Projet DELTA

Cahier des charges techniques

Batoussa MOUGAMADOU
Saliou NGOM
Vincent BOISSIN
Gabriel DUPONT
Julien KIRSCH
Frederic GARCIA

IUP3-Maitrise 2002-2003
Informatique et Mathématiques

Table des matières

Chapitre 1

Les Diagrammes

Pour la création du diagramme de classes, l'idée est de regrouper les différents paquetages qui s'utilisent de façon identique.

1.1 Diagramme de compte et de droit

Pour commencer, voyons le diagramme des interactions entre les différentes classes :

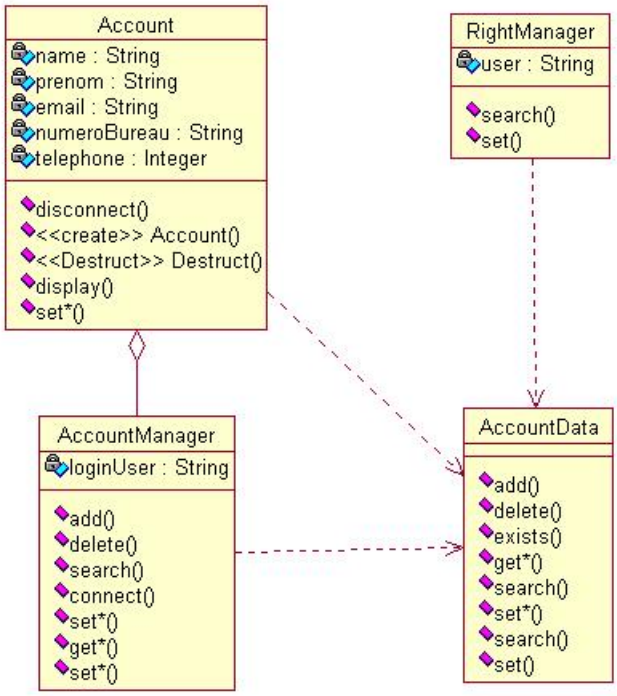


Diagramme de classes.

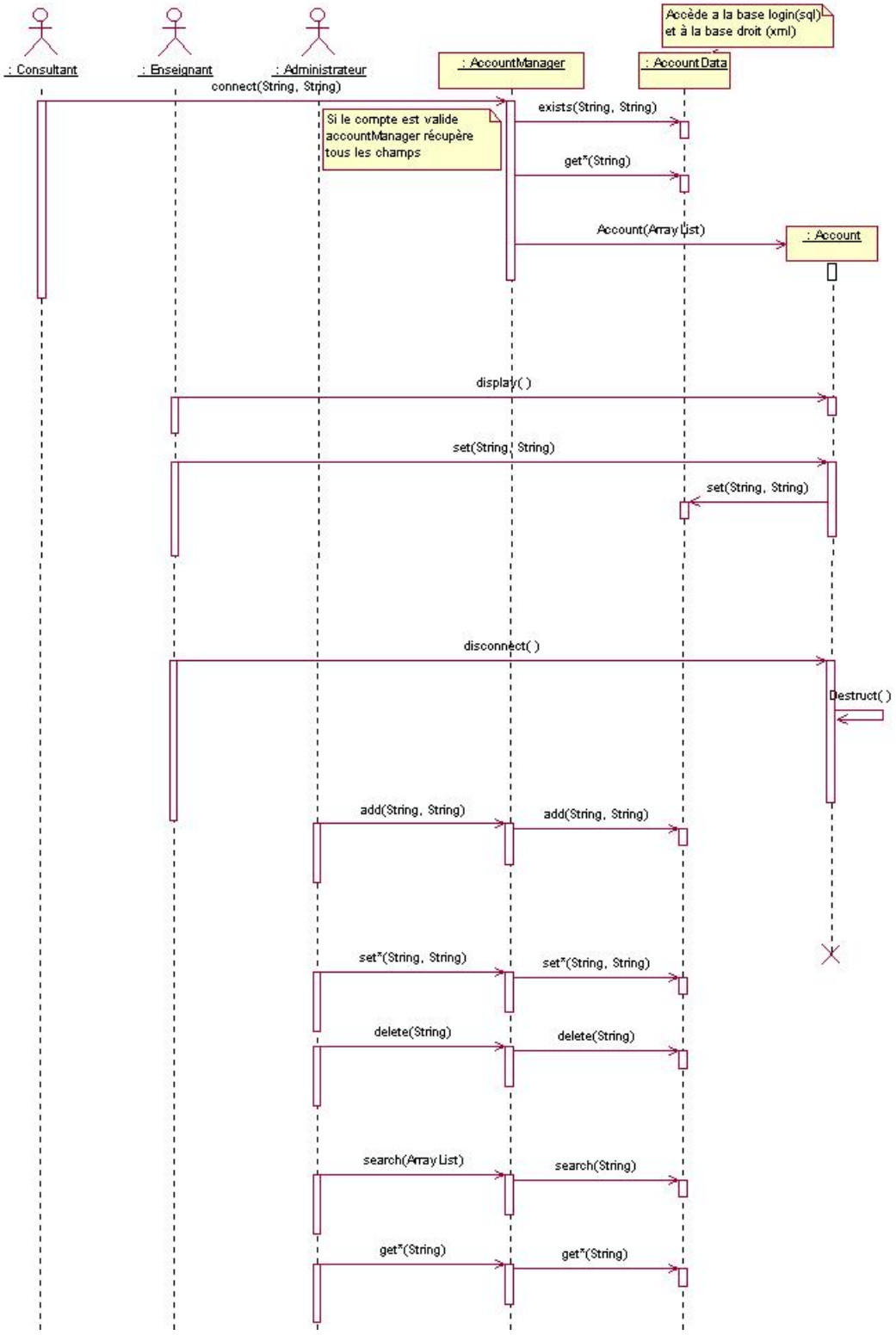
Les paquetages compte et droit a été créé pour gérer les droits d'identification et d'accés des différents acteurs. Dans ce diagramme nous avons les objets :

- Account : C'est un objet qui identifie un utilisateur par son login et son mot de passe. C'est l'objet de base du paquetage compte. Seul l'administrateur a le droit de le modifier ou de le supprimer.

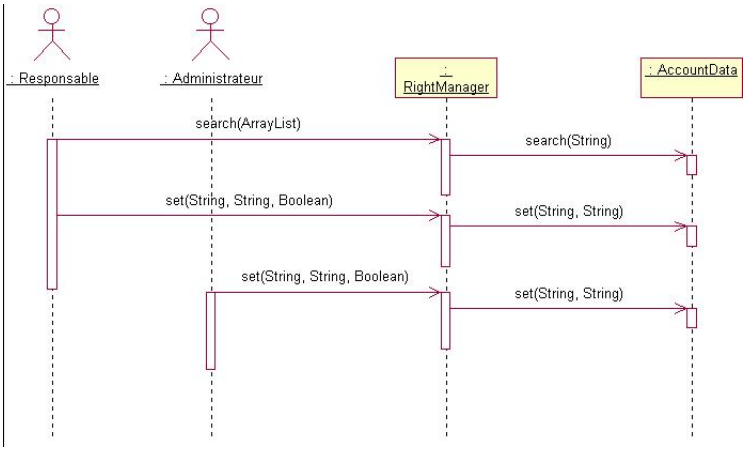
- AccountManager : C'est le gérant des comptes et c'est lui qui reçoit la requête d'un utilisateur qui veut se loger. Il utilise les paramètres du compte de l'utilisateur pour vérifier la validité de ces derniers avant de le connecter à l'application. Cette vérification se fait par délégation de la requête du consultant à l'objet AccountData.

