C-SAM

Correction Semi-Automatique Modulable

Encadrants : Sandra BRINGAY – Stéphanie LEON – Alexandre PINLOU

Etudiants : Luc BARRY HARIVELO – Mehdi BENGHABRIT –

Frédéric RAFFIN-CABOISSE – Julien TISER

2008-2009

TER M1

Responsable : Michel Leclere

Rapport sur le projet de première année de MASTER Informatique à l'Université de Montpellier II :

Sujet : création d'un logiciel de correction semi-automatique de copies du C2i.

Auteurs :

Luc BARRY HARIVELO

Mail : luc_barry@yahoo.fr

Mehdi BENGHABRIT

Mail : benghabrit.mehdi@gmail.com

Frédéric RAFFIN-CABOISSE

Mail : raffin_caboisse.frederic@yahoo.fr

Julien TISER

Mail : julien.tiser@gmail.com

Tuteurs :

Mlle **Sandra BRINGAY** : bringay@lirmm.fr

Mlle **Stéphanie LEON** : leon@lirmm.fr

M. **Alexandre PINLOU** : alexandre.pinlou@lirmm.fr

Remerciements

Ce projet a été une expérience très enrichissante pour nous, autant sur le plan personnel que pour notre formation d'informaticien.

Nous tenons tout d'abord à remercier nos trois tuteurs, Mlle Sandra Bringay, Mlle Stéphanie Léon, et M. Alexandre Pinlou, pour nous avoir encadrés et appuyés tout au long de ce projet.

Nous remercions également Mme Anne-Lise Gros d'avoir mis à notre disposition des outils pour la création et la mise en place de notre logiciel.

Nous exprimons notre gratitude à l'ensemble des enseignants du département informatique de l'Université Montpellier II pour nous avoir transmis le savoir qui nous a permis de mener à bien ce projet.

Sommaire

Liste des figures

Introduction

Dans le cadre de notre première année de MASTER informatique, nous avons créé un logiciel de correction semi-automatique de copies du C2I (Certificat Informatique et Internet) pour l'Université Montpellier III. Ce projet d'une durée de 4 mois nous a permis de mettre en pratique nos compétences acquises tout au long de cette formation.

Ce rapport décrit, dans un premier temps, les différentes phases du projet : l'analyse des besoins, l'analyse fonctionnelle, le développement ainsi que les résultats. S'en suivra ensuite dans un deuxième temps une discussion.

Contexte :

Le C2I[1], Certificat Informatique et Internet, atteste de compétences informatiques. Il a été créé par le gouvernement pour développer, renforcer et valider la maîtrise des technologies de l'information et de la communication par les étudiants en formation dans les établissements d'enseignement supérieur.

S'il n'est pas encore obligatoire, il est désormais fortement recommandé pour l'insertion professionnelle. Depuis quelques années maintenant, les universités proposent un accès à cette certification pour tous les nouveaux entrants. Á l'université de Montpellier III, cela se traduit par la correction de plus de 3000 copies électroniques à la fin de chaque semestre ! Afin de gérer ces importants volumes d'étudiants, il devient nécessaire de concevoir des outils et produits numériques pouvant être utilisés à des fins d'enseignement et d'apprentissage (produire, traiter, entreposer, échanger, classer, retrouver et lire des documents numériques).

Objectifs :

Le but principal de ce projet est de mettre en place un outil permettant à un seul enseignant de corriger la totalité des copies de façon semi-automatique :

[1] http://www2.c2i.education.fr/sections/accueil/c2i7120/

- par souci d'équité (le point de vue d'un seul enseignant pour toutes les copies).

- pour gagner du temps (automatisation de la plupart des cas similaires, une fois que le correcteur a décidé de la conduite à tenir (note) pour le premier cas rencontré).

