
update
delete
find
getDAO
retrieve
extract
BaseEditableManager <MailingType,Maili
MailingTypeManager
retrieve
getDAO
extract
BaseEditableManager <MassMailing,Mass
MassMailingManager
retrieve
getDAO
extract
EntityType extends EventableEntity, E...
BaseEditableManager <EntityType,BaseE
BaseEventableManager
addEvent
removeEvent
removeEvent
MailManager
sendMail
ManagerManager
instance
mailingListManager
teacherTutorManager
profileManager
mailingTypeManager
massMailingManager
eventManager
firmManager
extractionManager
userManager
BaseEventableManager <Firm,FirmDAO>
FirmManager
find
update
addJob
removeJob
removeJobs
addFirmActor
removeFirmActor
removeFirmActors
getDAO
retrieve
extract
BaseEventableManager <Apprentice,Apr
ApprenticeManager
find
excludeApprentice
moveUpApprentice
addAbsence
removeAbsence
getDAO
retrieve
extract

e. Métier

Les classes métiers sont le coeur de l'application elle représentes les données pur de l'application elle sont gérées par hibernate qui s'occupe de les mapper en base de données.



Les classes métiers héritent toutes de BaseEntity. BaseEntity permet d'attribuer aux classes héritant un identifiant unique ainsi qu'un numéro de version pour le bloquage pessimiste. Ainsi tout les objets stockés en base sont manipulables via leur super-classe BaseEntity.

La classe abstraite EventBaseEntity permet d'ajouter la gestion des évènements au entité de base.

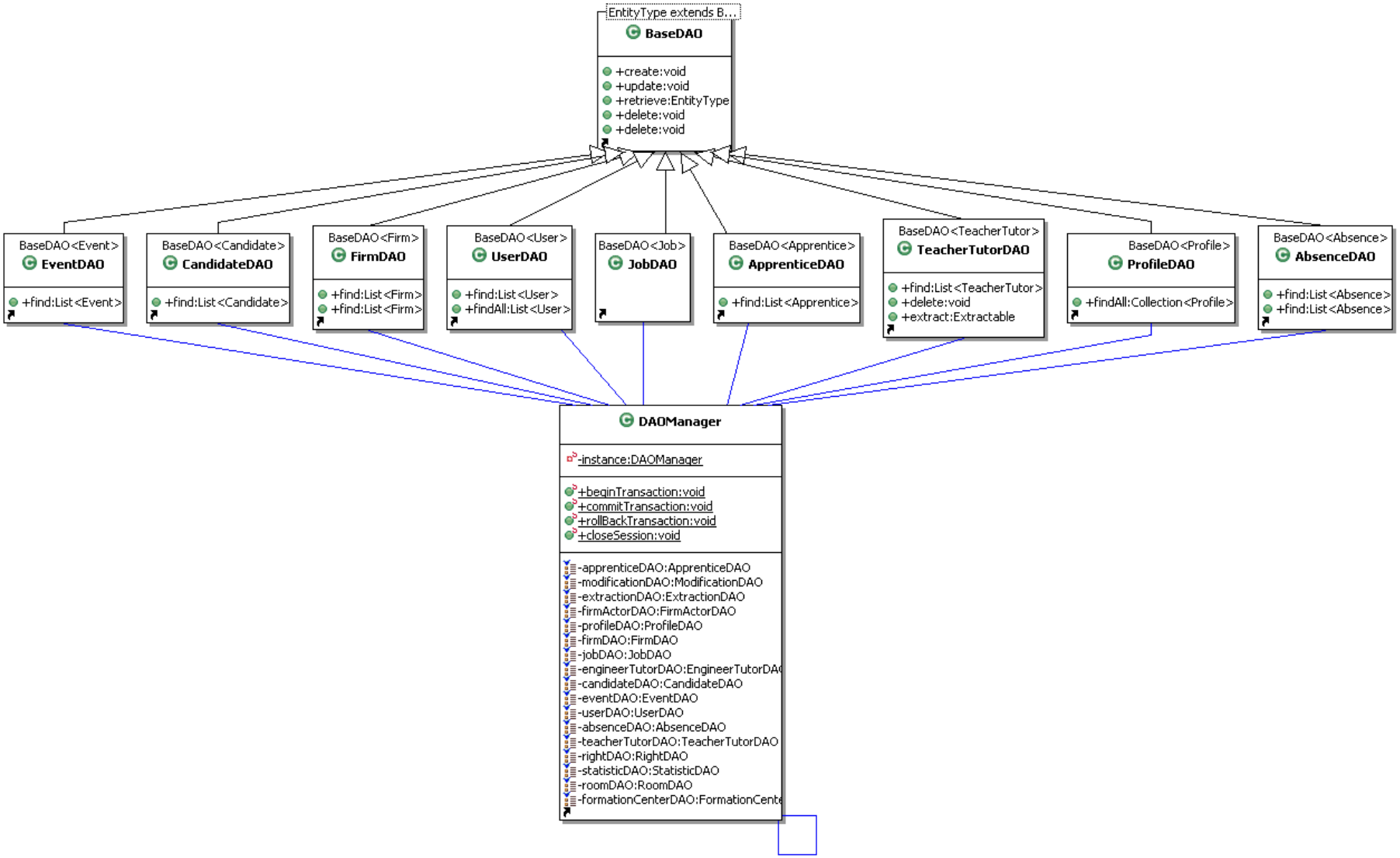
Enfin la classe EditableEntity permet d'ajouter la gestion des modifications au entity.

La plupart des entités hériteront donc de Editable entity, sauf les évènements proprement dits.

f. DAO

Les données doivent être indépendantes de la base de données, la couche d'abstraction DAO permet de faire cela, c'est un design pattern (cf. Design DAO). Les DAO permettent d'isoler les classes métiers de leur représentation dans la base de données. Ci-après les classes DAO, toutes n'ont pas été représentées pour ne pas complexifier le schéma, mais elles héritent toutes de BaseDAO, chaque classe est paramétrée avec sa classe métier (exemple: UserDAO<User>) :Les objets DAO héritent tous du DAO de base il permet de réaliser les 4 actions de base :

- C : create
- R : retrieve
- U: update
- D: delete


EntityType extends B...
BaseDAO
+create:void
+update:void
+retrieve:EntityType
+delete:void
+delete:void
BaseDAO<Event>
EventDAO
+find:List<Event>
BaseDAO<Candidate>
CandidateDAO
+find:List<Candidate>
BaseDAO<Firm>
FirmDAO
+find:List<Firm>
+find:List<Firm>
BaseDAO<User>
UserDAO
+find:List<User>
+findAll:List<User>
BaseDAO<Job>
JobDAO
BaseDAO<Apprentice>
ApprenticeDAO
+find:List<Apprentice>
BaseDAO<TeacherTutor>
TeacherTutorDAO
+find:List<TeacherTutor>
+delete:void
+extract:Extractable
BaseDAO<Profile>
ProfileDAO
+findAll:Collection<Profile>
BaseDAO<Absence>
AbsenceDAO
+find:List<Absence>
+find:List<Absence>
DAOManager
-instance:DAOManager
+beginTransaction:void
+commitTransaction:void
+rollBackTransaction:void
+closeSession:void
-apprenticeDAO:ApprenticeDAO
-modificationDAO:ModificationDAO
-extractionDAO:ExtractionDAO
-firmActorDAO:FirmActorDAO
-profileDAO:ProfileDAO
-firmDAO:FirmDAO
-jobDAO:JobDAO
-engineerTutorDAO:EngineerTutorDA
-candidateDAO:CandidateDAO
-eventDAO:EventDAO
-userDAO:UserDAO
-absenceDAO:AbsenceDAO
-teacherTutorDAO:TeacherTutorDAO
-rightDAO:RightDAO
-statisticDAO:StatisticDAO
-roomDAO:RoomDAO
-formationCenterDAO:FormationCente

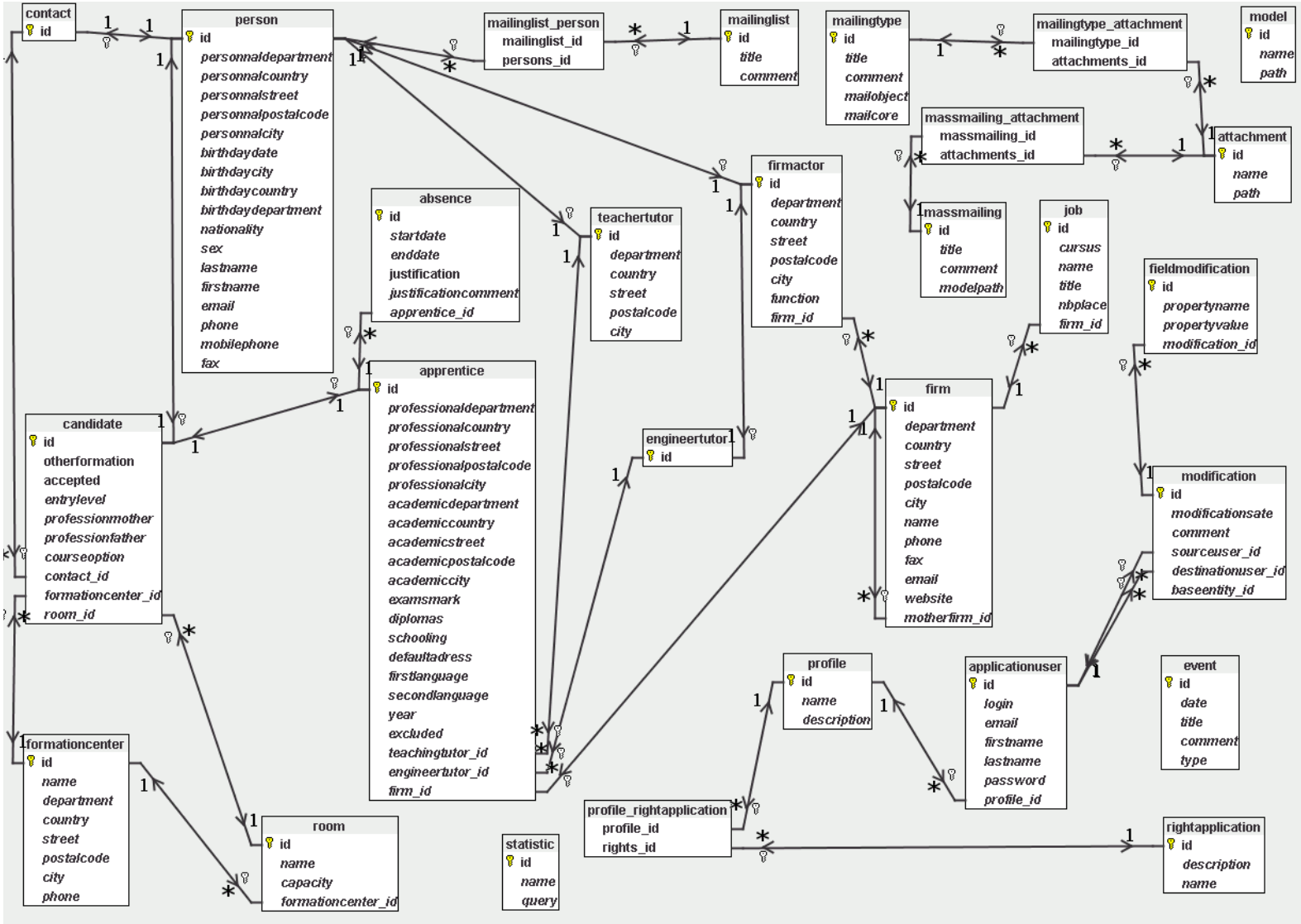
g. Base de données

La base de données est générée automatiquement avec Hibernate. Toutes les classes métier ont été annotées de manière à créer la base de données.

Il existe trois tables qui ne sont pas représentée sur le schéma. Elle représente respectivement les classes :

- BaseEntity
- EventableEntity
- EditableEntity

L'héritage étant représenté par une clef étrangère, chaque table a pour id l'id d'une des trois table ci-dessus. Si ces trois tables avaient été affichées cela aurait trop complexifié le schéma. Le schéma de base de données représente exactement le diagramme UML des classes, l'héritage étant représenté par une clef étrangère sur l'id de la classe parente.


contact
id
person
id
personnaldepartment
personnalcountry
personnalstreet
personnalpostalcode
personnalcity
birthdaydate
birthdaycity
birthdaycountry
birthdaydepartment
nationality
sex
lastname
firstname
email
phone
mobilephone
fax
mailinglist_person
mailinglist_id
persons_id
mailinglist
id
title
comment
mailingtype
id
title
comment
mailobject
mailcore
mailingtype_attachment
mailingtype_id
attachments_id
model
id
name
path
massmailing_attachment
massmailing_id
attachments_id
attachment
id
name
path
massmailing
id
title
comment
modelpath
absence
id
startdate
enddate
justification
justificationcomment
apprentice_id
teachertutor
id
department
country
street
postalcode
city
firmactor
id
department
country
street
postalcode
city
function
firm_id
job
id
cursus
name
title
nbplace
firm_id
fieldmodification
id
propertyname
propertyvalue
modification_id
apprentice
id
professionaldepartment
professionalcountry
professionalstreet
professionalpostalcode
professionalcity
academicdepartment
academiccountry
academicstreet
academicpostalcode
academiccity
examsmark
diplomas
schooling
defaultadress
firstlanguage
secondlanguage
year
excluded
teachingtutor_id
engineertutor_id
firm_id
candidate
id
otherformation
accepted
entrylevel
professionmother
professionfather
courseoption
contact_id
formationcenter_id
room_id
engineertutor
id
firm
id
department
country
street
postalcode
city
name
phone
fax
email
website
motherfirm_id
modification
id
modificationsate
comment
sourceuser_id
destinationuser_id
baseentity_id
profile
id
name
description
applicationuser
id
login
email
firstname
lastname
password
profile_id
event
id
date
title
comment
type
formationcenter
id
name
department
country
street
postalcode
city
phone
room
id
name
capacity
formationcenter_id
statistic
id
name
query
profile_rightapplication
profile_id
rights_id
rightapplication
id
description
name

4.2. Design Patterns utilisés

Les designs pattern permettent de proposer des solutions d'implémentation pour résoudre les problèmes couramment rencontré. Pour notre application de nombreux design pattern seront mis en oeuvre.

a. MVC

Le MVC est utilisé par Struts(cf Design MVC).

Problème

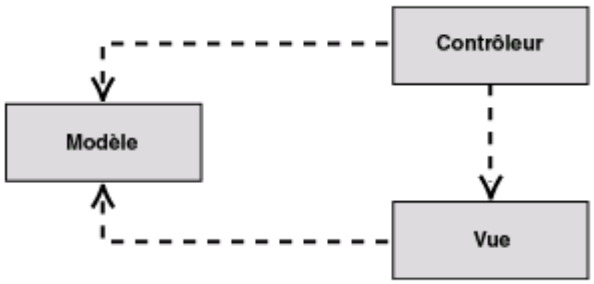
Comment modulariser l'interface utilisateur d'une application Web de sorte que chaque partie puisse être aisément modifiée individuellement ?

Solution

Le modèle *Model-View-Controller (MVC)* sépare la modélisation du domaine, la présentation et les actions reposant sur l'entrée utilisateur en trois classes distinctes :

- **Modèle**. Le modèle gère le comportement et les données du domaine d'application, répond aux demandes d'informations sur son état (souvent issues de la vue) ainsi qu'aux instructions de changement d'état (souvent issues du contrôleur).
- **Vue**. La vue gère l'affichage des informations.
- **Contrôleur**. Le contrôleur interprète les entrées clavier et souris de l'utilisateur et informe le modèle et/ou la vue des changements nécessaires.