- AccountData : C'est l'objet qui stocke toutes les informations sur les comptes des utilisateurs.C'est lui qui vérifie la validité d'un compte.

- RigthManager : C'est un objet qui permet à l'administrateur ou au responsable de pouvoir rechercher ou modifier les droits à un enseignant. Les droits sont stockés dans AccountData.



Voici le diagramme de séquence de compte.



Voici le diagramme de séquence de droit.

1.2 Diagramme d'enseignement/section/matière

Voici, le diagramme de classes du paquetage d'enseignement :

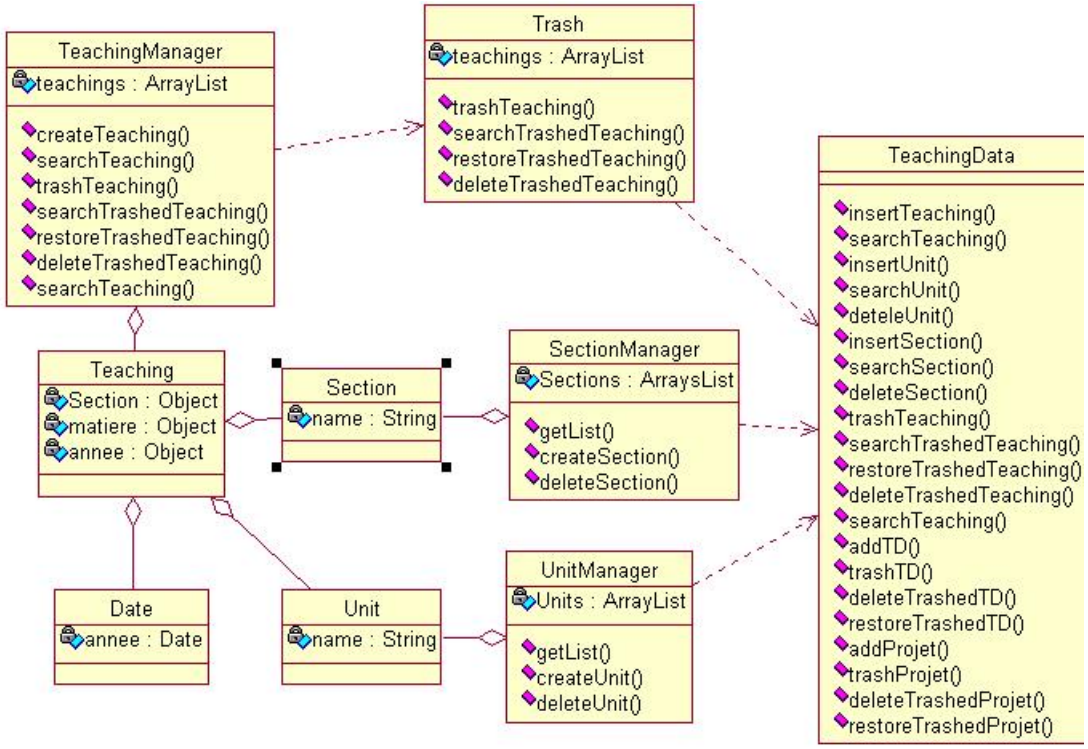


Diagramme de classe

On présente ici la gestion des enseignements. Elle regroupe également les paquetages de la section et de matière.

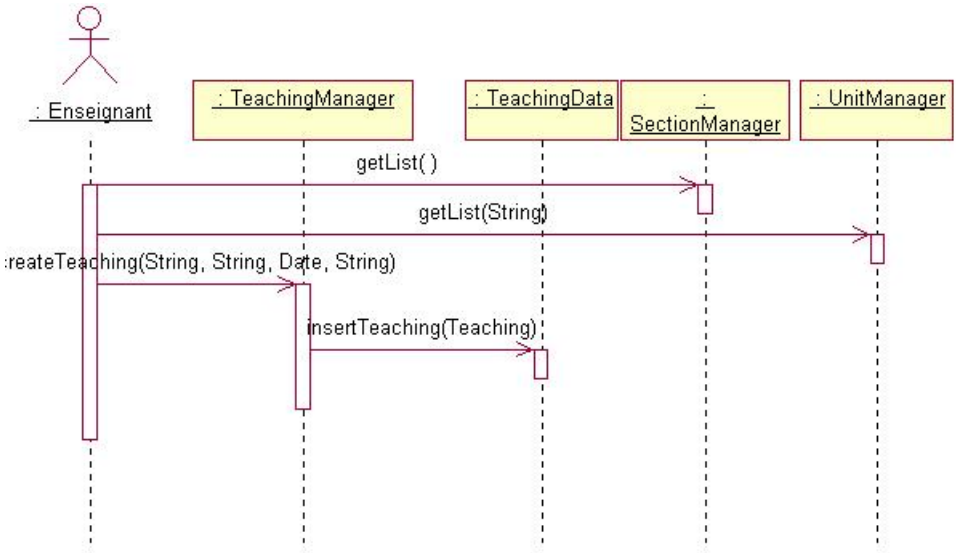
- TeachingManager : C'est l'objet qui gère les enseignements, il s'occupe de leur création, recherche, mise dans la corbeille, restauration et suppression.

- TeachingData : Il s'occupe du stockage des enseignements dans leur intégralité. Le teachingManager lui fait appel pour toutes les opérations.

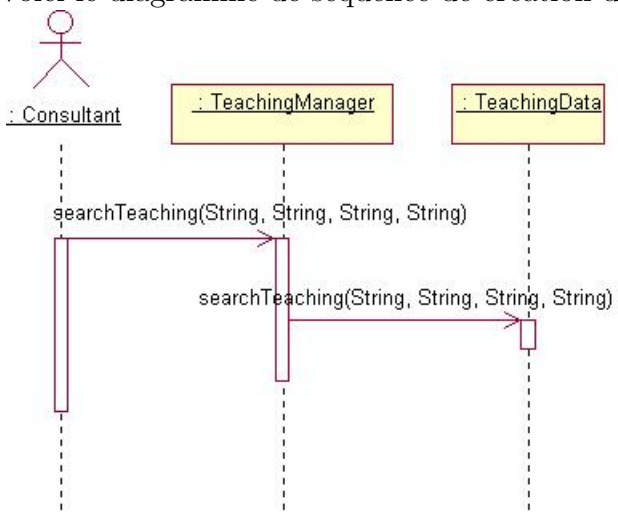
- Trash : C'est l'objet qui collecte tous les enseignements mis à la poubelle pour leur créateurs. Il les conserve jusqu'a leur suppression.