Nous avons une tâche principale à réaliser et une facultative concernant respectivement les exercices suivants proposés aux étudiants :

- **Gestion de leur espace de travail** (tâche principale) : Durant l'évaluation, les étudiants reproduisent une arborescence (répertoire, sous-répertoires contenant des fichiers de différents formats). Nous proposerons un outil capable de noter les travaux des étudiants. Les critères permettant d'évaluer le travail d'un étudiant seront notamment, une bonne syntaxe des noms des documents, un bon format et un bon emplacement de ces derniers.
Si l'arborescence proposée présente un cas déjà recensé, alors l'outil attribue la note adéquate ; sinon il est nécessaire qu'un enseignant intervienne pour corriger et noter le travail. La correction qu'il aura effectuée sera utilisée les prochaines fois où un cas identique sera rencontré, le logiciel aura appris la note et la correction sera plus rapide.

- **Structuration de documents complexes** (tâche facultative, ex : notes de bas de pages, sommaire, index, styles...) : L'évaluation des étudiants consiste à produire un texte au format OpenOffice.org respectant différentes contraintes de style. Or, la suite bureautique OpenOffice.org est basée sur le langage XML. Ce langage de balisage générique permet d'extraire automatiquement un grand nombre de propriétés d'un document. L'outil que nous avons développé peut être modifié afin d'accepter un module permettant d'extraire les données pertinentes du document XML et de vérifier la mise en forme du document.

Il faut toutefois noter que la seconde tâche est optionnelle, la première étant l'élément principal de notre projet.

Notre objectif est de produire un outil générique et adaptable à différents exercices, se reposant sur :

- l'apprentissage des erreurs,
- l'intervention d'un correcteur humain pour associer une note à une erreur non encore répertoriée,
- l'automatisation de la correction lorsque l'on rencontre des erreurs déjà répertoriées.

Ce projet s'appuie donc sur le principe des techniques de RAPC (Raisonnement à Partir de Cas), où l'on résout les problèmes en retrouvant des cas similaires dans notre base de connaissance et en les adaptant aux nouveaux cas considérés.

Organisation du projet

Ce projet a été réalisé par quatre étudiants et encadré par trois enseignants-chercheurs de l'Université de Montpellier 3 : Sandra Bringay, Stéphanie Léon et Alexandre Pinlou.

Chacun des quatre étudiants a participé à l'analyse, à la modélisation, ainsi qu'à l'implémentation. Par ailleurs, les autres tâches du projet furent réparties ainsi :

Nom	Rôle(s)	Responsabilités
BENGHABRIT	Designer et développeur	Responsable de la partie graphique
BARRY HARIVELO	Développeur	Implantation de la BD Programmation
TISER	Développeur	Programmation
RAFFIN-CABOISSE	Chef de projet et développeur	Programmation Encadrement

Figure 1 : rôle et responsabilités de chaque étudiant au sein du projet

Une fois par semaine, une réunion avec les tuteurs avait lieu. De plus, chaque vendredi, nous nous réunissions pour un travail de groupe et une mise en commun de notre avancée personnelle.

Analyse

Nous nous sommes donc concentrés sur la correction de l'exercice d'une arborescence de fichiers.
Les étudiants doivent donc construire une arborescence de fichiers et de dossiers comme le décrit l'exemple ci-dessous :