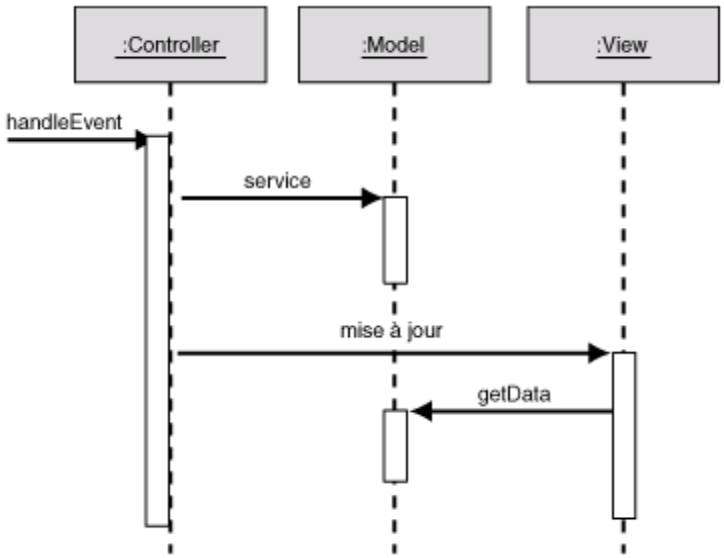
La figure 1 présente les relations structurelles entre les trois objets.



Il est important de noter que la vue et le contrôleur dépendent tous deux du modèle. En revanche, le modèle ne dépend ni de la vue ni du contrôleur. C'est l'un des principaux avantages de la séparation. Cette séparation des tâches permet de créer et de tester le modèle indépendamment de la représentation visuelle. La séparation entre la vue et le contrôleur est secondaire dans la plupart des applications clientes riches et, en effet, de nombreuses infrastructures d'interface utilisateur implémentent les rôles comme un seul objet. Dans les applications Web, à contrario, la séparation entre

la vue (le navigateur) et le contrôleur (les composants côté serveur gérant les requêtes HTTP) est clairement définie.

Le modèle *Model-View-Controller* est un modèle de conception fondamental en matière de séparation de la logique de l'interface utilisateur et de la logique métier. Malheureusement, la popularité de ce modèle a eu pour conséquence un certain nombre de descriptions erronées. En particulier, le terme « contrôleur » a été utilisé pour signifier différentes choses dans différents contextes. Fort heureusement, l'avènement des applications Web a permis de résoudre certaines ambiguïtés, tant la séparation entre la vue et le contrôleur est évidente :



b. CRUD

CRUD n'est pas réellement un design pattern mais plutôt une interface. Acronyme informatique anglais souvent utilisé lorsqu'on parle du langage de requête SQL. Ce sont les quatre opérations de base qui constituent tout ce qu'il est possible de faire sur Système de base de données

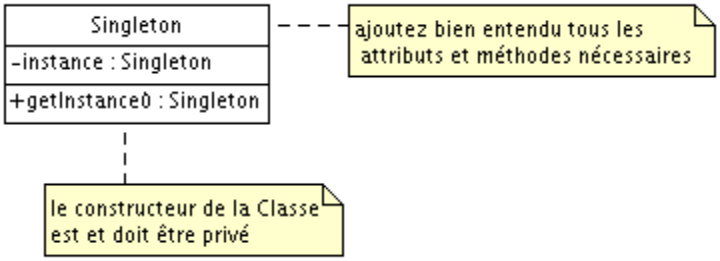
Les quatre lettres signifient **C**reate, **R**etrieve, **U**pdate et **D**elete.

c. Singleton

Le design pattern singleton est utilisé dans les manager pour obtenir toujours la même instance.

Le **singleton** est un modèle de conception *(design pattern)* dont le but est de restreindre l'instanciation d'une classe à un seul objet (ou bien quelques objets seulement). Il est utilisé lorsque l'on a besoin d'exactement un objet pour coordonner des opérations dans un système. Le modèle est parfois utilisé pour son efficience, lorsque le système est plus rapide ou occupe moins de mémoire avec peu d'objets qu'avec beaucoup d'objets similaires.

On implante le modèle **singleton** en écrivant une classe contenant une méthode qui crée une instance uniquement s'il n'en existe pas encore. Sinon elle renvoie une référence vers l'objet qui existe

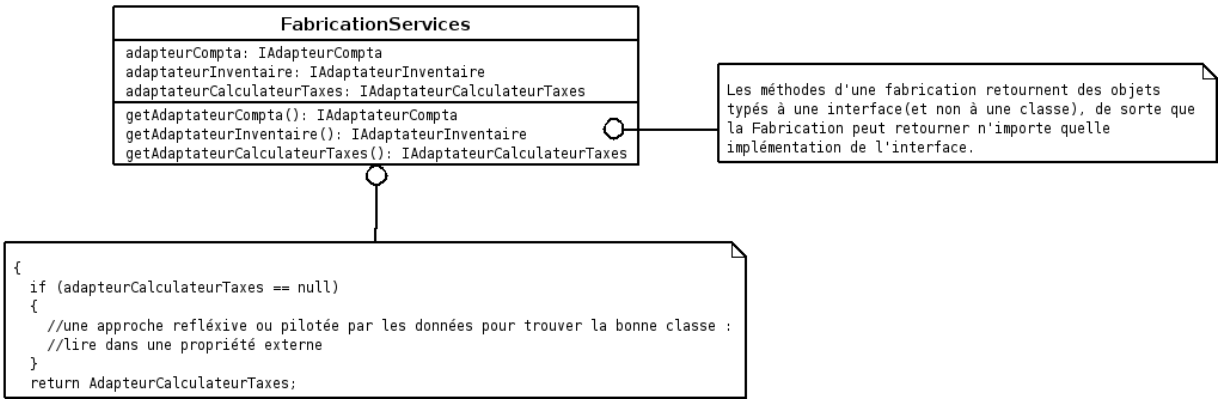


déjà. Dans beaucoup de langages à objets, il faudra veiller à ce que le constructeur de la classe soit *privé* ou bien *protégé*, afin de s'assurer que la classe ne peut être instanciée autrement que par la méthode de création contrôlée.

Le modèle **singleton** doit être implanté avec précaution dans les applications multi-thread Si deux threads exécutent *en même temps* la méthode de création alors que l'objet unique n'existe pas encore, il faut absolument s'assurer qu'un seul créera l'objet, et que l'autre obtiendra une référence vers ce nouvel objet

d. Factory

La factory est utilisé dans la classe SecurityProxyFactory (cf. Security) pour permettre de récupérer un manager spécifique.



- **Contexte/Problème :**

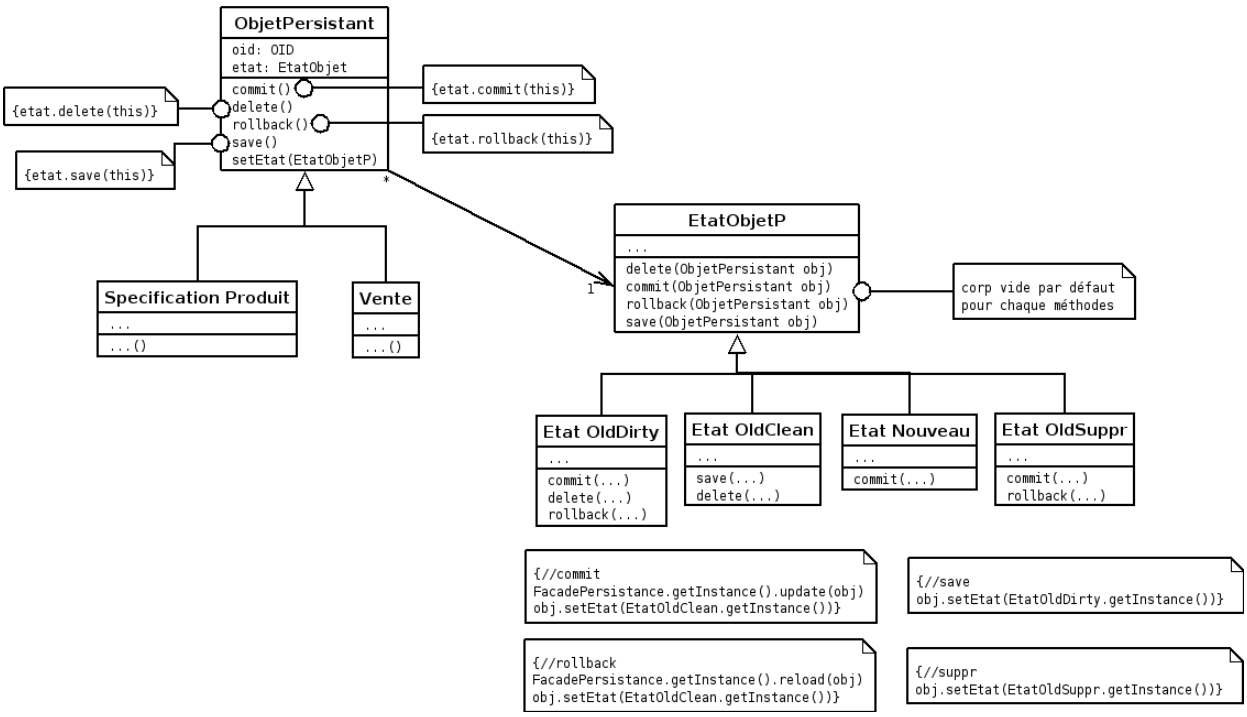
  Qui doit être responsable de la création des objets lorsque ceux-ci correspondent à des considérations spéciales, telles qu'une logique de création complexe, une volonté de séparer les responsabilités de création pour une meilleure cohésion, etc. ?

- **Solution :**

  Créez un objet Fabrication pure nommé Fabrication pour gérer la création.

e. Commande

Struts utilise le design pattern commande pour executer les actions. Le design pattern commande encapsule une requête comme un objet, ce qui permet de faire un paramétrage des clients avec différentes requêtes, files d'attente, ou historiques de requêtes, et d'assurer le traitement des opérations réversibles.



- **Contexte/Problème :**

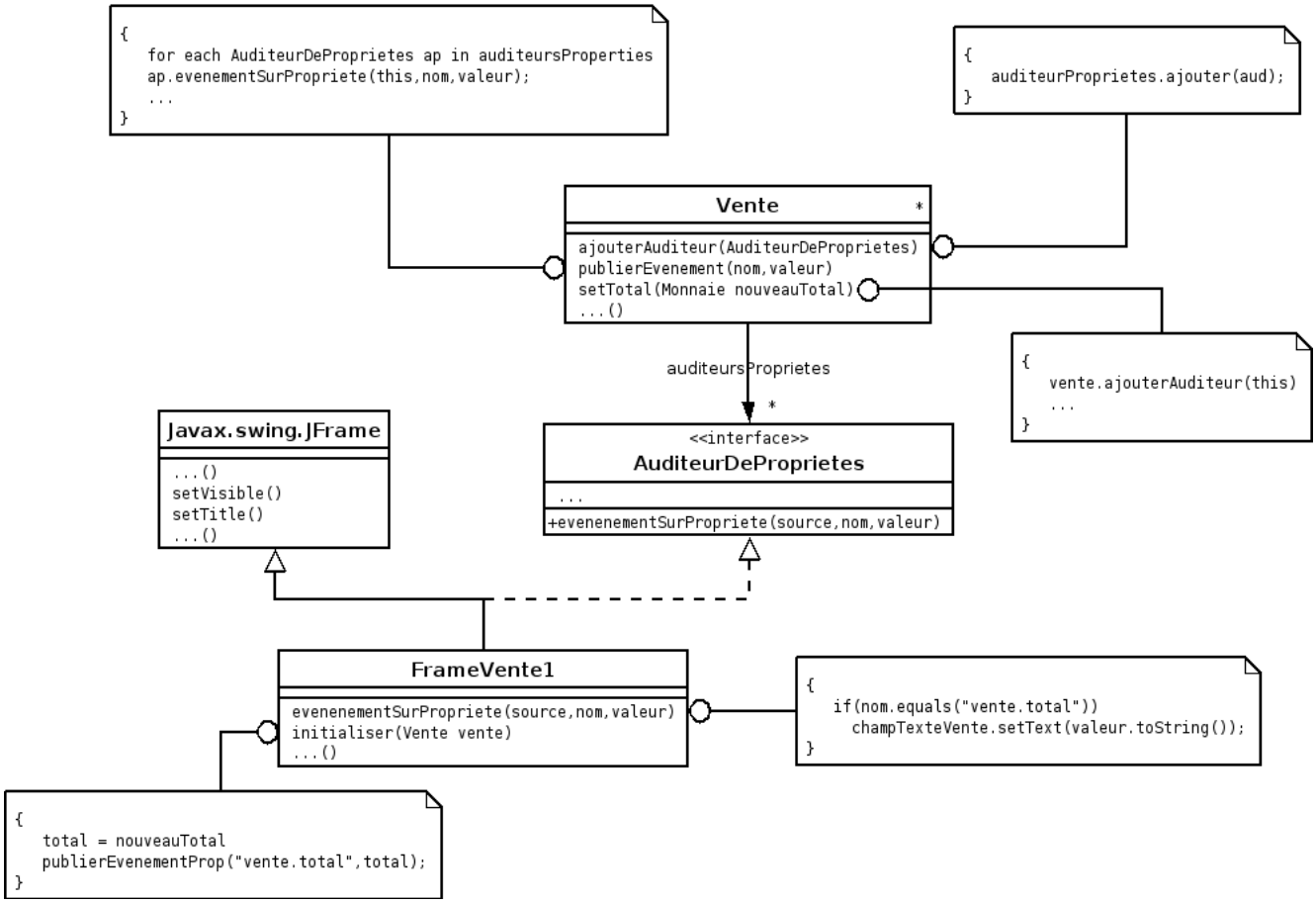
Comment gérer les requêtes qui nécessitent des fonctions telles que le tri (attribuion de priorités), la mise en file d'attente, le report, la journalisation ou l'annulaion ?

- **Solution :**

Faire de chaque tâche une classe qui implémente une interface commune.

f. Observer

L'observer est utilisé pour surveiller l'état des session utilisateur, connecté, déconnecté.