- SectionManager : Il gère l'ensemble des sections et intéragit avec le TeachingManager lors de la création ou de la recherche.

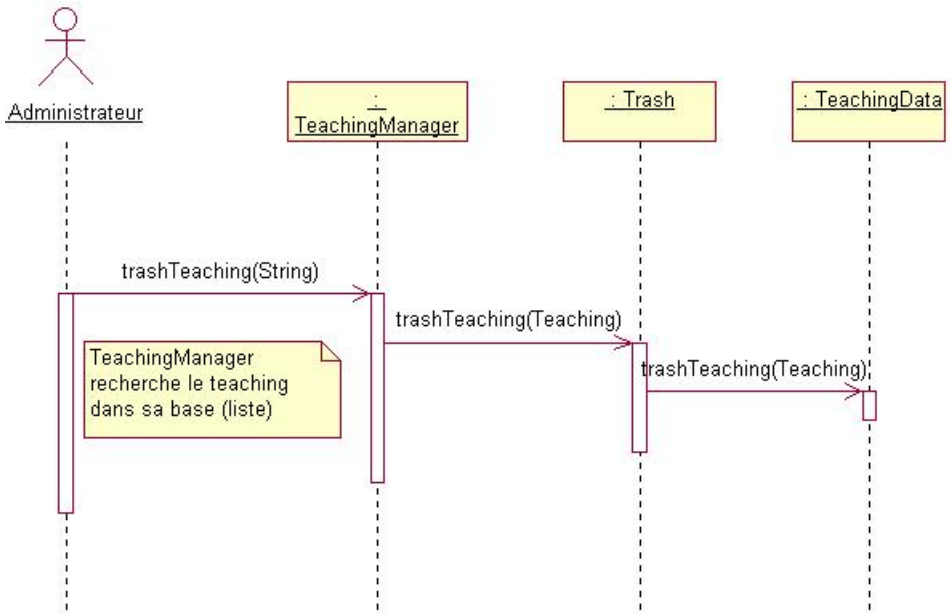
- UnitManager : Il s'occupe des matière et intéragit avec le TeachingManager lors de la création ou de la recherche.



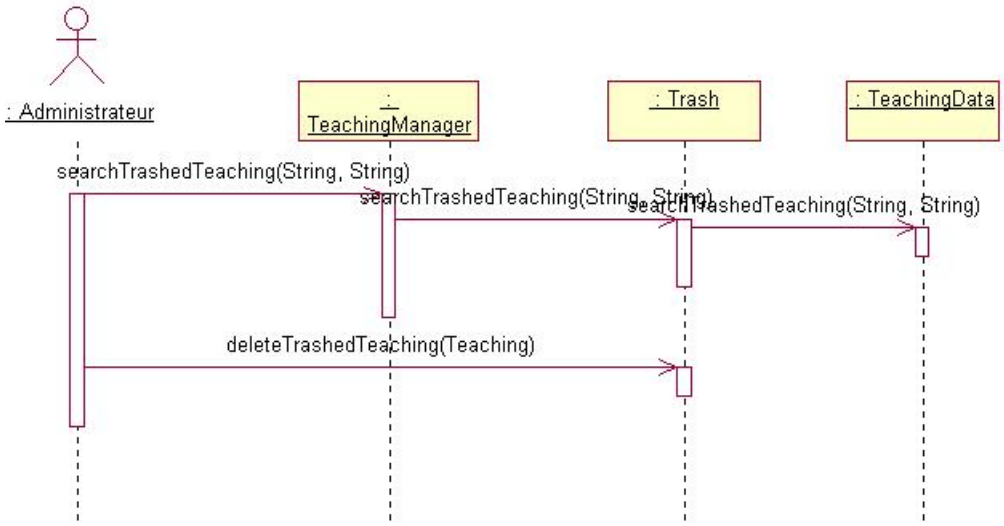
Voici le diagramme de séquence de création d'enseignement.



Voici le diagramme de séquence de recherche d'enseignement.



Voici le diagramme de séquence de mise à la corbeille d'enseignement.



Voici le diagramme de séquence de la suppression d'enseignement.

1.3 Diagramme des exercices

Nous présentons ici la gestion des exercices.

Pour commencer, voyons un aperçu des différentes classes entrants en jeu :

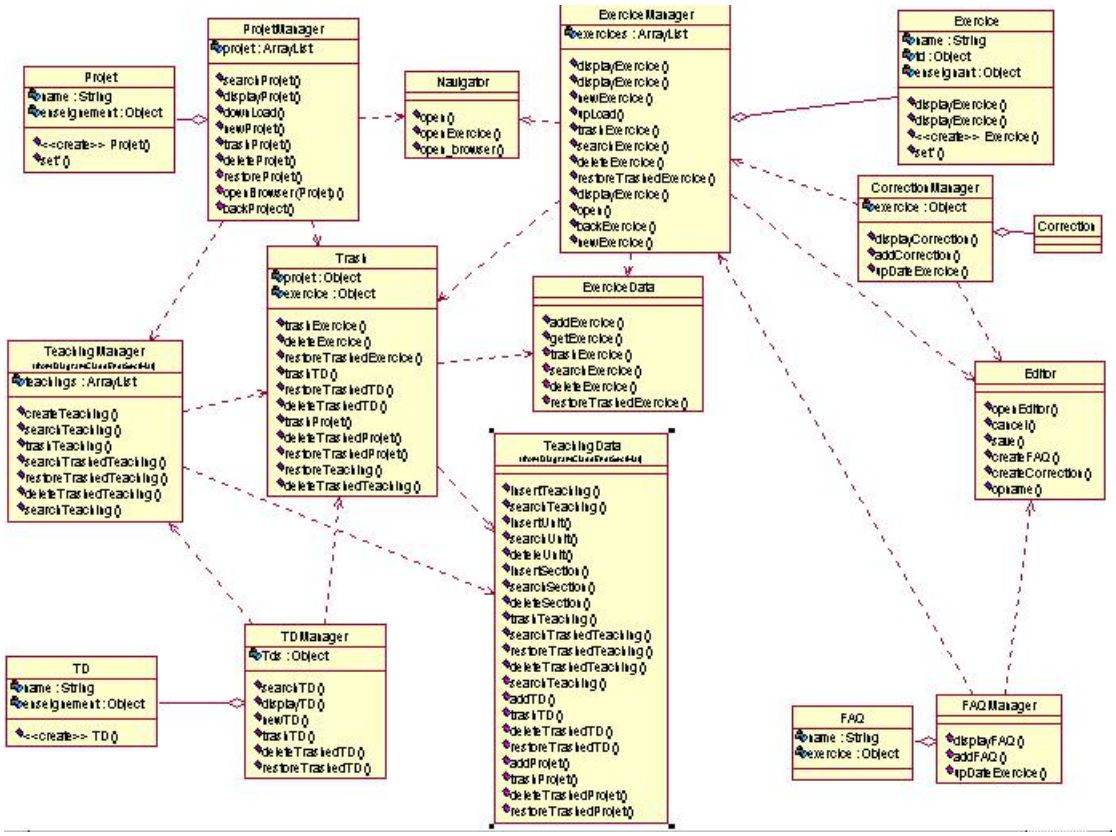


Diagramme de classes.