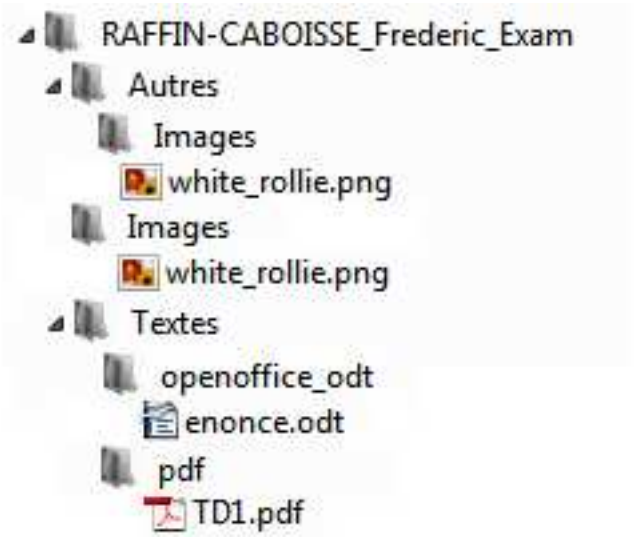


Figure 2 : exemple d'une arborescence demandée à un étudiant

Nous vérifions donc que les dossiers et fichiers soient bien nommés et bien placés. Lors de l'analyse des copies des étudiants, le correcteur doit faire face à plusieurs types d'erreurs et attribuer une note en fonction de celles-ci. Le tableau suivant expose les erreurs les plus fréquentes qu'il peut rencontrer lors de la correction :

Types d'erreurs	Description	Exemple
Casse	Erreur sur les majuscules ou minuscules.	« images » au lieu « d'Images »
Emplacement	Élément se trouvant au mauvais endroit dans l'arborescence.	Dossier « pdf » dans « Images »au lieu de« Textes »
Nom	Regroupe les fautes d'orthographe, les fautes de frappes, etc.	« Texte » au lieu de « Textes »
Tiret	Erreur sur le type de tiret.	« TISER-Julien-Exam » au lieu de « TISER_Julien_Exam »
Multiple	Composition de plusieurs types d'erreurs.	« texte » au lieu de « Textes »

Figure 3 : typologie des erreurs

A partir de ces différents cas d'erreurs, nous avons pu réfléchir plus précisément sur le principe de notre application. Nous avons donc des traitements spécifiques pour chaque type d'erreurs et selon si elles portent sur un dossier ou un répertoire.
Notre traitement des erreurs fonctionne donc selon le schéma ci-dessous.