➢ **Contexte/Problème :**

Différents types d'objets souscripteurs sont concernés par les changements d'état et les événements d'un objet diffuseur (publisher) et veulent réagir à leur façon lorsque le diffuseur génère un événement. De plus, le diffuseur veut maintenir un

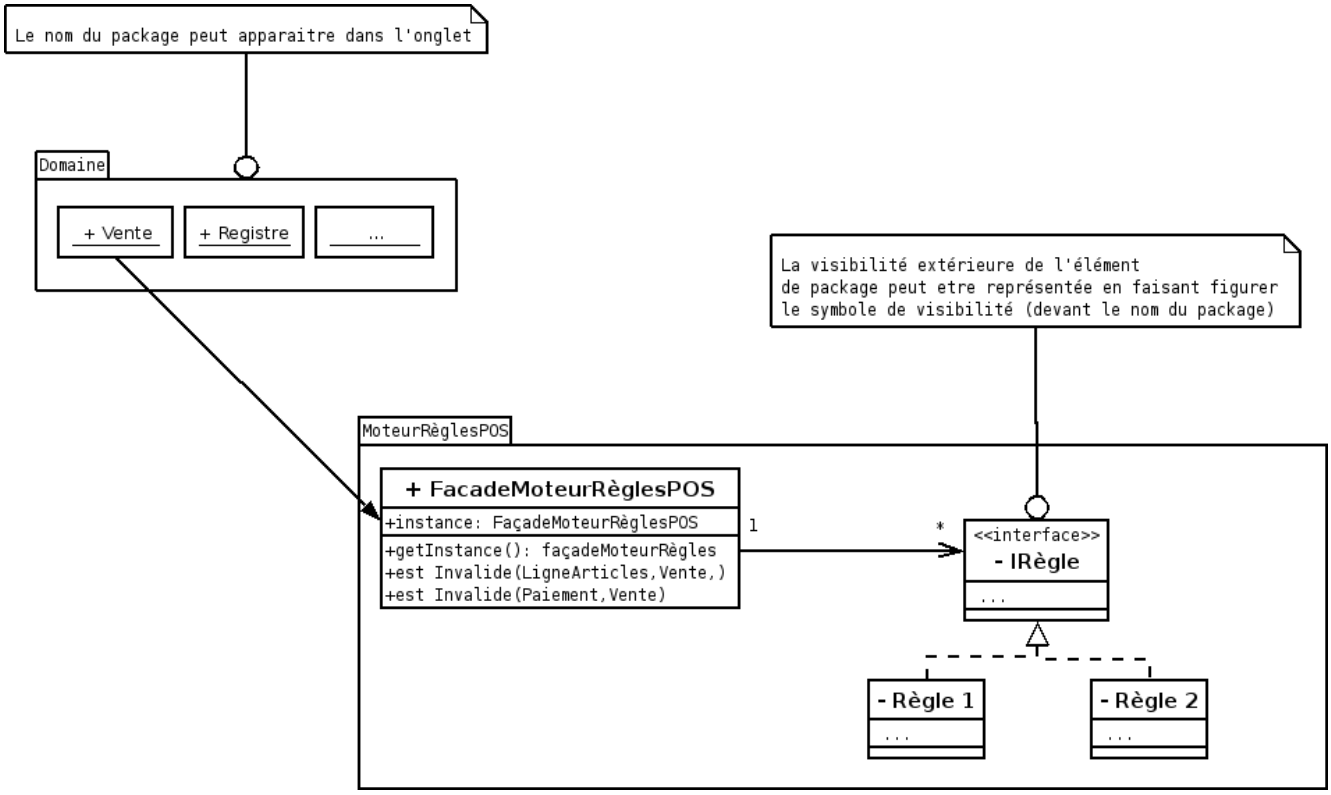
faible couplage avec les souscripteurs. Qne faire ?

➢ **Solution :**

Définir une interface "souscripteur" ou "auditeur". Les souscripteurs implémentent cette interface. Le diffuseur peut enregistrer dynamiquement les souscripteurs intéressés par un événement et le leur signaler.

g. Façade

Les Managers sont des façades, ils permettent de n'avoir qu'une seule et unique classe pour gérer toutes les actions associées à un type de classe métier.



➢ **Contexte/Problème :**

Il faut une interface unifiée et commune à un ensemble disparate d'implémentations ou d'interfaces (comme dans un sous-système). Il peut y avoir des couplages

non souhaitables avec beaucoup d'éléments du sous-système, et l'implémentain de ce sous-système peut varier. Que faire ?

➢ **Solution :**

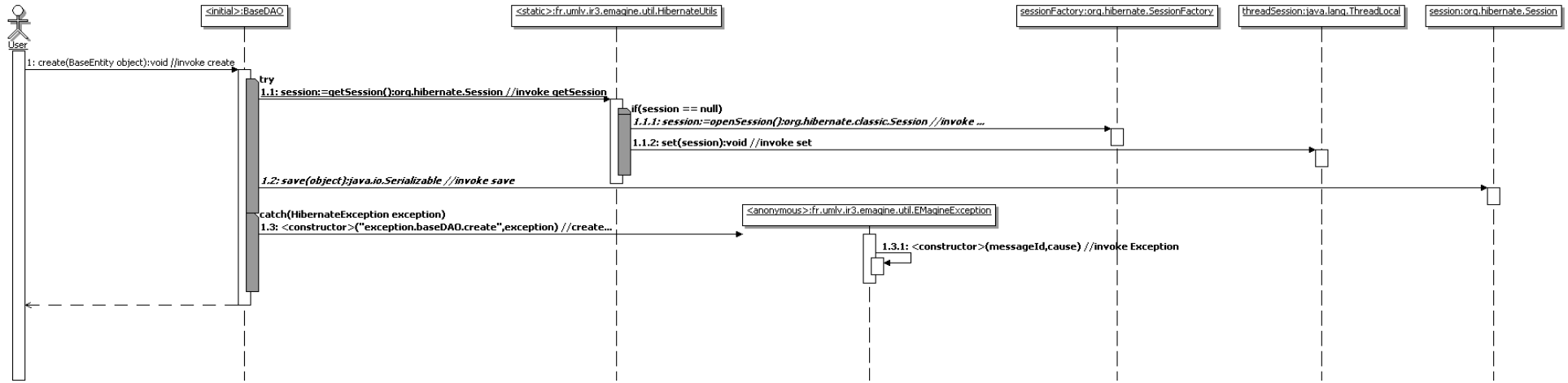
Définir un point de contact unique avec le sous-système, à savoir un objet façade qui l'enveloppe. Cet objet façade présente une interface unique unifiée, et il est chargé de collaborer avec les composants du sous-système.

5. DIAGRAMMES GENERIQUE

Les diagrammes génériques sont des diagrammes qui apparaissent plusieurs fois dans les Use Cases, afin de limiter le nombre de diagrammes de séquences quatre diagrammes de base sont utilisés.

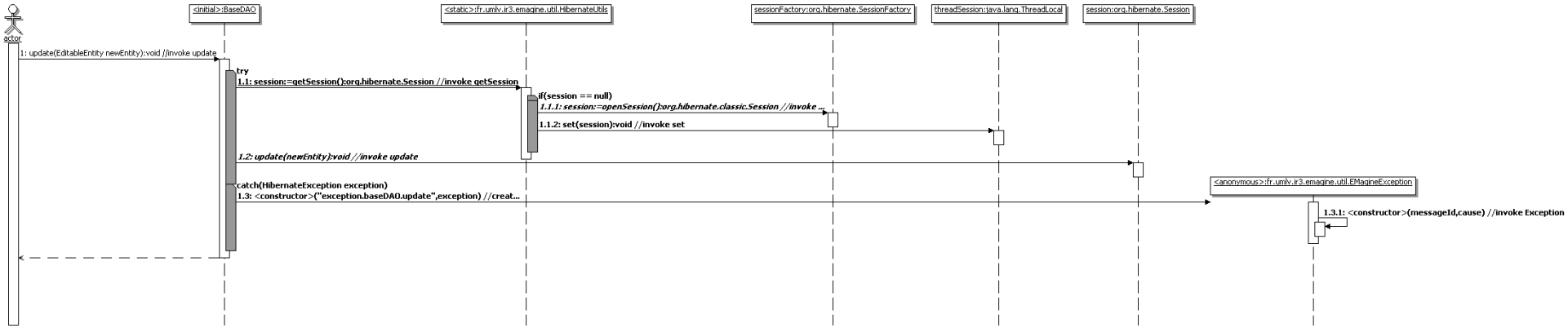
5.1. CRÉATION

Le diagramme de création créer un objet métier à partir de l'action de l'utilisateur, ce diagramme est générique, il s'applique quelque soit le type de l'objet.



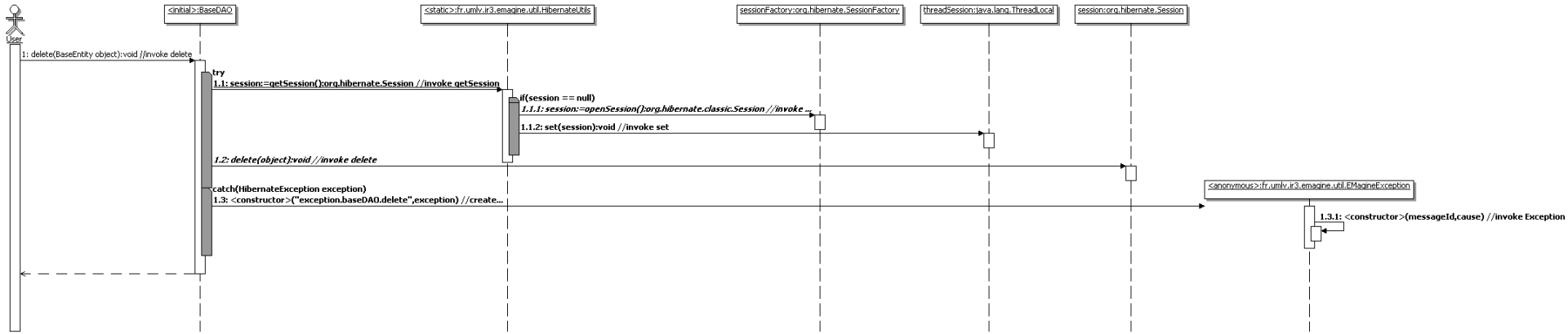
5.2. MODIFICATIONS

Le diagramme de modification modifie un objet métier à partir de l'action de l'utilisateur, ce diagramme est générique, il s'applique quelque soit le type de l'objet.



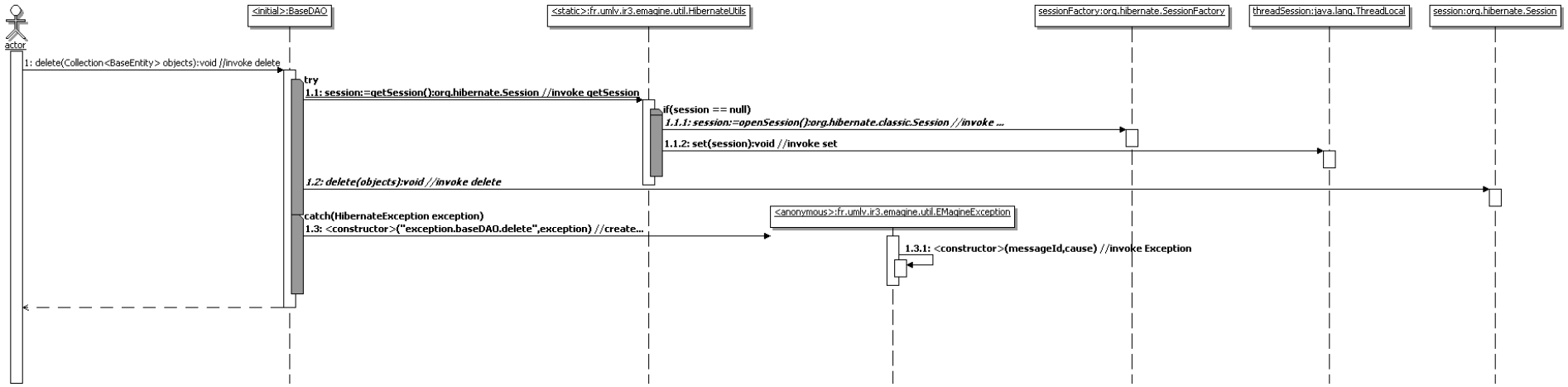
5.3. SUPPRESSION

Le diagramme de suppression supprime un objet métier à partir de l'action de l'utilisateur, ce diagramme est générique, il s'applique quelque soit le type de l'objet.



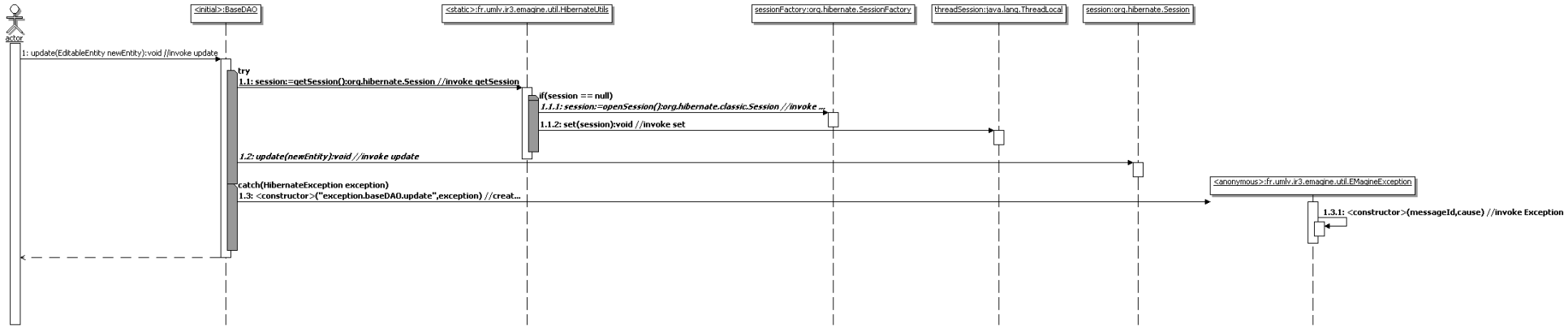
5.4. SUPPRESSION D'UNE COLLECTION

Le diagramme de suppression supprime une collection de classe métier à partir de l'action de l'utilisateur, ce diagramme est générique, il s'applique quelque soit le type de l'objet.



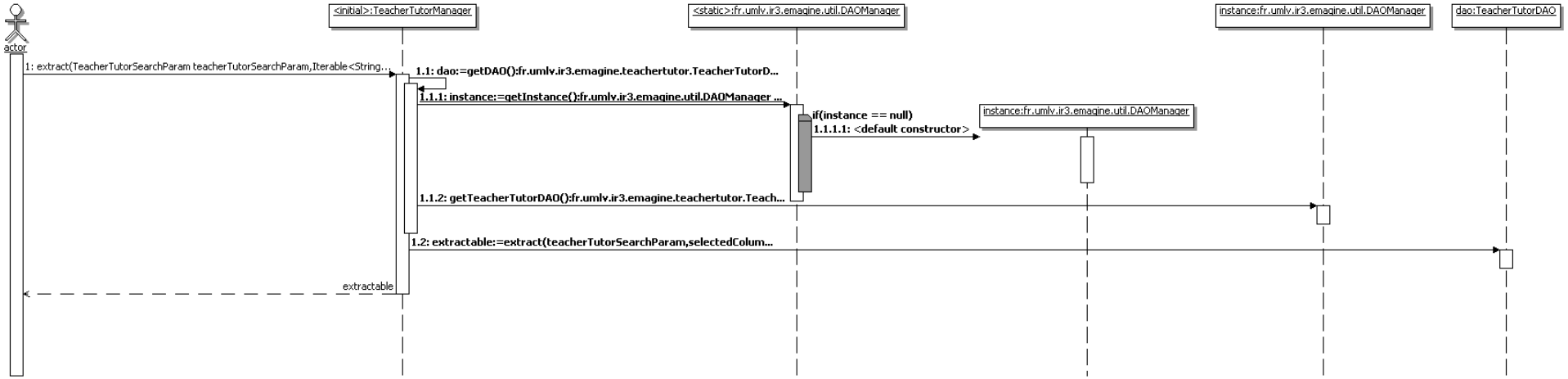
5.5. CHARGEMENT

Le diagramme de chargement charge un objet métier à partir de l'action de l'utilisateur, ce diagramme est générique, il s'applique quelque soit le type de l'objet.