– Exercice : Représente un objet contenant les données d'un exercice. Après sa création, il est stocké dans ExerciceManager.

– ExerciceManager : C'est l'objet qui gère les exercices, il s'occupe de leur création, recherche.

– ExerciceData : Il gére le stockage des exercices dans leur intégralité. Le ExerciceManager lui fait appel pour toutes les opérations.

– Trash : C'est l'objet qui collecte tous les exercices mis à la poubelle par leur créateurs. Il les conserve jusqu'a leur suppression définitive.

– FAQManager : Il gère l'ensemble des Foire Aux Questions et intéragit avec le ExerciceManager pour la création ou la suppression de FAQ dans un exercice donné.

– CorrectionManager : Il gère l'ensemble des corrections et intéragit avec le ExerciceManager pour la création ou la suppression de correction dans un exercice donné.

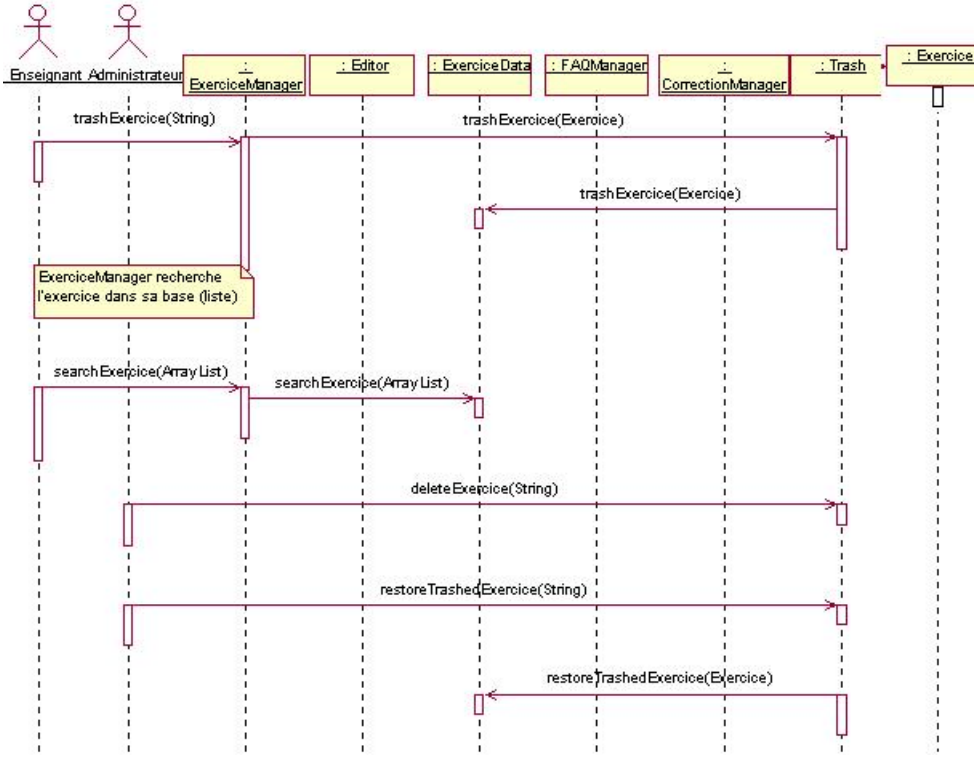


Diagramme de séquence de la recherche et de la suppression des exercices