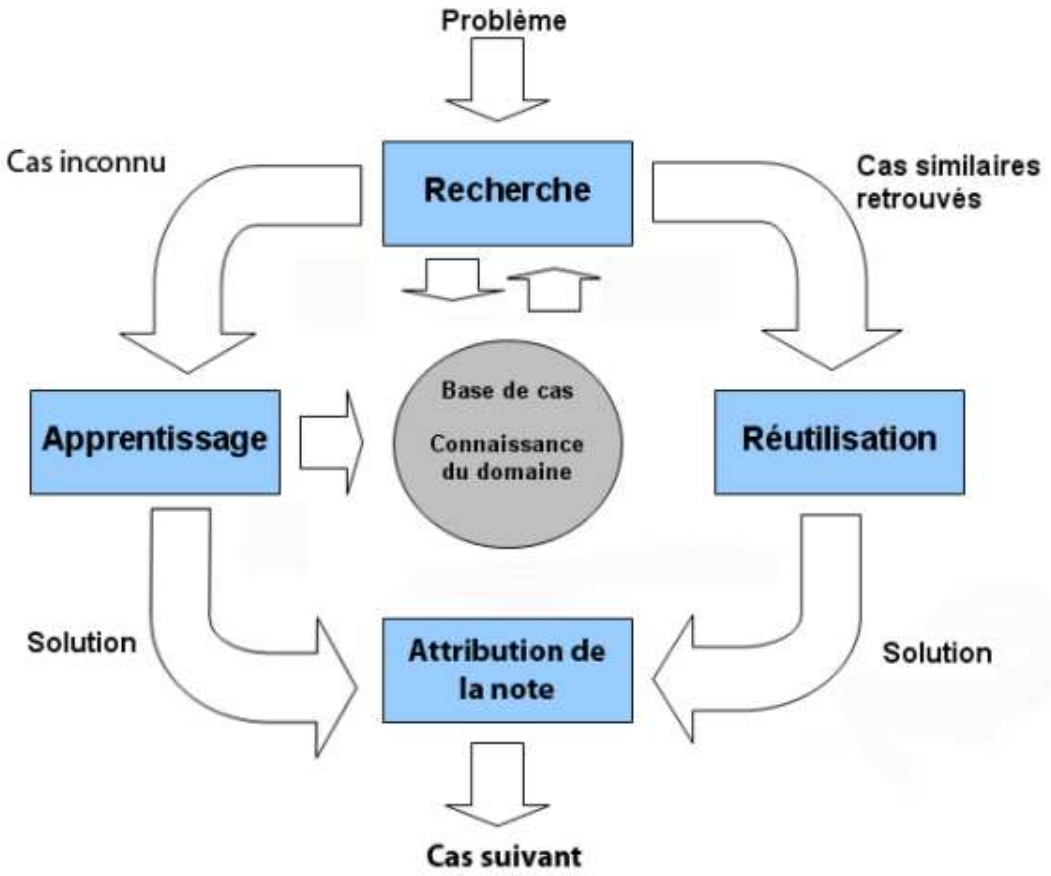


Figure 4 : raisonnement par cas

Pour ce qui est du fonctionnement global de notre application :

Le correcteur doit passer en entrée un énoncé appelé ici fichier de configuration, et un paquet de copies (sous forme d'archive zip). Ensuite il lance la correction et le traitement des erreurs vu ci-dessus est effectué. Enfin il accède aux notes et peut les exporter au format Excel à partir de la base de données.
Le schéma ci-dessous illustre ce fonctionnement.