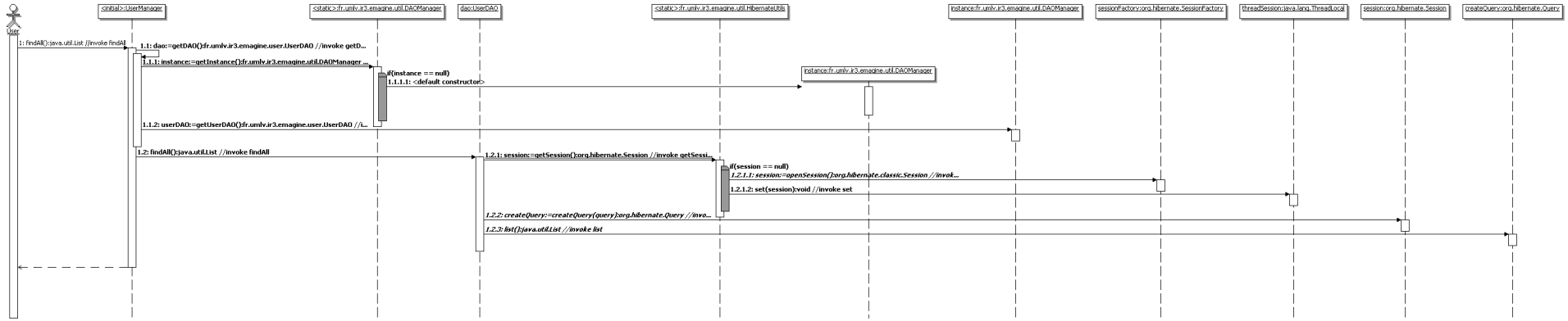
5.6. EXTRACTIONS

Le diagramme d'extraction extrait un objet métier à partir de l'action de l'utilisateur vers un flux, ce diagramme est générique, il s'applique quelque soit le type de l'objet.



5.7. RECHERCHE

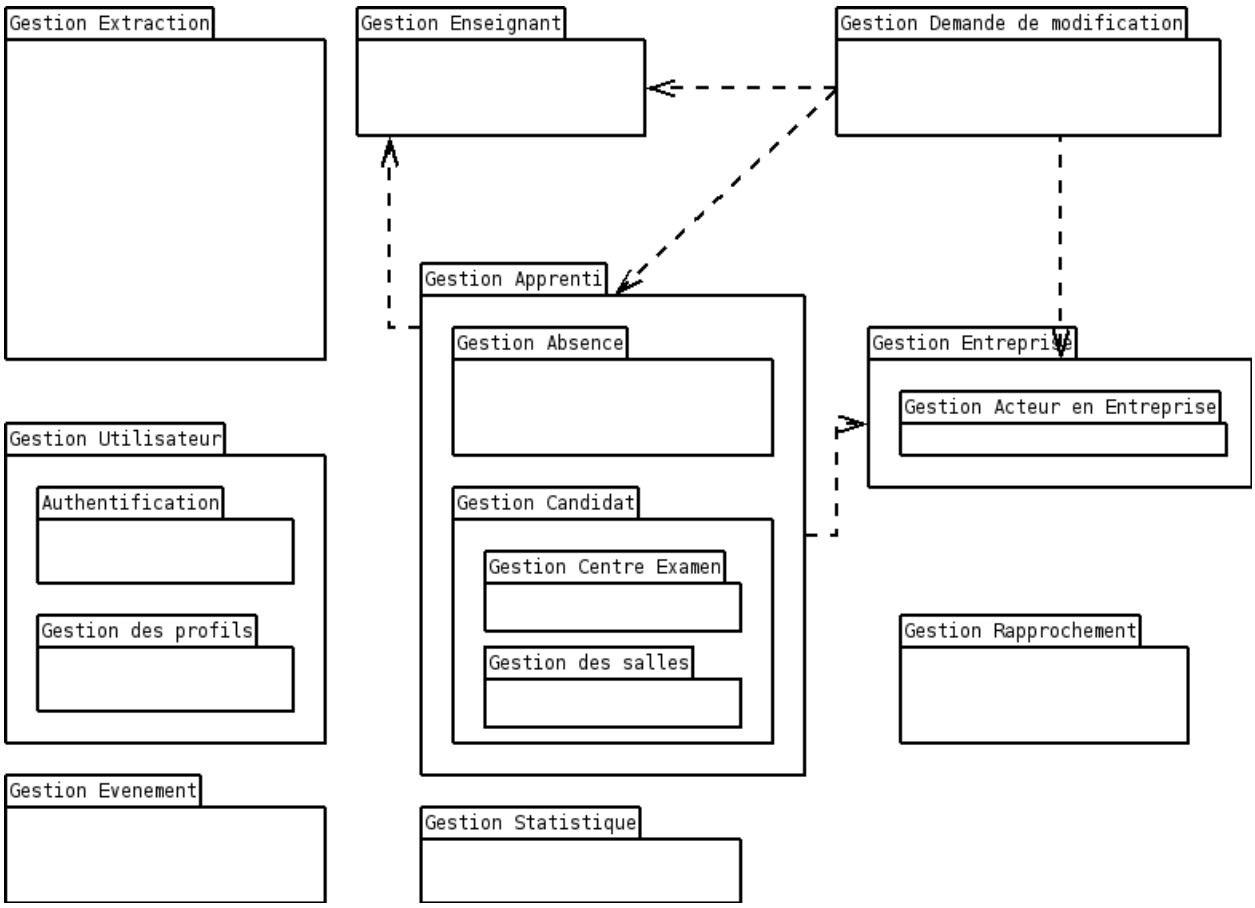
Le diagramme de recherche recherche un ou plusieurs objets métiers à partir d'une liste de paramètres. Ce diagramme est générique, il s'applique quelque soit le type de l'objet. Ici, et pour bien comprendre le principe, nous avons utilisé la classe User. Mais ce diagramme est applicable à n'importe quel classe métier.



6. PAQUETAGES

Ce chapitre liste les Uses Case regroupés en paquetages. La philosophie des paquetages a été d'en créer plusieurs de petite tailles, plutôt que d'en créer quelque gros pour éviter d'avoir des paquetages qui contiennent des fonctionnalités qui n'ont aucun lien entre elles.

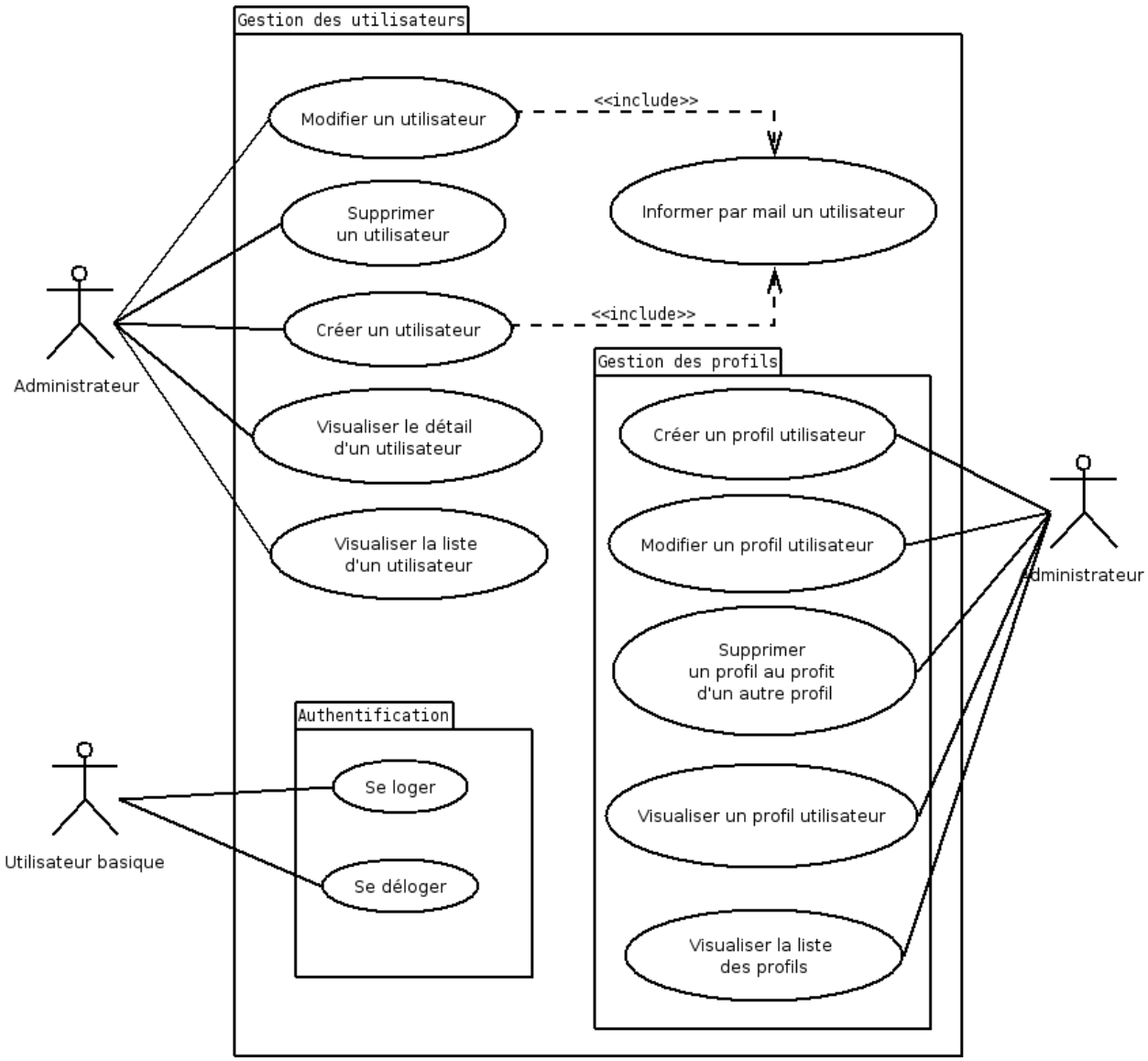
Le schéma suivant représente une vue globale du système :



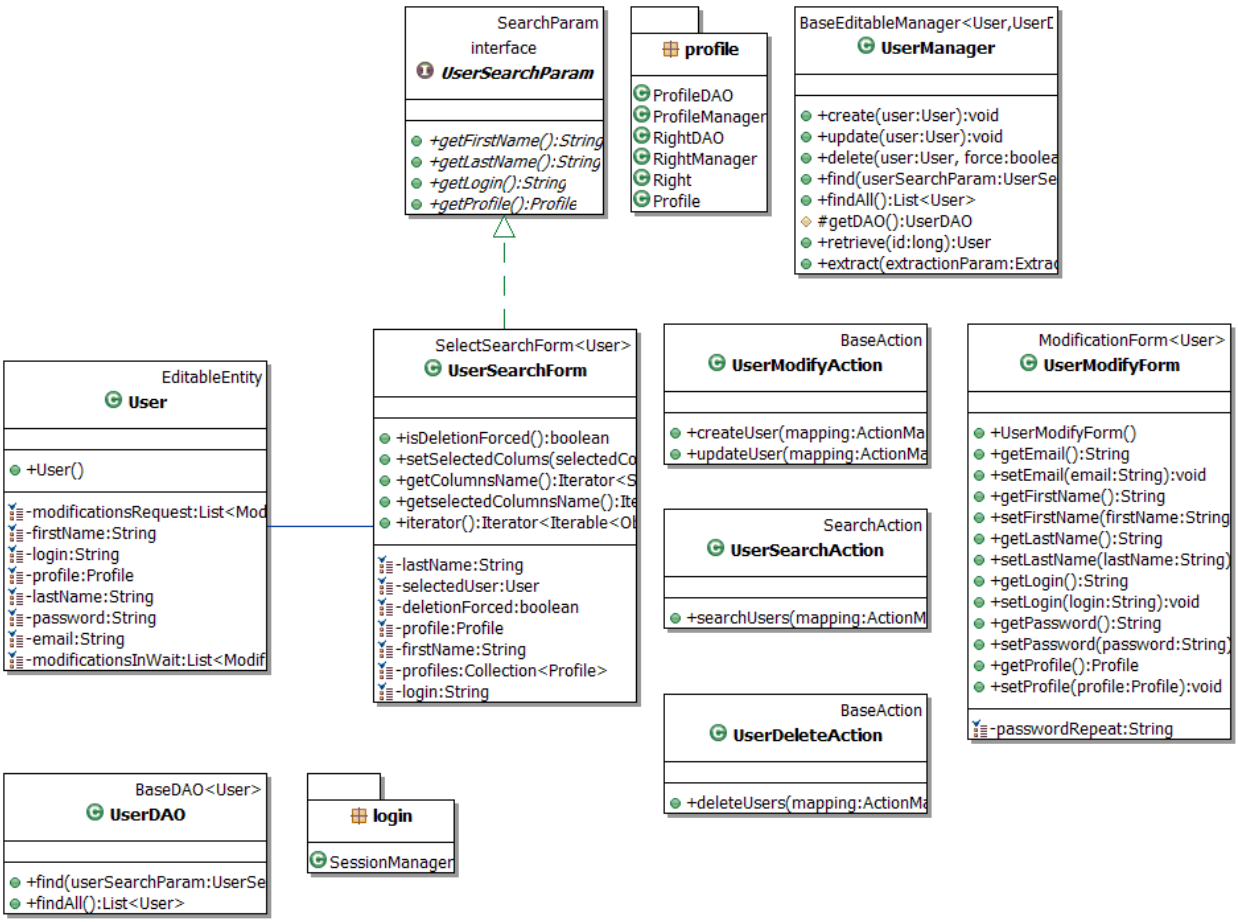
6.1. Paquetage : Gestion des Utilisateurs

Ce paquetage regroupe toutes les fonctionnalités liées à la manipulation des utilisateurs.