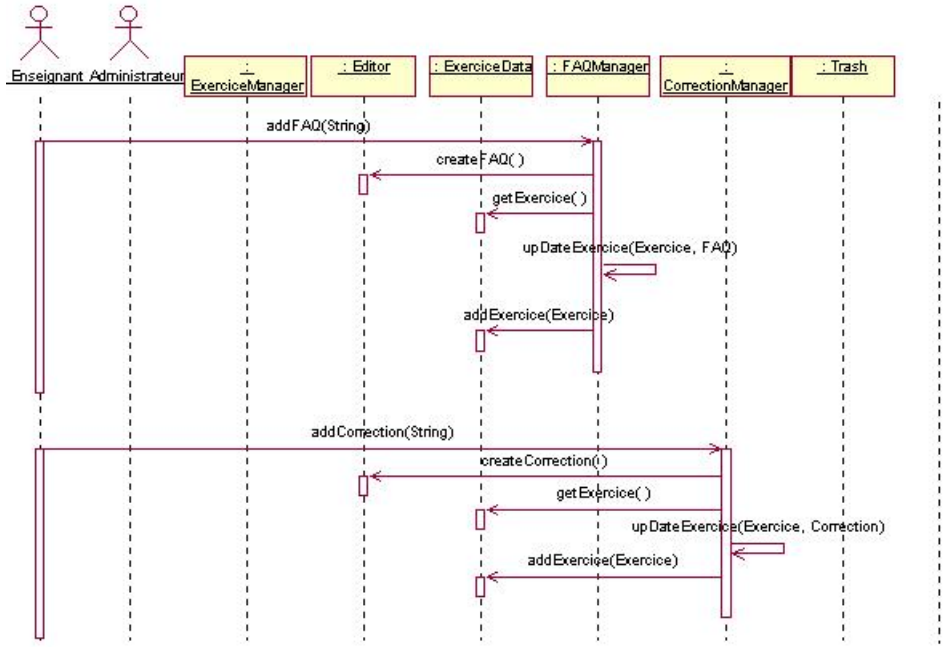


Diagramme de séquence de la FAQ et de la correction

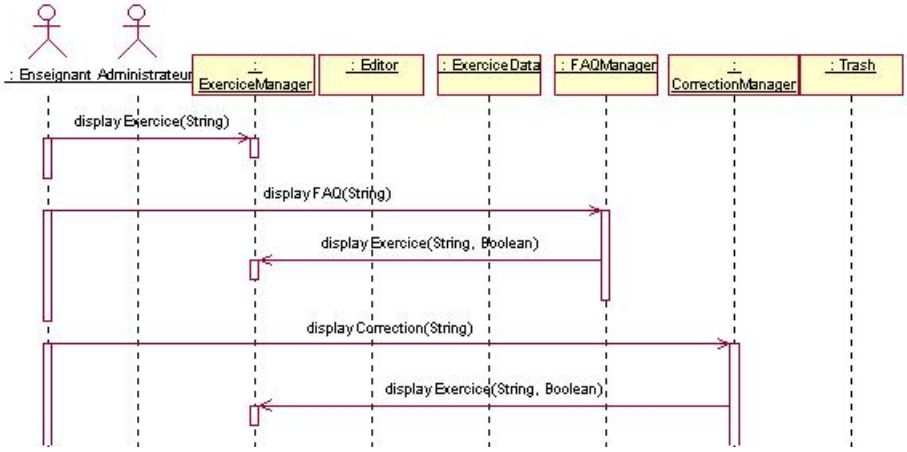


Diagramme de séquence de l'affichage

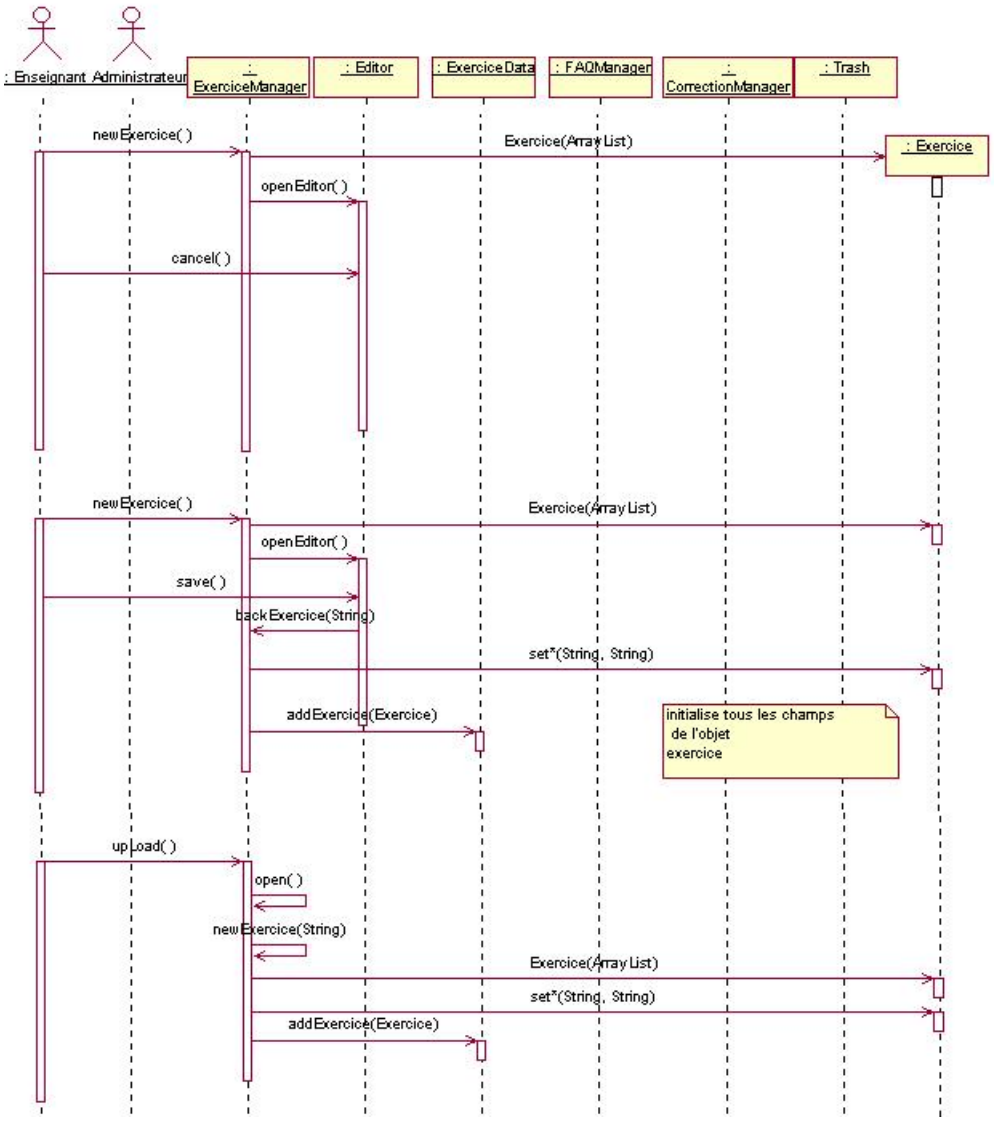


Diagramme de séquence de la création.

1.4 Diagramme des projets

Nous présentons ici la gestion des projets.

- Projet : Représente un objet contenant les données d'un projet. Après sa création, il est stocké dans ProjetManager.

- ProjetManager : C'est l'objet qui gère les projets, il s'occupe de leur création, recherche.

- TeachingData : Il gére le stockage des projets dans leur intégralité. Le ProjetManager lui fait appel pour toutes les opérations.

- Trash : C'est l'objet qui collecte tous les projets mis à la poubelle par leur créateurs. Il les conserve jusqu'a leur suppression définitive.

- TeachingManager : Il gère l'ensemble des projets pour la création ou la suppression.