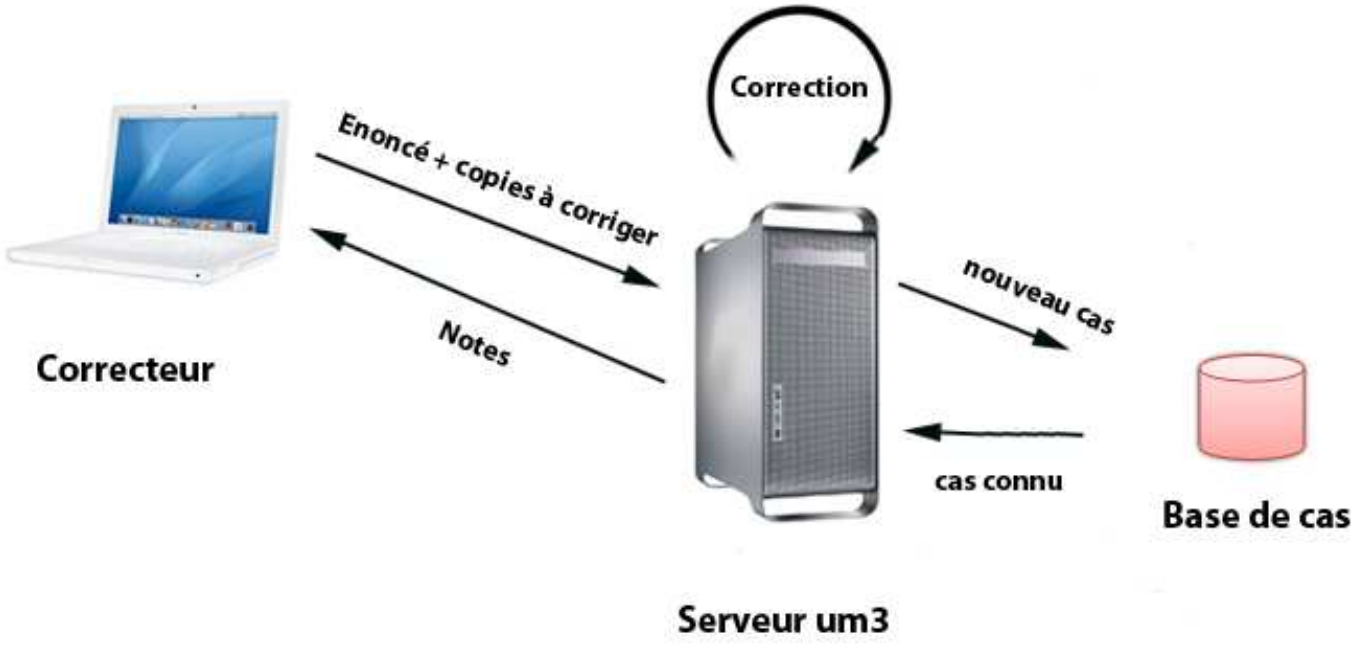


Figure 5 : Architecture globale du système

I. Spécification du fichier de configuration

Nous avons tout d'abord réfléchi à la manière de coder un énoncé, pour qu'il ne soit pas trop difficile à créer par l'utilisateur et à la fois assez détaillé pour que nous puissions traiter assez facilement tous les cas possibles.

Nous avons donc décidé d'écrire le fichier de configuration en XML de la manière suivante pour l'énoncé ci dessous vous avez le fichier de configuration XML qui correspond.

Ce fichier sera écrit par le correcteur et lui permettra de spécifier exactement ce qu'il attend et quel barème il applique à chaque question.

Enoncé :

A) Mise en route

1. Créez dans le dossier Etudiant un dossier intitulé NOM_Prenom_Exam (exemple : BOE_Jean-Marie_Exam).

2. Dans ce dossier, créez trois sous-dossiers intitulés "Images", "Textes" et "Autres". Dans le dossier "Textes", créez deux sous-dossiers intitulés "openoffice_odt" et "pdf".

3. Placez dans chacun de ces deux derniers sous-dossiers au moins un document du format correspondant (vous récupérerez ces documents par téléchargement à partir d'un site ou tout autre moyen).

4. Récupérez l'image des alpinistes sur la page : http://montagne.evasion.free.fr/sortsem/montblanc2003/montblanc.html et enregistrez-la dans le sous-dossier "Images".

5. L arborescence des dossiers de votre ordinateur peut être en partie décrite par le schéma suivant. Enregistrez une copie du fichier white_rollie.png dans le sous-dossier "Images".

6. Placez dans le dossier "Autres" une copie du dossier "Images".

Modélisation de l'énoncé :

```
<repertoire_racine nom="NOM_Prenom_Exam" fin_nom="Exam" question="0" bareme="1">
  <eval question="1" bareme="0.333333">
    <repertoire nom="Images" question="1" traiter="false">
      <eval question="7" bareme="1">
        <eval question="5" bareme="1">
          <fichier type_mime="null" nom="null" taille="" question="5" traiter="false"/>
        </eval>
        <eval question="6" bareme="1">
          <fichier type_mime="png" nom="white_rollie.png" question="6" traiter="false"/>
        </eval>
      </eval>
    </repertoire>
    <repertoire nom="Textes" question="1" traiter="false">
      <eval question="2" bareme="0.5">
        <repertoire nom="openoffice_odt" question="2" traiter="false">
          <eval question="3" bareme="1">
            <fichier type_mime="odt" question="3" nom="null" traiter="false"/>
          </eval>
        </repertoire>
        <repertoire nom="pdf" question="2" traiter="false">
          <eval question="4" bareme="1">
            <fichier type_mime="pdf" question="4" nom="null" traiter="false"/>
          </eval>
        </repertoire>
      </eval>
    </repertoire>
    <repertoire nom="Autres" question="1" traiter="false">
      <eval question="8" bareme="1">
        <copie type="repertoire" question="8" nom="Images" traiter="false"/>
      </eval>
    </repertoire>
  </eval>
</repertoire_racine>
```

Figure 6 : fichier XML de configuration

- La balise <repertoire_racine> est un cas particulier et n'apparaîtra qu'une seule fois car elle est spécifique à chaque étudiant, elle permet de préciser la manière dont sera créé le dossier racine de l'examen de l'étudiant. Les attributs de chaque balise permettent de mettre en place le contenu de la question.