Diagramme d'interaction :

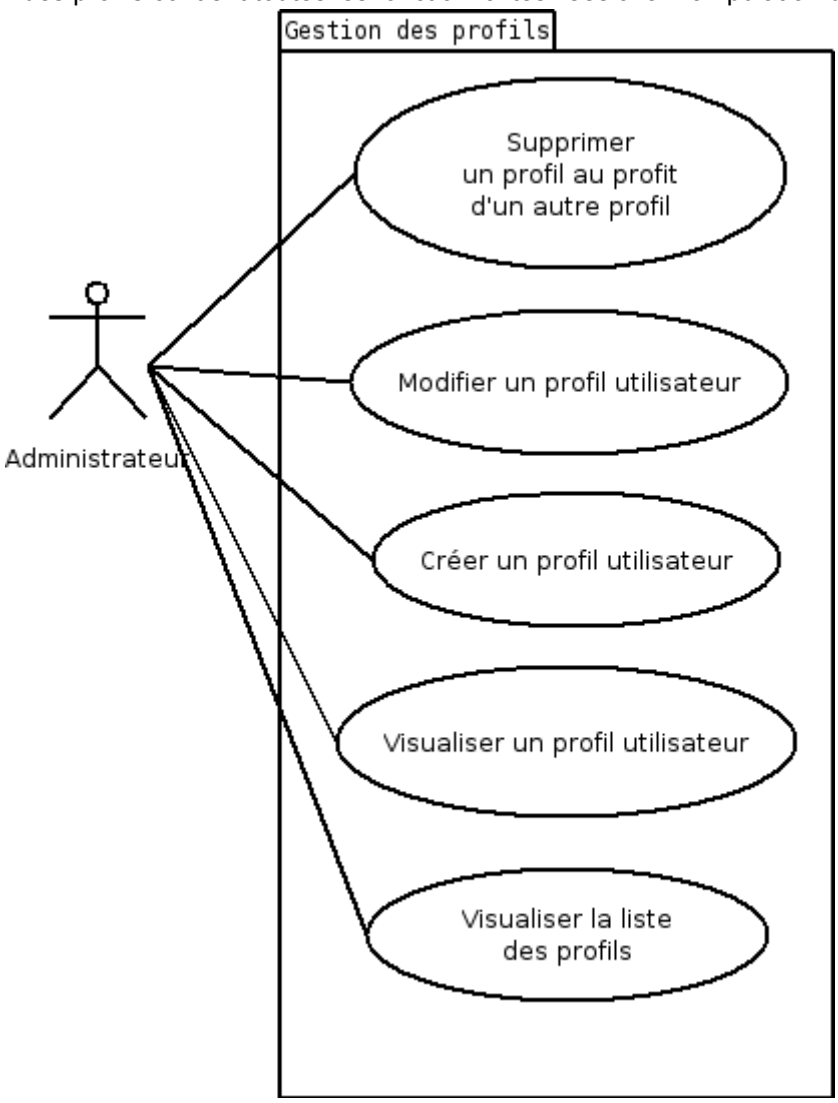


➢ Diagramme de classes :

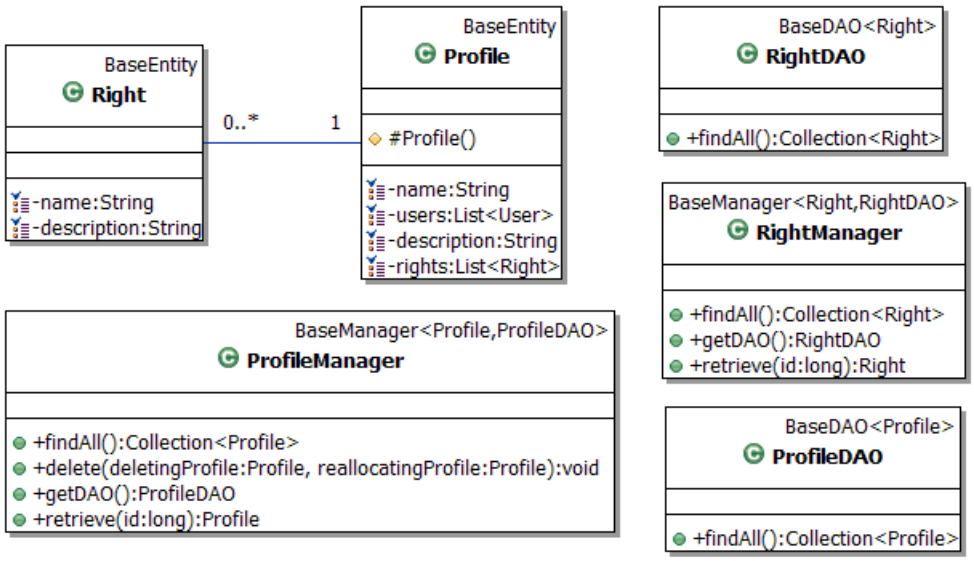


a. Paquetage : Gestion des Profils

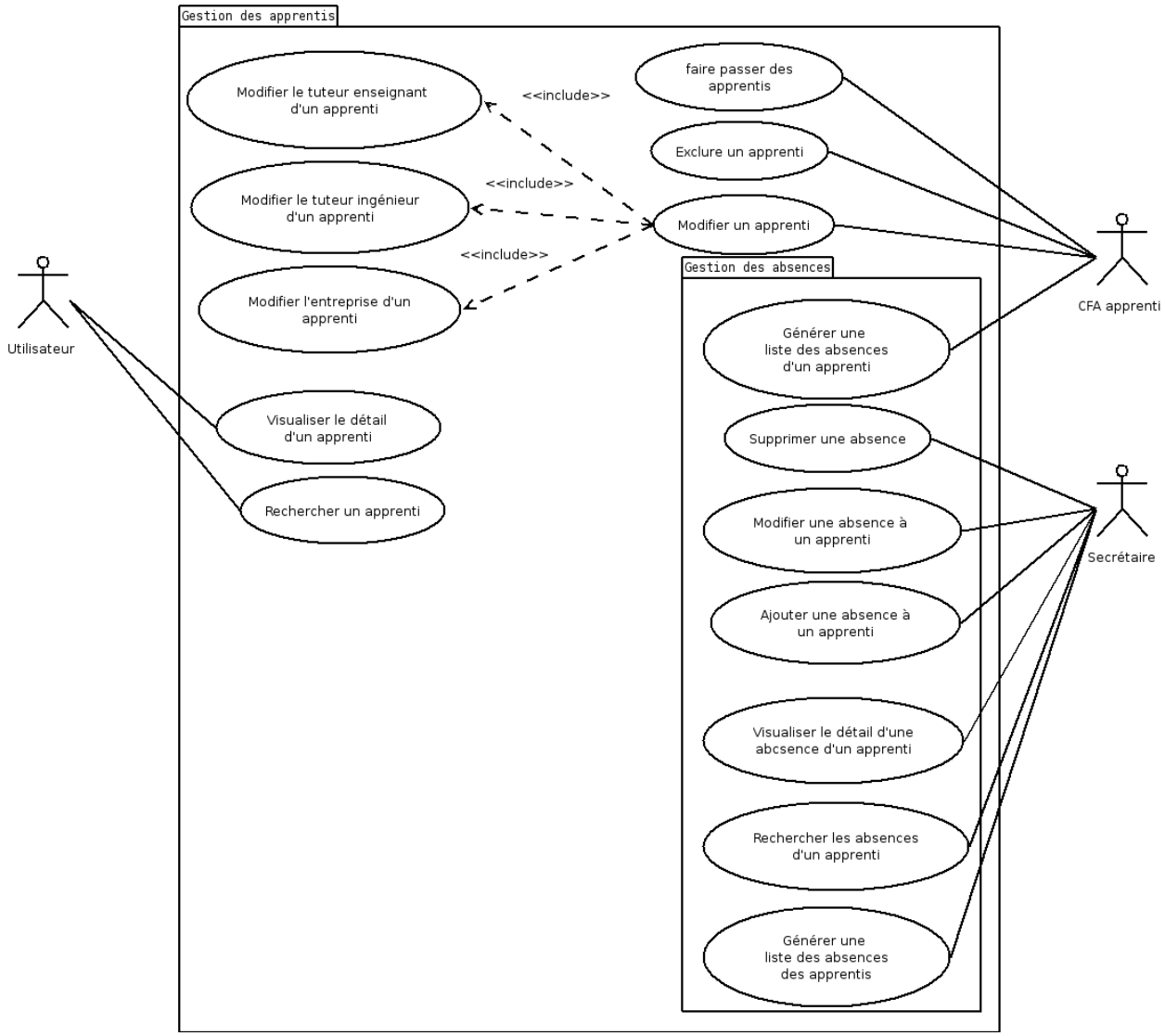
La gestion des profils contient toutes les fonctionnalités liées à la manipulation de profils.



➢ Diagramme de classe

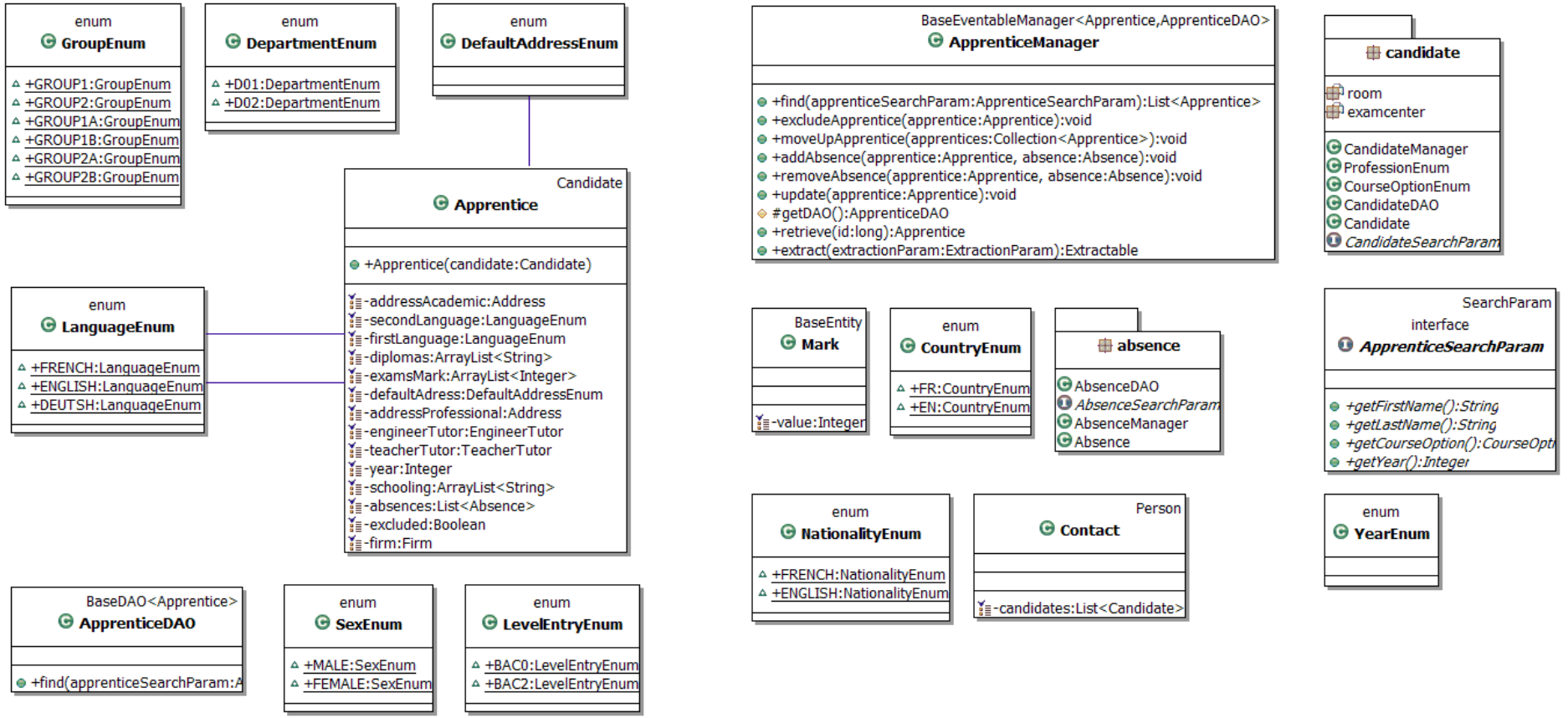


6.2. GESTION DES APPRENTIS



Le module de gestion des apprentis permet de rechercher et de modifier des apprentis. De plus, ce module gère les tutelles, le changement d'entreprise et les absences des apprentis.

➢ Diagramme de classe

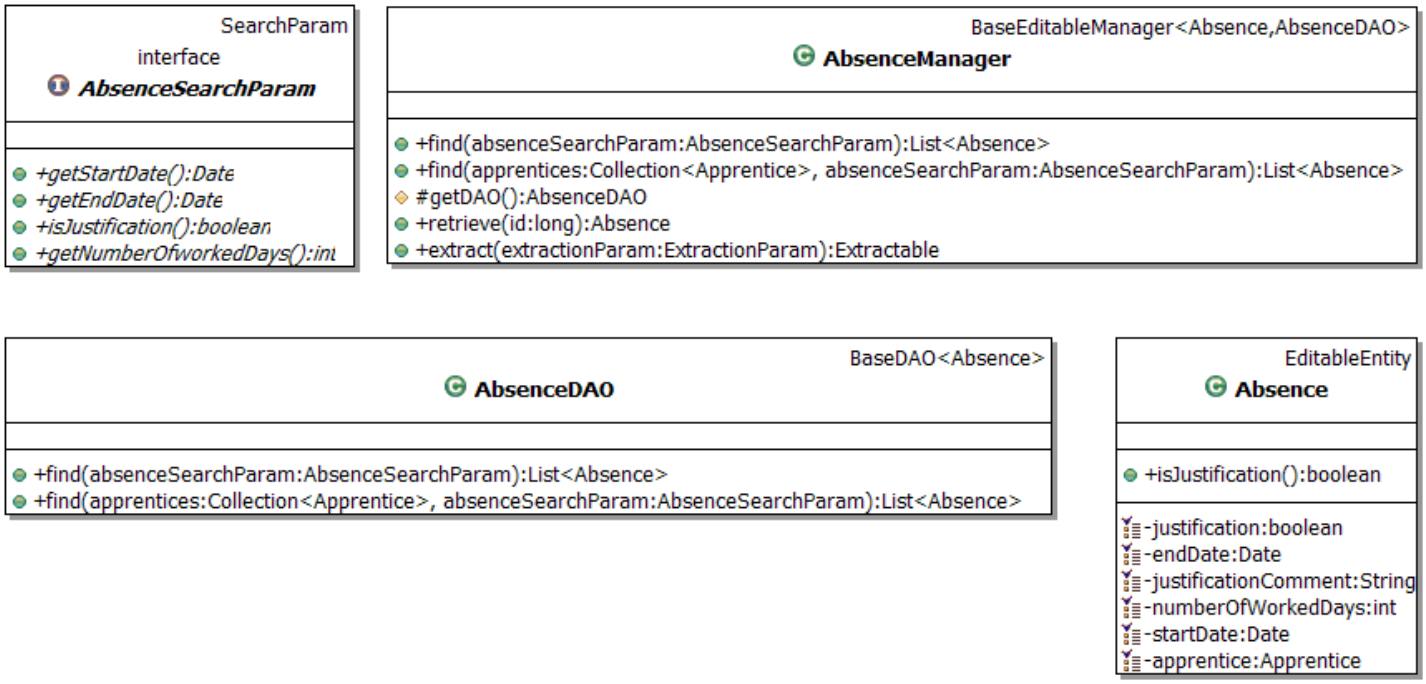


Les énumérations représentent ici des données qui sont amenées à ne jamais changer. Nous les complèteront dans la suite du processus.