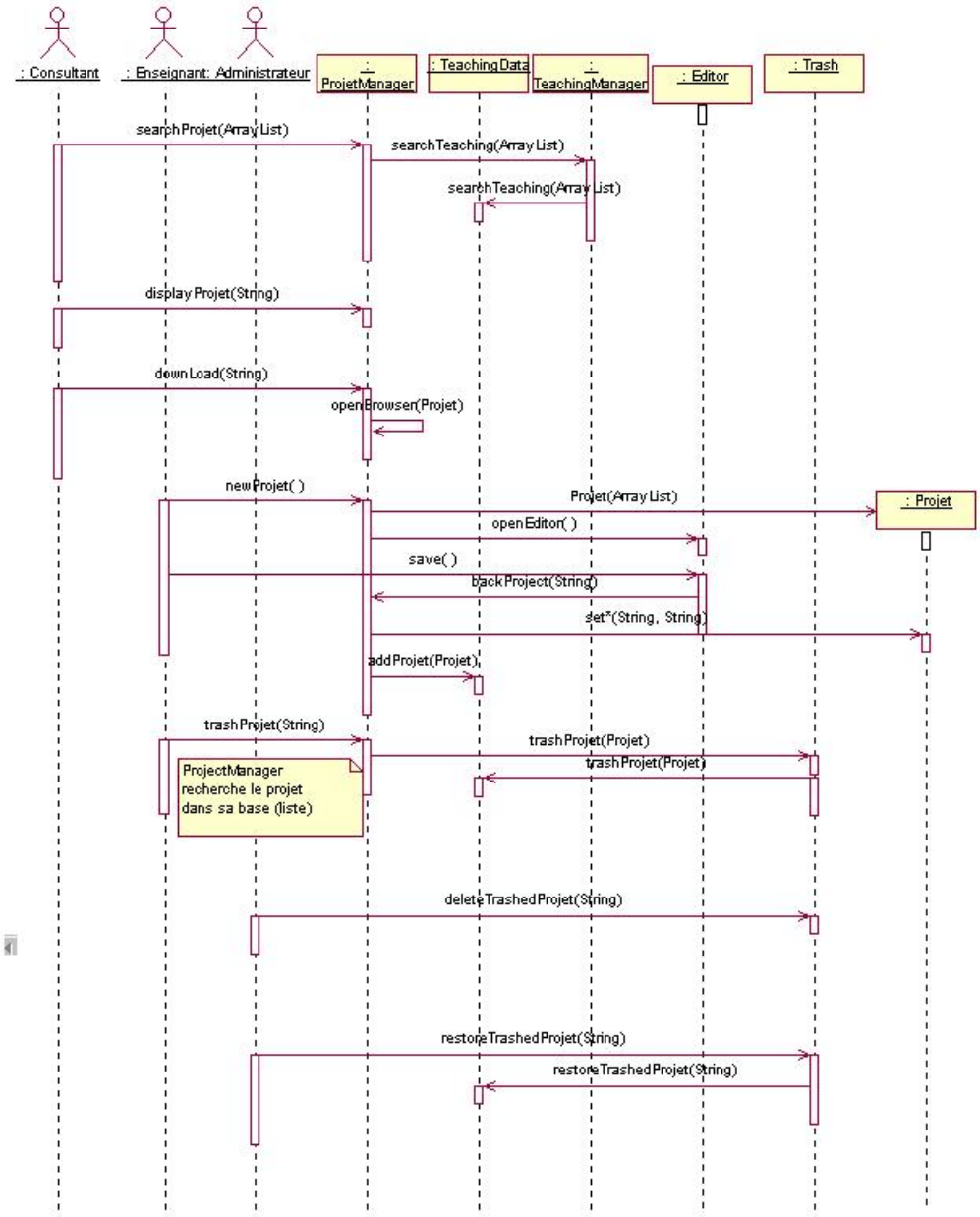


Diagramme de séquence des projets

1.5 Diagramme des TD

Nous présentons ici la gestion des td.

- TD : Représente un objet contenant les données d'un TD. Après sa création, il est stocké dans TDManager.
- TDManager : C'est l'objet qui gère les td, il s'occupe de leur création, recherche.
- TeachingData : Il gére le stockage des TD dans leur intégralité. Le TDManager lui fait appel pour toutes les opérations.
- Trash : C'est l'objet qui collecte tous les TD mis à la poubelle par leur créateurs. Il les conserve jusqu'a leur suppression définitive.
- TeachingManager : Il gère l'ensemble des TD pour la création ou la suppression.

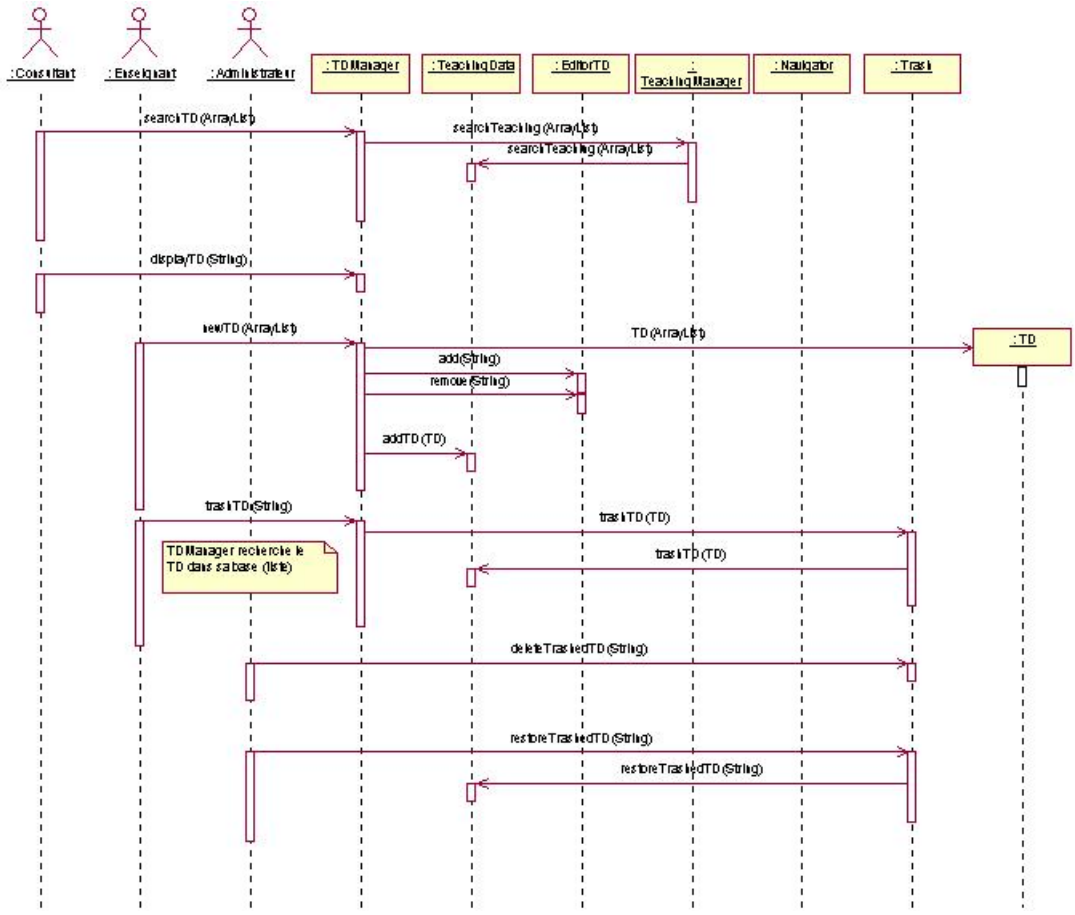


Diagramme de séquence des TD

Chapitre 2

Bibliothèques et logiciels utilisés

2.1 Bibliothèques

Voici un tableau récapitulatif des différentes bibliothèques dont nous aurons besoin :

Outil	**Bibliothèques**
Tomcat 4.1.18	org.apache.catalina
JUnit	junit
Ant	org.apache.tools.ant
Strusts	org.apache.struts
Midas	...
XML/XSLT	javax.xml/org.apache.xalan.xslt
FOP	Déjà intégré à partir de la jdk 1.2
SAX	org.xml.sax
JDBC	java.sql

Voici maintenant la spécification des différents outils, à commencer par TOMCAT.

2.2 Tomcat

TOMCAT est un conteneur de servlet.
Le conteneur de servlet associe à une URL virtuelle une servlet. TOMCAT intégre les servlets et les JSP. Nous avons choisi d'utiliser TOMCAT car il respecte une norme concernant les technologies Java Servlet et JavaServer Pages. Voyons ce que représente une servlet.

2.2.1 Servlet

Une servlet est un programme Java qui réside et s'exécute sur un serveur pour apporter des fonctionnalités ou traiter des données sur le serveur.
Son exécution n'est pas visible du côté client, elle est transparente. En effet, les données sont traduites avant d'être envoyées vers le client.
Cela rend plus flexible notre travail, nous pouvons modifier des traitements sans le signaler au client.
Le seul inconvénient des servlets est le temps nécessaire pour la mise en forme des pages html. C'est pourquoi, nous nous proposons d'utiliser les JSP.

2.2.2 Java Server Pages (JSP)

Le JSP est une technologie pour controller le contenu ou l'apparence de pages web à travers les servlets (définie précédemment). Parfois, le JSP est représenté comme l'interface des programmes d'application servlets. Il permet d'entrelacer du code HTML avec du code Java. Malgré le fait que les JSP paraissent différentes des servlets, Java traduit les JSP en servlets. Ce qui signifie que les JSP sont en fait juste une représentation plus aisé des servlets, ce qui facilite la programmation.

Dans notre application, nous allons utiliser le JSP pour l'affichage en HTML et les servlets pour les traitements. Donc, nos programmes JSP vont faire appel à des servlets.