- La balise <eval> indique au programme que c'est une question à part entière à laquelle correspond un barème.
- Les balises <repertoire> et <fichier> spécifient que l'on a affaire à une question portant sur respectivement un répertoire ou un fichier.
- La balise <copie> exprime la copie de fichier ou de dossier.

Toutes ces balises créées sont propres à l'exercice que nous traitons mais nous pourrions facilement ajouter de nouvelles balises afin de traiter d'autres types de cas.

II. Modélisation UML

La modélisation UML (Unified Modeling Language) nous a permis de représenter simplement notre problème et de le simuler. Elle comporte deux composantes :

- L'analyse, qui est l'étude du problème.
- La conception, soit la mise au point d'une solution à notre problème.

Nous avons utilisé plusieurs diagrammes pour modéliser notre application tout d'abord, d'une manière statique (diagramme de cas d'utilisation et diagramme de classe), c'est-à-dire représentant le système d'un point de vue physique, et par la suite de manière dynamique (diagramme de séquences et diagramme d'états) décrivant son fonctionnement.

Modèle statique :

- *Diagramme de cas d'utilisation* :

Les cas d'utilisation ou « USE CASES » nous ont permis de délimiter le système (ce qui est extérieur et qui communique avec le système, et ce qui est interne), les modèles descriptifs du point de vue des utilisateurs ainsi que les interactions avec les acteurs extérieurs.

Nous sommes partis sur l'analyse des besoins, avec les deux concepts :

- L'acteur, représentant l'entité extérieure à l'application et interagissant avec celui-ci.
- Les cas d'utilisation (interactions acteur-système) qui représentent toutes les manières d'utiliser le programme, suite d'évènements notable du point de vue de l'utilisateur.

Pour notre application, un seul acteur, le correcteur, interagit avec le système. Il aura la possibilité d'effectuer plusieurs actions qui, pour la plupart d'entre elles, sont liées par des relations d'utilisation « include » où le cas d'utilisation d'origine contient obligatoirement l'autre.

Le diagramme ci-dessous met en avant le fait que le correcteur doit impérativement s'identifier pour effectuer une quelconque action. Il décrit également qu'il devra charger le fichier de configuration correspondant au sujet avant de commencer la correction des copies et que l'arrêt « manuel » d'une correction nécessite au préalable le lancement d'une correction.