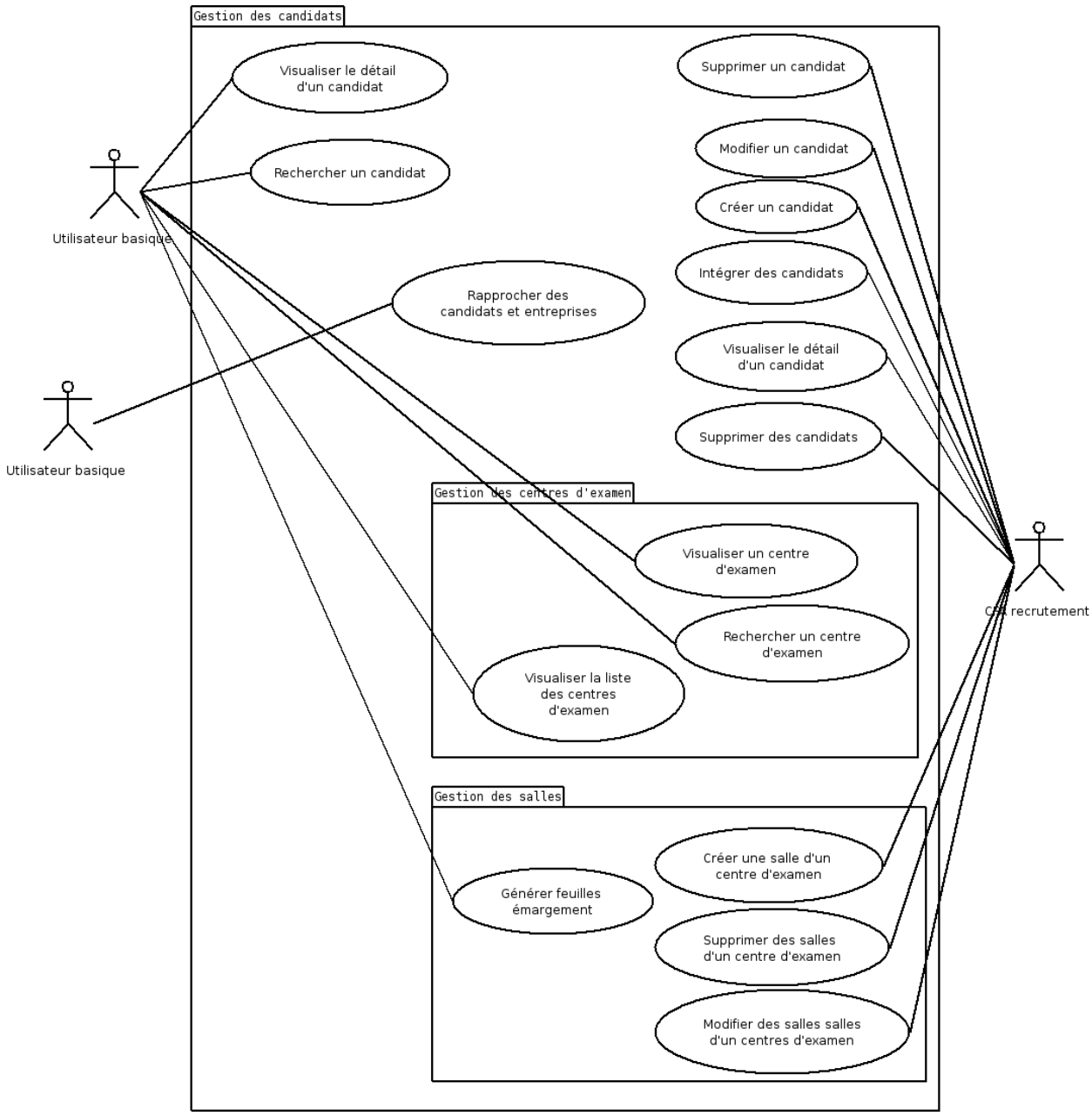
a. Gestion des absences

Le module de gestion des absences permet de rechercher, d'ajouter, de modifier, de supprimer et de générer des extraction sur les absents. Ce module sera accessible à partir du menu général de l'application et non dans le module des apprentis.

Diagramme de classe

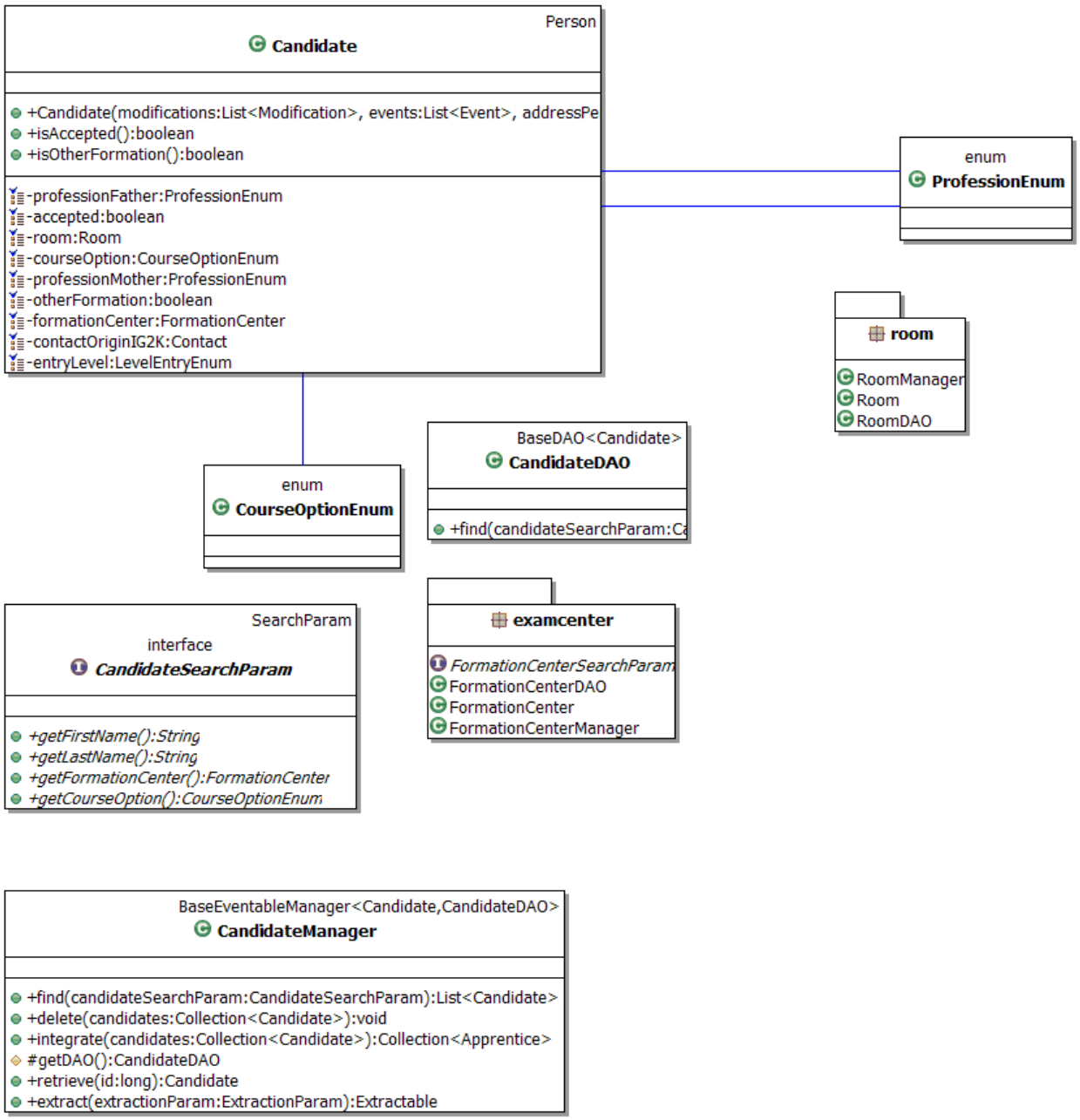


b. Paquetage : gestion des candidats



Le module de gestion des candidats permet de rechercher, d'ajouter, de modifier, de supprimer des candidats. Un système de gestion de salles est également est mis à disposition. Ce module permet de générer des feuilles d'émargement pour les concours d'entrée à Ingénieurs 2000.

➢ Diagramme de classe

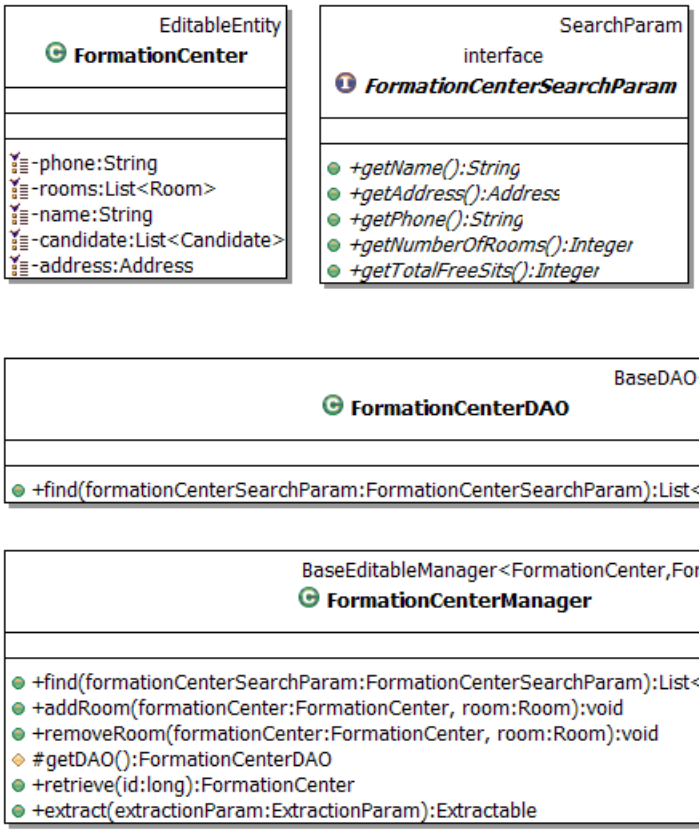


Le candidat possède un constructeur comportant tous les paramètres à donner sa classe mère pour pouvoir correctement initialiser tous ses champs. Il en vat de même, ainsi l'on peut créer un apprenti à partir d'un candidat juste en le passant en paramètre du constructeur de l'apprenti.

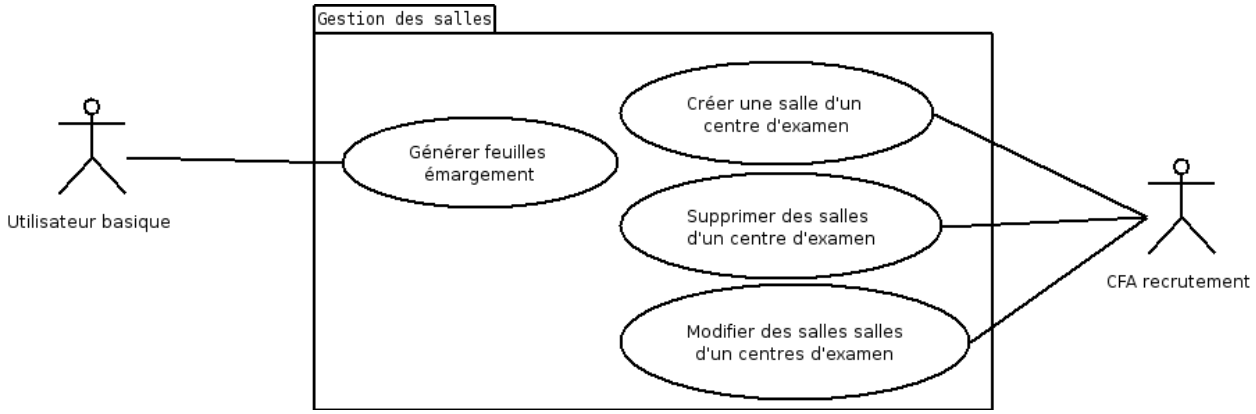
Paquetage : gestion des centres d'examen

La gestion du centre d'examen a un lien fort avec les candidats. C'est pour cette raisons qui est dans gestion des candidats.

Diagramme de classe

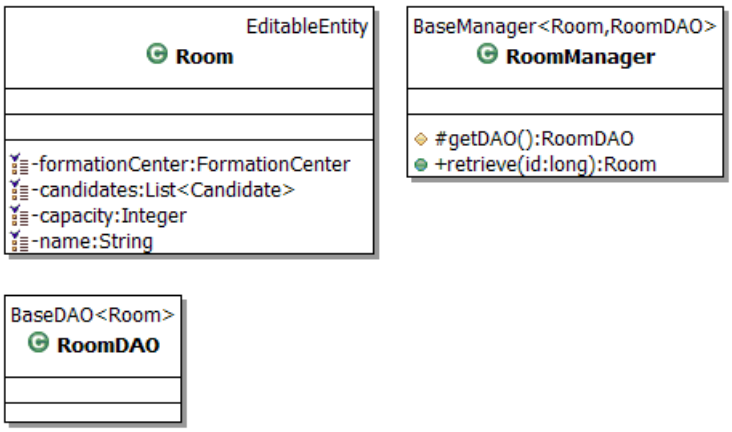


Paquetage : gestion des salles



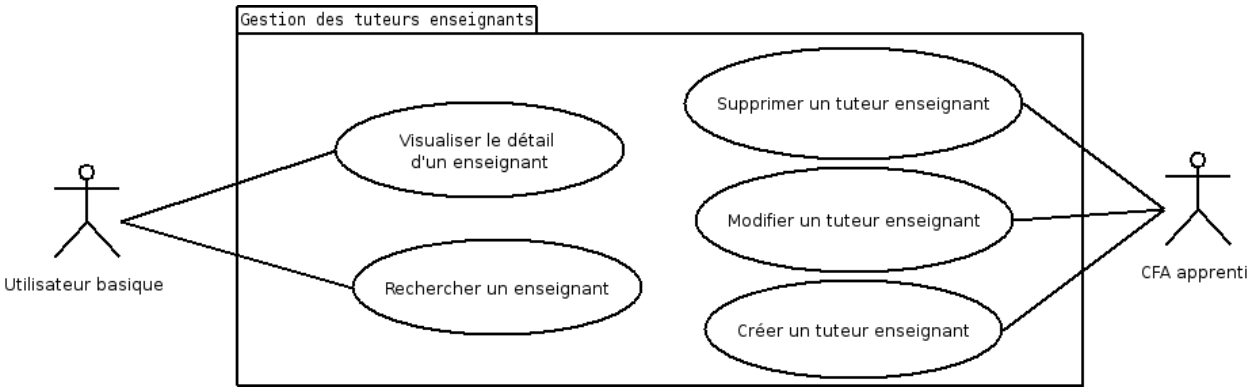
La gestion des salles d'examen étant assez complexe il a été décider d'en faire un paquetage.

Diagramme des classes

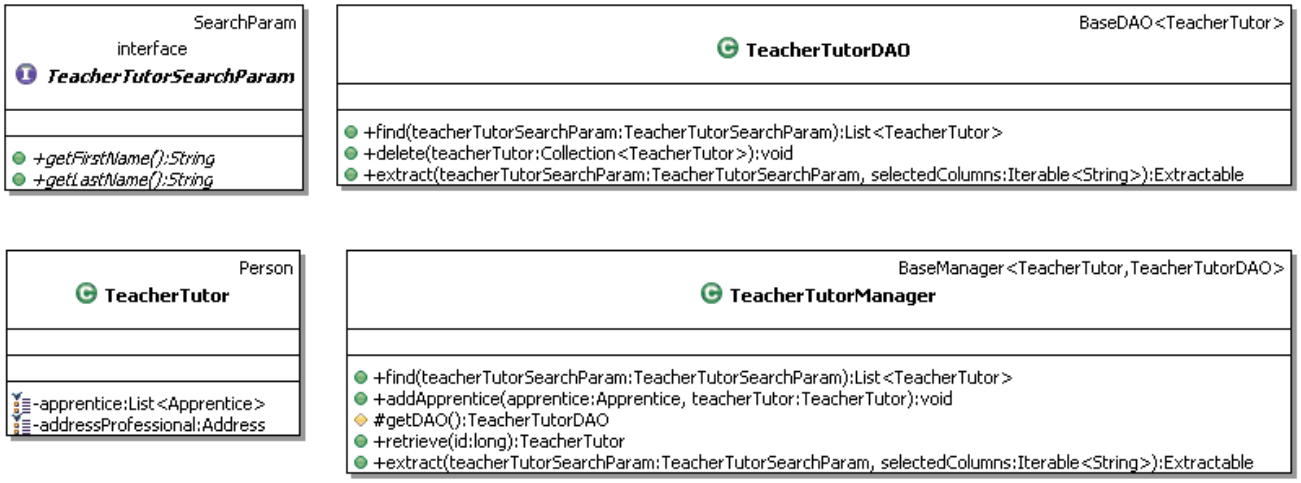


6.3. Gestion des Tuteurs Enseignants

Le module de gestion des tuteurs enseignants permet de rechercher, d'ajouter, de modifier, de supprimer des tuteurs enseignants. Il s'agit simplement de l'annuaire des enseignants étant tuteur ou ayant été tuteur d'au moins un apprenti.