En ce qui concerne les tests, nous nous proposons d'utiliser le JUnit.

2.3 JUnit

Le JUnit est une structure permettant de faire du code sûr, fiable, qui marchera dans toutes les situations et surtout dans le temps.

Les inconvénients sont qu'au départ, le code sera un peu long. En effet, en plus du code pour l'application, il faudra rajouter du code pour tester le code de l'application.

Les avantages sont mutliples. Le JUnit est un façon de regrouper les expressions tels que *print*, les debuggers et les scripts de tests en des objets permettant de garantir le code. Le fait que les tests soient des objets garantit une lisibilité du code. En effet, cela évite l'utilisation d'écriture sur la sortie standard avec l'appel de *print*. Cela va permettre à nos développeurs d'éviter de faire trop d' efforts supplémentaires quant aux erreurs. JUnit permet également à des personnes extérieures qui n'ont pas codé les sources d'interprété les réultats générés.
JUnit distingue les échecs des erreurs. En effet, un échec peut être anticipé alors

qu'une erreur ne peut pas.

En ce qui concerne la compilation, nous nous proposons d'utiliser un makefile Java : **Antidote**.

2.4 Antidote (Ant)

ANT est au monde Java ce que MAKE est au monde du C. ANT va bien au delà d'un simple apport technique. ANT exige une réelle démarche méthodologique autour de l'organisation des fichiers et de leur devenir dans les différents processus de livraison (avec des problémes tels que la séparation sources-livrables ou la gestion multi-environnements/multi-OS). Il possède tous les atouts propres aux standards : simple, bâti sur des technologies ouvertes (Java et XML), extensible, et supporté par des acteurs stratégiques.

Pourquoi ANT ?
Selon les domaines d'application, il existe différents moyens d'automatiser des traitements batchs : fichiers batchs, outil MAKE, programmes d'ordonnancements, installateurs voire des programmes développés spécifiquement. ANT est efficace dans le sens où il peut prendre en charge de nombreuses problématiques mais s'il est performant dans certains domaines (eg. : construction d'application), d'autres produits présentent une meilleurs qualité de service dans d'autres domaines (eg. : installation de logiciels). **L'inconvénient** est l'absence d'interface graphique constitue néanmoins un frein important pour bon nombre d'utilisateurs. ANT n'en propose pas plus pour l'administration que pour le suivi de l'exécution. Cependant, la plupart des IDE Java disposent de plugins dédiés à l'utilisation de ANT, avec leur propre interface graphique : JBuilder, Eclipse/WSAD, NetBeans/Forte, Intellij IDEA, ...

Les avantages sont mutliples.

- Il est **portable** : implémenté en Java et configurable via XML, ANT fonctionne de la même façon sous n'importe quel OS supportant Java et il permet également d'effectuer des traitements spécifiques selon l'OS.
- Il est **extensible** : ANT est conçu de telle manière que l'intégration de nouvelles tâches (interfaçage avec un produit, opération utilitaire, ...) soit un jeu d'enfants.
- Il est **simple** : La configuration XML est simple, tout comme les concepts de base. Il est parfois plus facile d'utiliser certains outils via ANT (eg. : javadoc ou javac) du fait de l'homogénéité des tâches ANT qui les interfacent, que ce soit au niveau de leur syntaxe ou de leur documentation.
- Il est **configurable via XML** : il permet à la fois de s'affranchir de l'apprentissage d'un nouveau format propriétaire et de bénéficier des nombreux moyens de traitement disponibles autour d'XML.
- Il est **OpenSource** : Contrairement à d'autres projets OpenSource, ANT n'induit aucun coût caché (assistance technique, documentation payante, ...).

La licence Apache protégeant ANT est très permissive, elle permet notamment de commercialiser un produit bâti sur ANT.

Concernant le développement, nous allons développer selon le concept *Modèle-Vue-Controleur*.

2.5 Strusts

Stuts va nous permettre de développer un *modèle-vue-controleur* de manière facile.

Qu'est ce que c'est ?

Struts est un cadre de travail (framework) open source développé pour encourager une architecture applicative basée sur le modèle conceptuel Model-View-Controller (MVC), très utile dans la construction d'applications Web reposant sur la technologie des Servlets Java et des Pages Serveur Java (Java Server Pages - JSP). Il est compatible avec la plate-forme J2EE de Sun et principalement basé sur la technologie des servlets et du JSP. Le package Struts, même s'il n'en est qu'à sa révision 1.0, comporte une documentation complète. Vous y trouverez un guide utilisateur tout comme des guides pour développeurs.

Le framework en tant que tel peut être divisé en quatre parties principales, trois d'entre eux correspondant merveilleusement au modèle MVC :

- Le modèle, qui est un classe Action (dont nous parlerons dans une minute), subvient à la logique business écrite par le développeur. La distribution effectuée par le contrôleur à la classe Action est basée sur une configuration qui est prodiguée par le fichier struts-config.xml.
- La vue, qui est un ensemble de librairies JSP de balises personnalisées, travaille en concert avec la servlet du contrôleur et cela permet la création rapide de formulaires pour une application.
- Le contrôleur,qui est une servlet, distribue les requêtes reçues à la classe Action appropriée.
- Un nombre de classes utilitaires supportent l'analyse XML, l'enrichissement automatique des propriétés des JavaBean et l'internationnalisation des messages affichés.

Le contrôleur est vraiment l'agent de la circulation du framework et c'est lui qui détermine à quel moment les choses arrivent. La servlet contrôleur est responsable du packaging et du routage des requêtes HTTP vers l'objet approprié du framework. Ce rôle est pris en charge soit par une page JSP, soit par une classe Action. Le modèle est un objet qui prend la requête d'un utilisateur et stocke les résultats pour toute la durée du process. L'objet vue est souvent une page JSP. Le framework Struts ne fournit pas actuellement de JSP mais il fournit par contre de nombreuses

librairies de balises qui permettent une intégration aisée de JSP dans le framework. L'interaction de Struts avec JSP permet le stockage de données d'un formulaire de saisie dans un bean formulaire.

Les avantages sont :
- il est extrêmement stable.
- économie de temps pour coder un Modèle-Vue-Controleur complet

Les désavantages sont :
- compte tenu que c'est un projet à grande échelle, il y a beaucoup d'écrans et de types de requêtes différents. De ce fait, il y a beaucoup de classes Action puisque nous en avons besoin d'une seule par type de requêtes.
- Besoin d'apprentissage de cet outils.
- Necessite une bonne connaissance du modèle-vue-controlleur.

Concernant la saisi en mode texte, nous proposons d'utiliser un éditeur *de bord* : Mozilla Midas.

2.6 Mozilla Midas

Midas va nous permettre d'éditer des documents en ligne à partir du navigateur web de l'utilisateur.

C'est un éditeur de fichier texte implanté de base dans le navigateur mozilla.

Les avantages sont :
- il est fourni de base dans mozilla.
- il possède une large collection de méthodes.
- il est simple d'emploi.

Nous avons besoin de des feuilles de style XSLT pour respecter les fonctionnalités du projet.