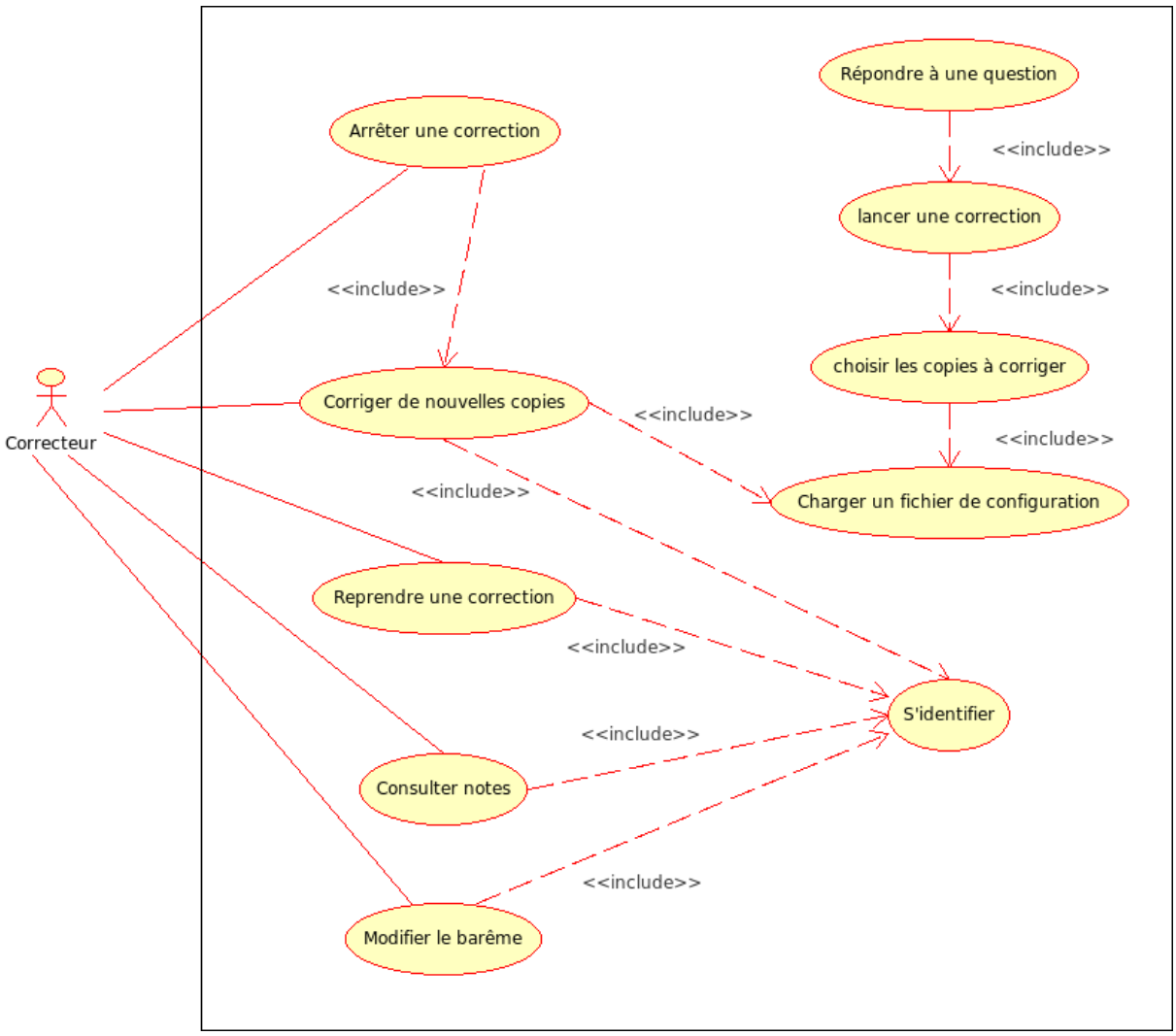


Figure 7 : diagramme de cas d'utilisation

Le correcteur aura également la possibilité de consulter les notes lorsque les corrections seront terminées, de modifier le barème des questions ainsi que de reprendre une correction dans le cas où il aurait quitté le programme avant la fin d'une correction.

- *Diagramme de classes :*

Le diagramme de classe décrit la totalité de notre application d'une manière abstraite, en termes de classes.

Il nous a permis de bien comprendre les contraintes posées par notre application et de pouvoir implémenter notre base de données.