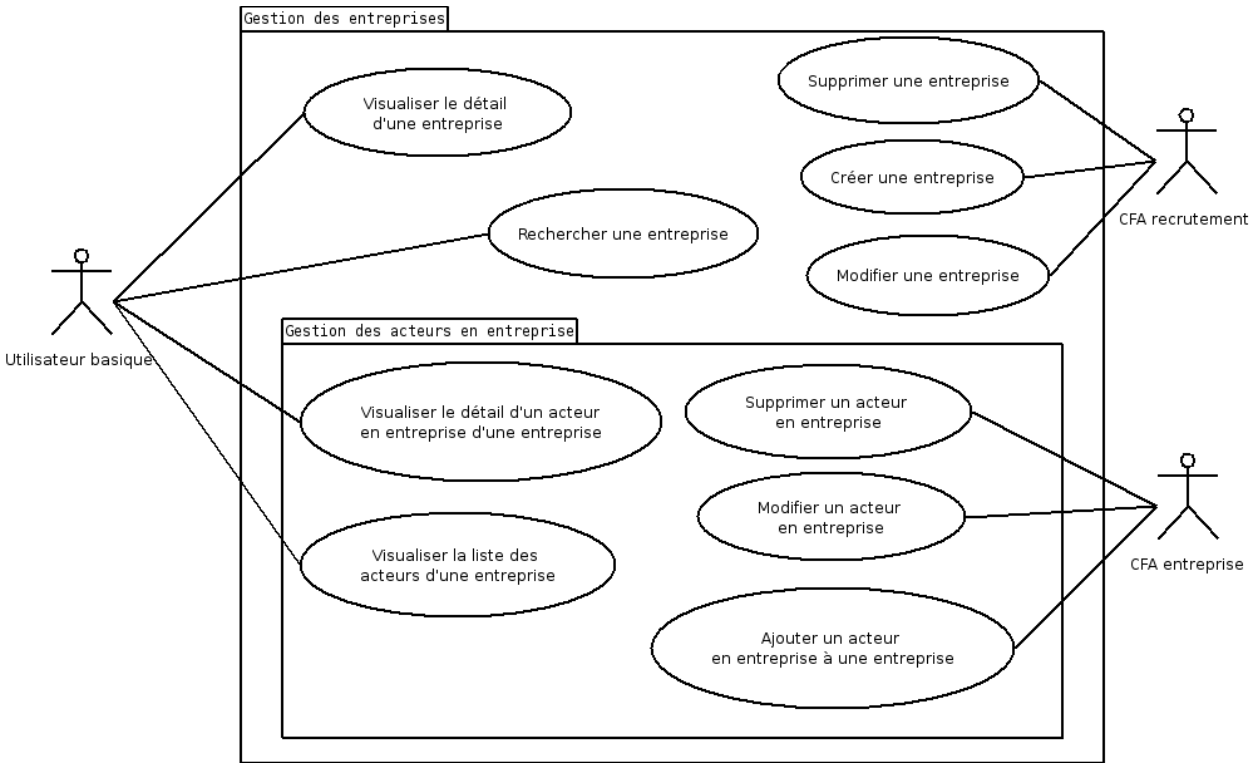


➢ Diagramme de classe

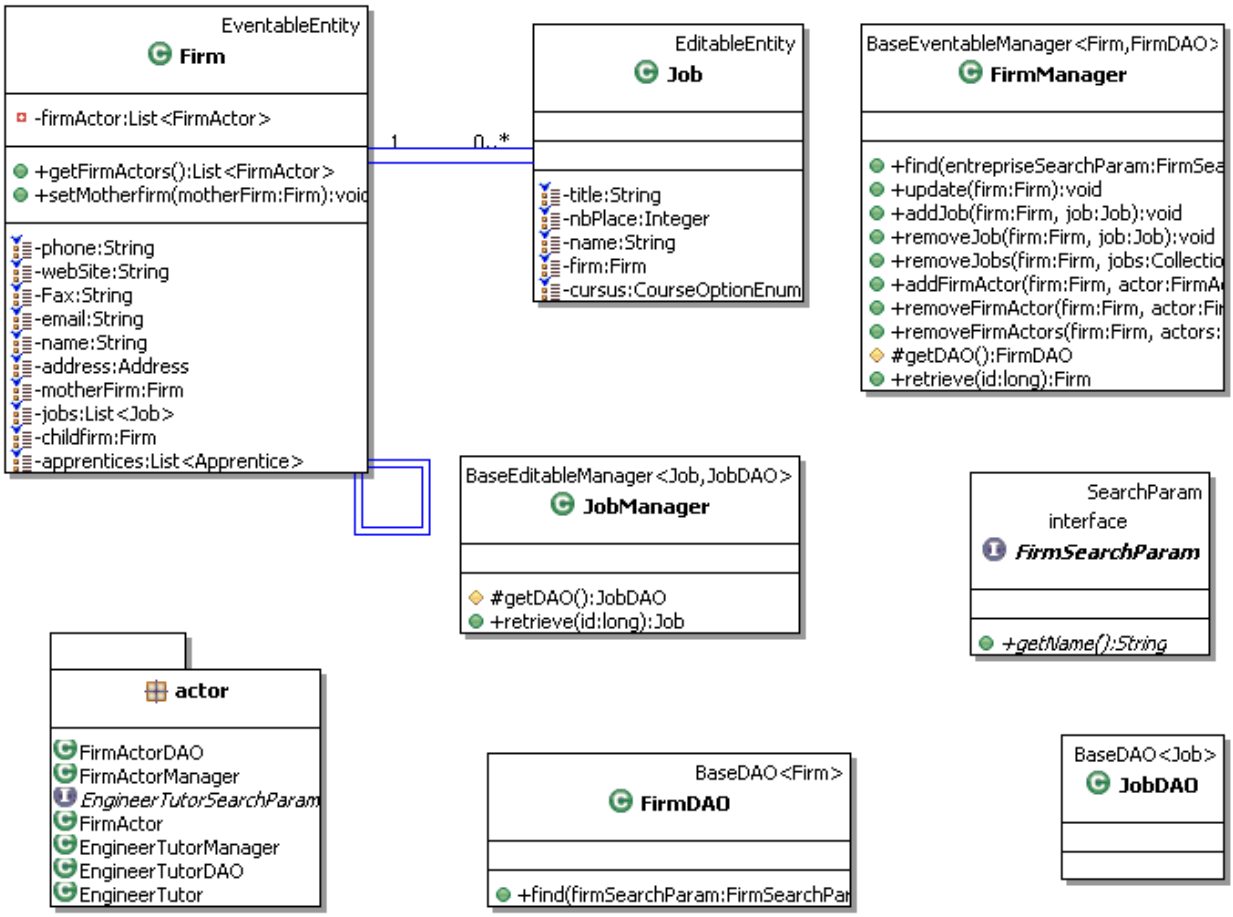


6.4. GESTION DES ENTREPRISES

Ce paquetage est un paquetage conséquent. Il regroupe toutes les fonctionnalités liées aux entreprises. Le module de gestion des entreprises permet de rechercher, d'ajouter, de modifier, de supprimer des entreprises.



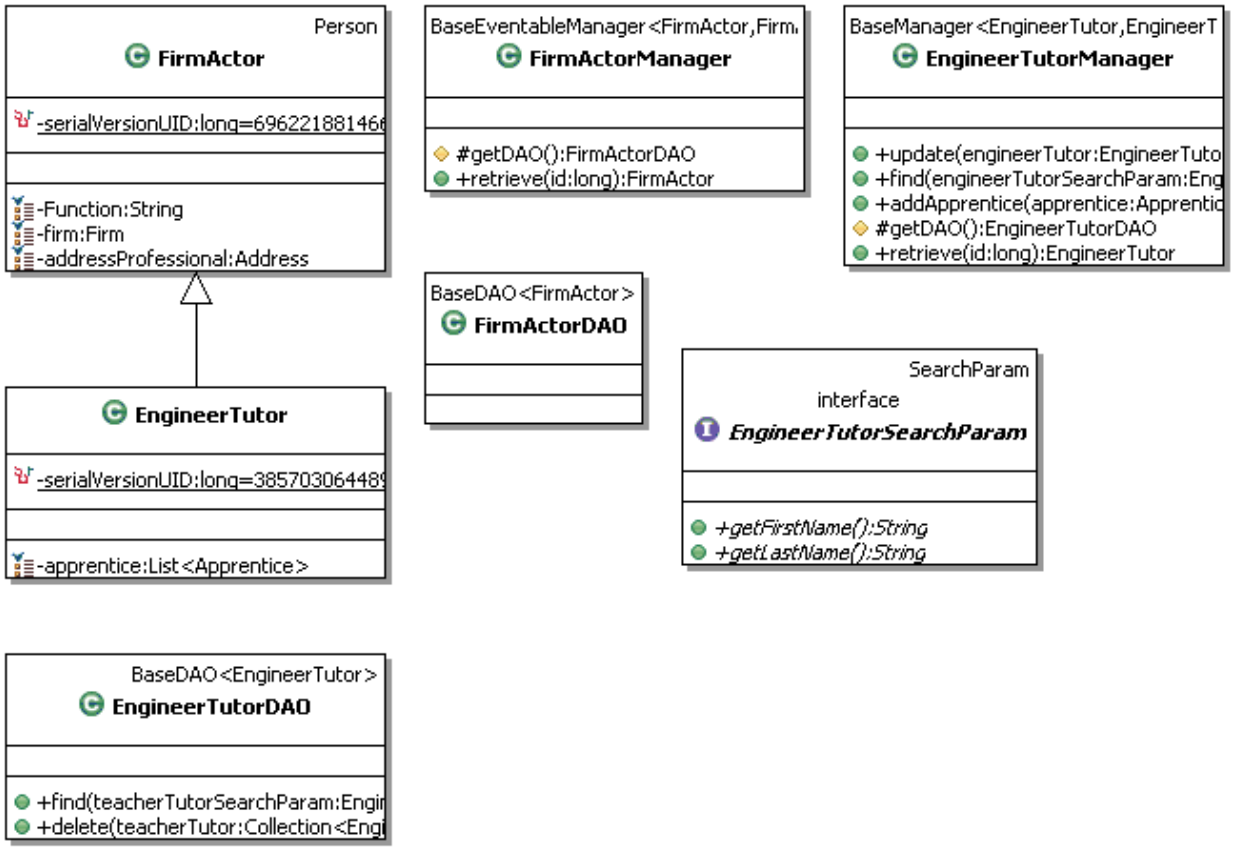
➢ Diagramme de classe



a. Gestion des acteurs en entreprise

Ce paquetage a été mis dans « gestion entreprise » parce qu'il a un lien fort avec la gestion des entreprises, si il n'y a pas d'entreprise il n'y aura pas d'acteurs, néanmoins il peut y avoir des entreprises sans qu'il y est d'acteur.

➢ **Diagramme de classe**



6.5. Gestion des statistiques

Ce module permet de créer, de modifier et de visualiser des statistiques.

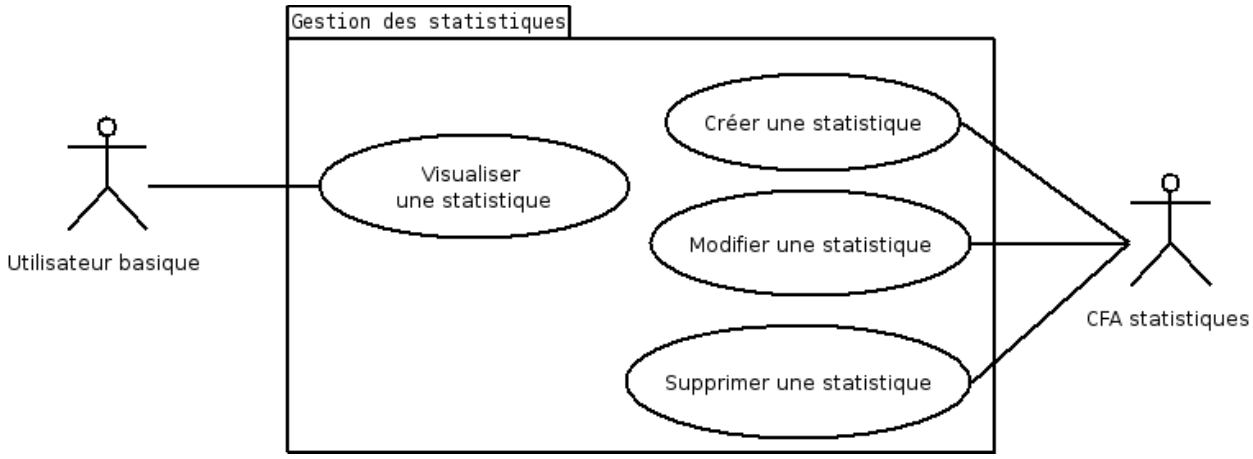
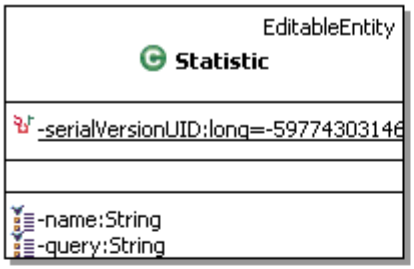
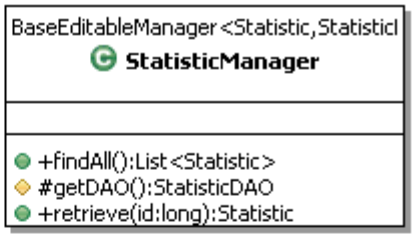
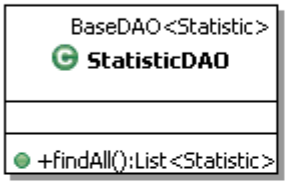
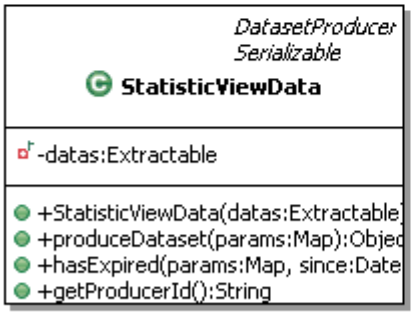