2.7 XML et XSLT

Le **XML** sera utilisé comme une base de donnée pour les exercices, les projets, les TD, et toutes les données qui nécéssiteront d'être stockées. Les exercices seront également formatté avec un balisage XML pour permettre l'application de feuille de style. Elles permettront de mettre en forme le rendu final des exercices à destination de l'utilisateur.

Interêt pour le projet

XML (Extensible Markup Language, ou Langage Extensible de Balisage) est le langage destiné à succéder à HTML sur le World Wide Web. Comme HTML

(Hypertext Markup Language) c'est un langage de balisage (markup), c'est-à-dire un langage qui présente de l'information encadrée par des balises. Mais contrairement à HTML, qui présente un jeu limité de balises orientées présentation (titre, paragraphe, image, lien hypertexte, etc.), XML est un métalangage, qui va permettre d'inventer à volonté de nouvelles balises pour isoler toutes les informations élémentaires (titre d'ouvrage, prix d'article, numéro de sécurité sociale, référence de pièce ?), ou agrégats d'informations élémentaires, que peut contenir une page Web.

Inconvenients : Le stockage sous format XML est dans la pratique moins rapide qu'une base de donnee standard.Une bonne structuration des fichiers constituant alors notre base est nécéssaire pour faciliter la recherche et ne pas la rendre trop lente.

Avantages : L'application de feuille de style est possible avec XML. Elle nous permettra de générer les TDs dans un grand nombre de format.

Le **XSLT** (eXtensible Stylesheet Language Transformation) permet de transformer des documents XML à l'aide de feuille de style contenant les règles appelées *template rules*.
Le processeur XSLT crée une structure logique arborescente pour produire un arbre résultat représentant la structure d'un document HTML. La méthode directe consiste à lier le document XML ç la feuille de style XSL par l'intermédaire d'une *processing instruction*.

Pour parvenir a nos fins, nous avons besoin des feuilles de styles telle que XSL/FO.

2.8 FOP

Interêt pour le projet : Le projet FOP est intégré au projet XML Apache.c'est un processeur XSL (à ne pas confondre avec XSLT) acceptant en entrée un fichier XML comportant des *Formatting Objects* pour produire en sortie un document au format PDF. Cet outil est idéal pour des applications nécessitant des besoins en impression limités en volume et en complexité : factures, contrats, bons de commandes... Le langage XSL/FO permet de formatter l'affichage ou l'impression d'un document XML (il s'agit de l'équivalent des style-sheet CSS)

Avec FOP, on dispose d'un jeu de librairie qui vont nous permettre d'appliquer des feuilles de rendu sur des fichiers XML. Ces APIs vont nous servir à générer nos TDs au format PDF.

Inconvénients : L'inconvénient général est que toute les fonctionnalités ne sont pas toujours présentes ou finies. Le langage XSL/FO n'est pas standardisé.

Avantages : Il à l'avantage d'être gratuit et efficace. Il permet de faire une conversion en fichiers affichables sur quelque support que ce soit (par exemple du format XML vers le format PDF). Il permet une bonne qualité d'impression autant sur le navigateur que sur l'imprimante.

2.9 Les parseurs

Nous utilisons des fichiers au format XML pour le stockage des données.

Lors de la parcours des fichiers XML nous avons besoin d'une interface d'analyseur qui permet de parcourir les fichiers et de trouver les éléments recherchés rapidement.

Les parseurs sont des logiciels qui permettent de lire des documents XML. Cette lecture peut être faite avec un contrôle du document par rapport à son schéma ou à sa DTD. Dans ce cas, le parseur est dit " validant ".

Deux types d'interfaces sont définis entre un parseur et l'application qui l'utilise :

- l'interface SAX (Simple Access to XML) qui enchaîne un ensemble de rétro appels vers l'application au fur et à mesure que les objets XML sont traités (balise de début, attribut, texte, balise de fin, ?) ;
- l'interface DOM (Document Object Model) qui construit une hiérarchie d'objets représentant le document en mémoire (document, éléments, attributs, contenus...).

Le traitement SAX est intégralement réalisé en un seul passage. Par conséquent, SAX est généralement plus performant que DOM pour l'analyse de documents d'une taille équivalente, puisque ce dernier doit passer en revue plusieurs fois l'arborescence. De plus, comme une partie seulement d'un document XML doit se trouver en mémoire à un instant donné, SAX est généralement plus efficace que DOM en termes d'utilisation de la mémoire avec des documents volumineux : Ce problème est spécifique des structures arborescentes du DOM : les grands arbres exigent plus de mémoire et DOM doit charger la totalité du document en mémoire avant que l'analyse puisse être lancée..
Côté **inconvénients**, les applications SAX présentent souvent des instructions *if/else* longues et complexes pour déterminer quelle action entreprendre lorsqu'un élément particulier est traité. De même, le traitement de structures de données réparties entre plusieurs éléments XML s'avère difficile avec SAX puisque des données intermédiaires doivent être stockées entre les événements d'analyse. Enfin, la structure de gestion des événements d'une application SAX implique généralement que les applications SAX doivent être personnalisées pour une structure de document spécifique, alors que les applications DOM peuvent rester beaucoup plus générales.

Ceci explique pourquoi nous avons décidé d'utiliser le parseur SAX.

2.10 Java DataBase Connection (JDBC)

En ce qui concerne l'identification des utilisateurs, nous avons besoin d'une connection permettant de manipuler les bases de données du client.
JDBC a été conçu pour garder simples, les choses simples. Cela veut dire que l'API JDBC fait des tâches quotidiennes sur les bases de données (comme les requêtes SELECT...) quelque chose de très facile. Il permet de faire la relation entre un programme et une base.

Pour la facilité de programmation nous allons utiliser CVS.

2.11 Concurrent Versions System (CVS)

CVS est un outil de maintien et de suivi de versions pour travailler à plusieurs sur un projet informatique. Il introduit la notion de groupe de travail et permet de gérer l'arborescence de fichiers d'un projet donné. Il permet de travailler en local sur l'arbre du projet, puis de notifier les modifications sur l'arbre commun. Il gère l'historique des versions. Il gère dans une certaine mesure les conflits de versions dus à des modifications concurrentes.

C'est pour cela que nous allons nous servir de cet outils, pour simplifier la programmation.
Nous avons trouvé un serveur sur internet (berlios.de) qui nous fournit un serveur CVS, un forum, un logiciel de gestion des bugs, un espace site web.

Chapitre 3

Contacts

Pour toutes questions complémentaires veuillez vous adresser à l'une des personnes suivantes :

Fonction	Nom	Email
Chef de projet	Batoussa MOUGAMADOU	bat_fr@hotmail.com
Responsable qualité	Saliou NGOM	sngom@mailcity.com
Responsable gestion des fichiers	Vincent BOISSIN	vboissin@etudiant.univ-mlv
Administrateur CVS	Gabriel DUPONT	gdupont@etudiant.univ-mlv
Responsable XML	Julien KIRSCH	jkirsch@etudiant.univ-mlv.f
Développeur	Frederic GARCIA	fgarci01@etudiant.univ-mlv.
Maître d'ouvrage	Remi FORAX	forax@univ-mlv.fr