Diagramme de classe





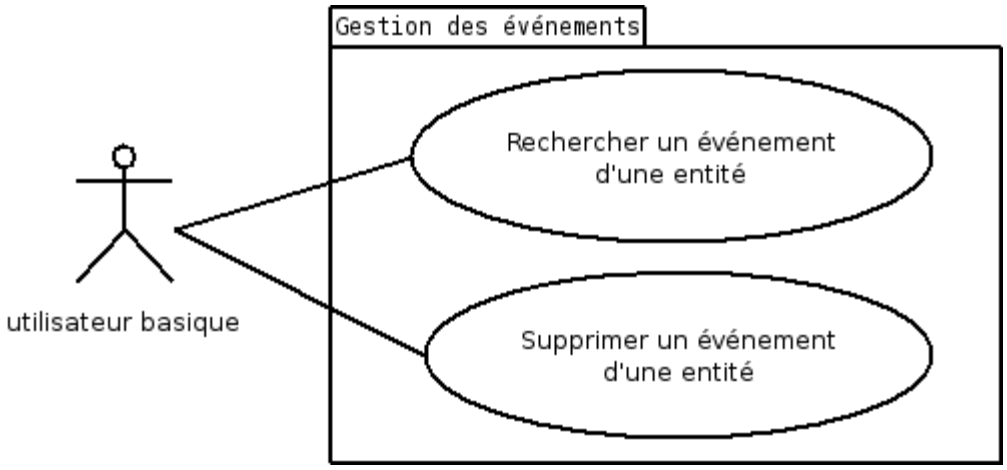




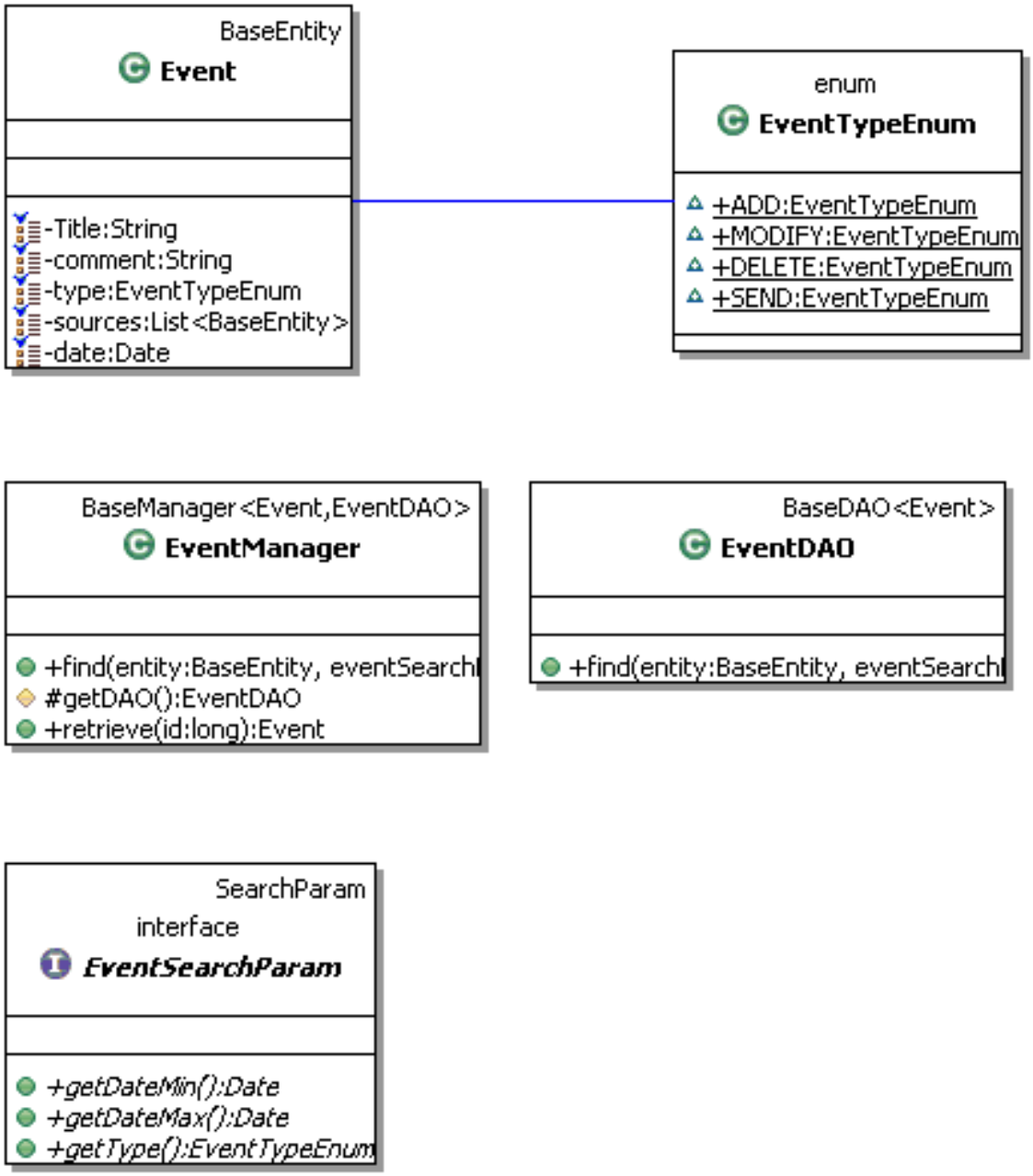
6.6. Gestion des événements

Un événement est le terme qui désigne l'historique laissé par une entité (entreprise, enseignant, candidat, apprenti ou acteur en entreprise) dans la base de donnée. Un événement peut être une modification réussie de l'entité rattachée, un envoie de mailing à l'entité ou un publipostage de l'entité.

La gestion de ces événements a été regroupée dans un seul paquetage étant donné que quelque soit l'entité rattachée, la manière de le rechercher ou de le supprimer est la même.



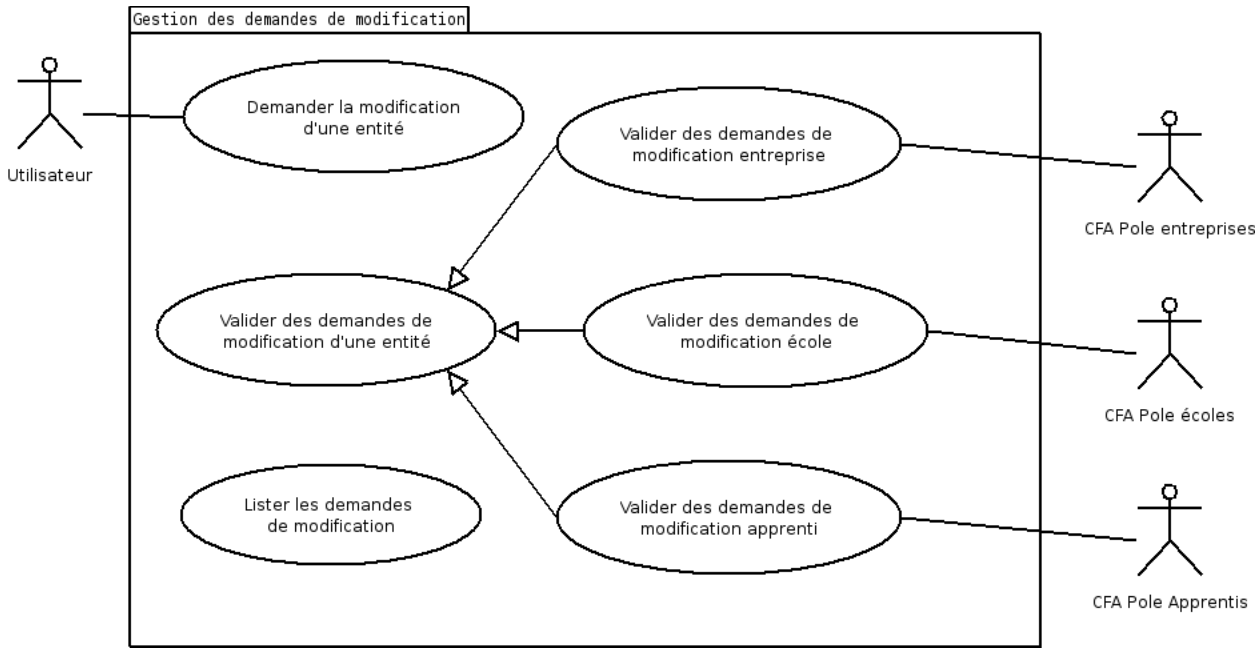
➢ **Diagramme de classe**



6.7. Gestion des demandes de modification

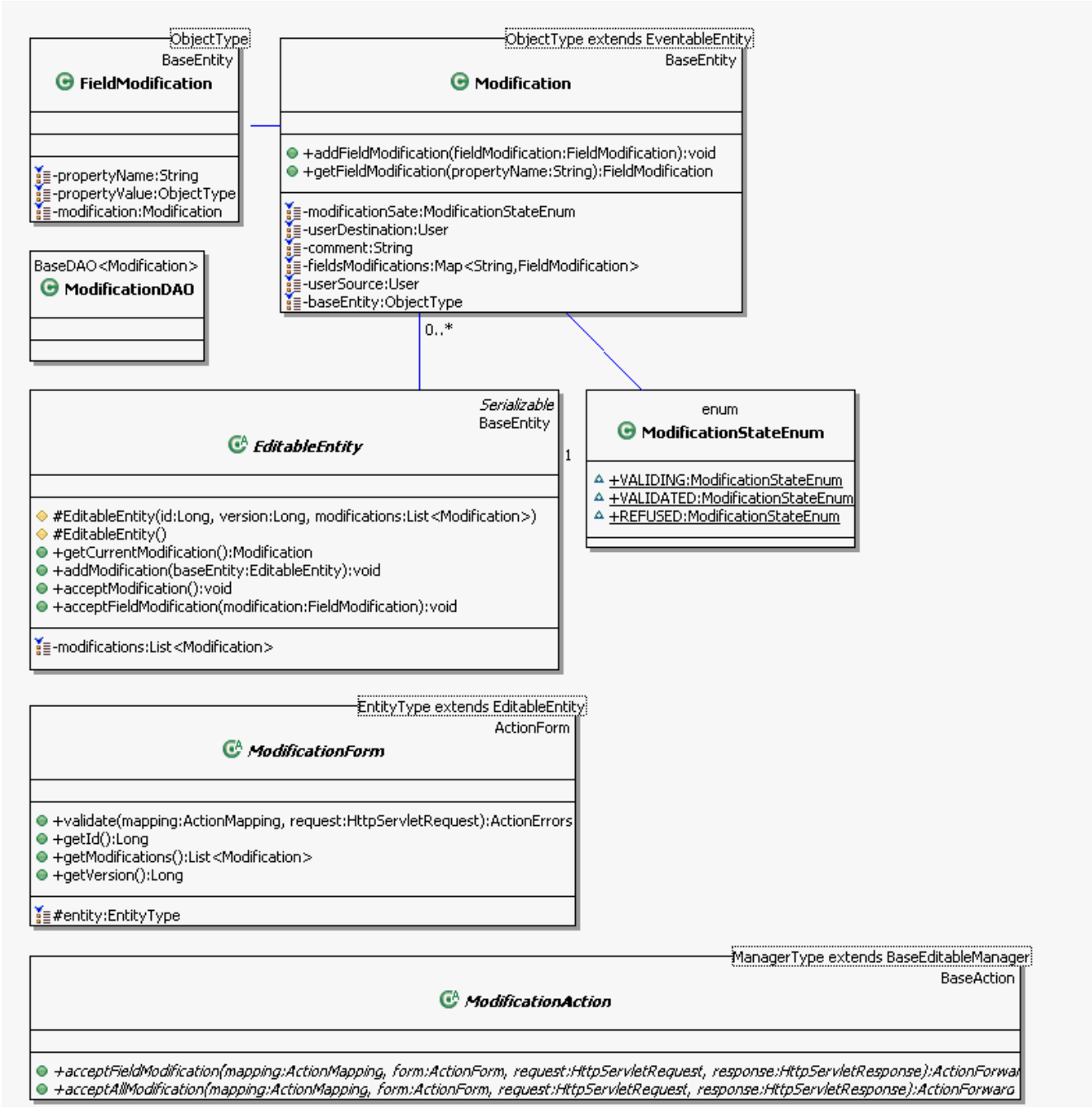
Le système de modification de données sur les acteurs du système est assez complexe, c'est un système de demande et validation. Un paquetage a donc été créer spécialement pour ce système.

- **Diagramme de paquetage**

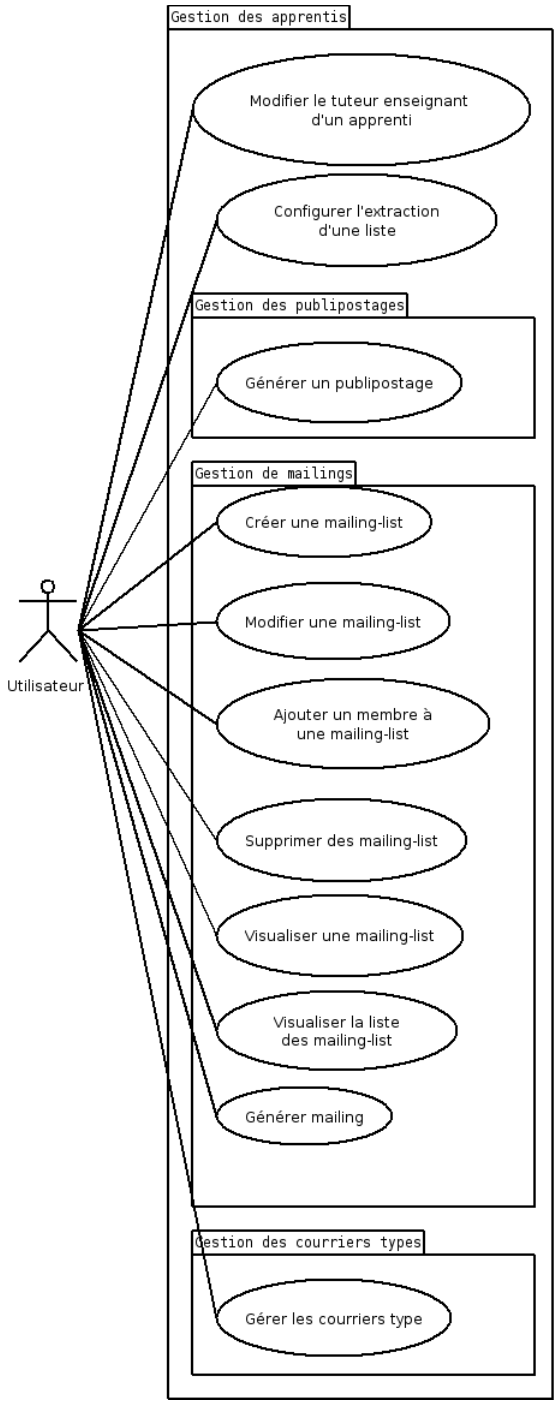


-

➢ Diagramme de classe



6.8. GESTION DES EXTRACTIONS



Le paquetage d'extraction contient toutes les fonctionnalités liées à l'extraction et la transformation des données du système. Ce paquetage étant assez volumineux. Plusieurs sou-paquetages ont été crées.

a. Gestion des publipostages

L'utilisateur reçoit un fichier correspondant à la liste visionnée, puis le publie grâce à Word, en ayant précisé avant cela, le courrier type qui doit être rattaché au publipostage.

b. Gestion de mailings

L'utilisateur crée une mailing-list en précisant simplement son nom. Il est redirigé ensuite sur la page de « Modification d'une mailing-list » qui permet d'ajouter des personnes à la liste.

c. Gestion des courriers types

Permet de regrouper toutes les fonctionnalités pour la gestion des courriers type.

7. RESSOURCES

Pour permettre un suivi et un déploiement de l'information rapidement un site internet à été mis en place. Les dates importantes de rendu des différents livrables sont détaillés ci-après.

7.1. SITE INTERNET

Un site Internet est à disposition sur *http://emagine.berlios.de* afin de permettre au client de suivre le développement du projet eMagine.

7.2. DATES IMPORTANTES

Date	Livrable
10/10/05	Cahier des charges
24/10/05	Cahier des charges fonctionnelles
14/11/05	Spécification de l'interface graphique
05/12/05	Cahier des charges techniques
23/01/06	Livraison du logiciel version bêta
27/02/06	Livraison du logiciel version 1.0

CONCLUSION

La documentation développeur permet d'intégrer un développeur au sein de l'équipe de développement plus rapidement. Par ailleurs, un oeil externe aura plus de facilité à comprendre l'application grâce à cette documentation.

Parallèlement à cette documentation, une documentation technique type Javadoc a été mise en place et sera fournie avec le code source final.