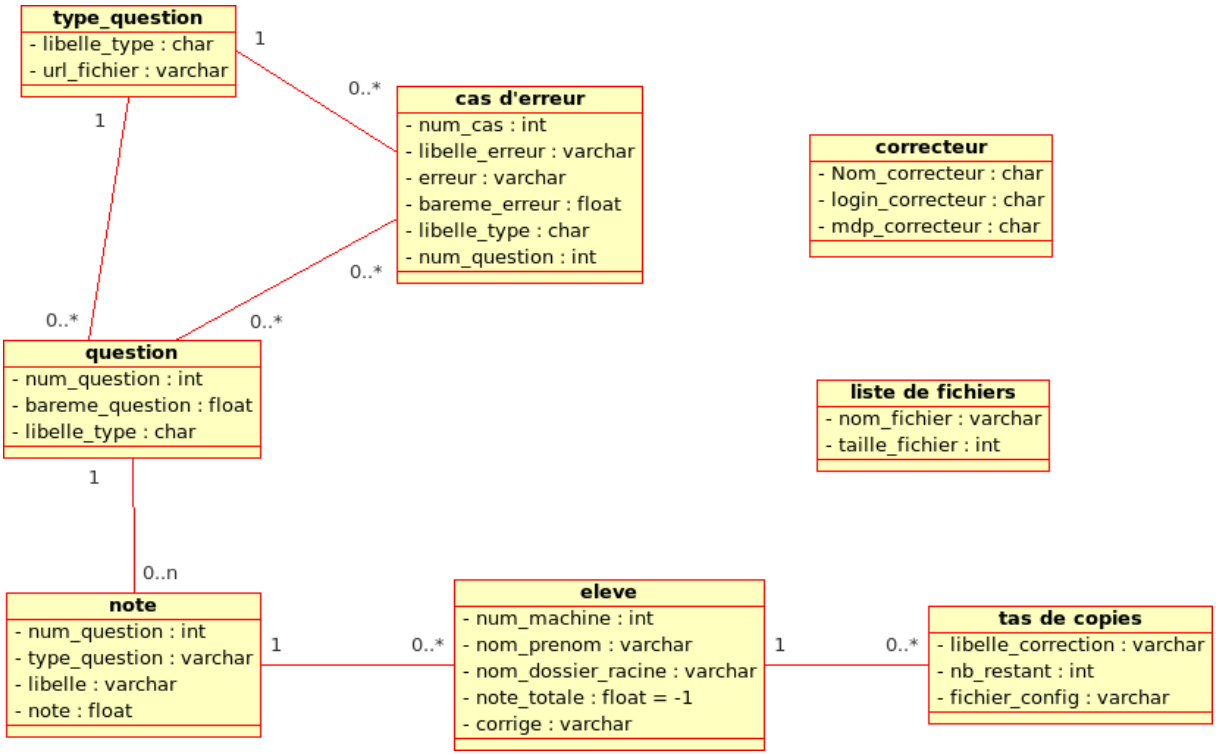


Figure 8 : schéma de la base de données

Cette base de données nous permet de réduire notre recherche de cas d'erreurs aux cas correspondant seulement à la question posée améliorant donc grandement la rapidité d'exécution. Par exemple, pour une question portant sur un répertoire nous regarderons seulement les cas d'erreurs associés au type de question répertoire. Suite à une discussion avec nos tuteurs nous avons aussi choisi de rajouter dans la table cas d'erreur un attribut url_fichier qui permet d'optimiser notre code, le but étant d'essayer de rendre notre script le plus générique possible. Au lieu de lancer une grosse exécution d'un fichier PHP contenant tous nos algorithmes, nous lançons simplement un petit noyau qui à chaque fois qu'il rencontre un cas d'erreur lance une fonction associée au traitement de ce cas d'erreur spécifique. L'attribut url_fichier permet donc d'accéder au fichier contenant la fonction PHP correspondante.

Exemple : Pour le traitement d'une balise <fichier>, notre programme va chercher dans url_fichier la fonction PHP qui traite les cas de fichier.

Modèle dynamique :

- *Diagramme de séquence* :

Tout comme le diagramme de cas d'utilisation, le diagramme de séquences décrit l'interaction entre les objets pour la réalisation d'une fonctionnalité de l'application. La différence réside dans le fait que pour ce dernier, l'accent est mis sur la chronologie des évènements, pas forcément sur la manière dont ils s'enchaînent car dans notre diagramme la plupart des actions sont indépendantes.

Nous aurions bien pu englober toutes les actions par la ligne de vie de « s'identifier » mais nous avons préféré modéliser le fait que l'utilisateur doit s'identifier avant de pouvoir effectuer les autres actions dans un ordre aléatoire, excepté pour celles qui sont englobées dans les lignes de vie des autres.

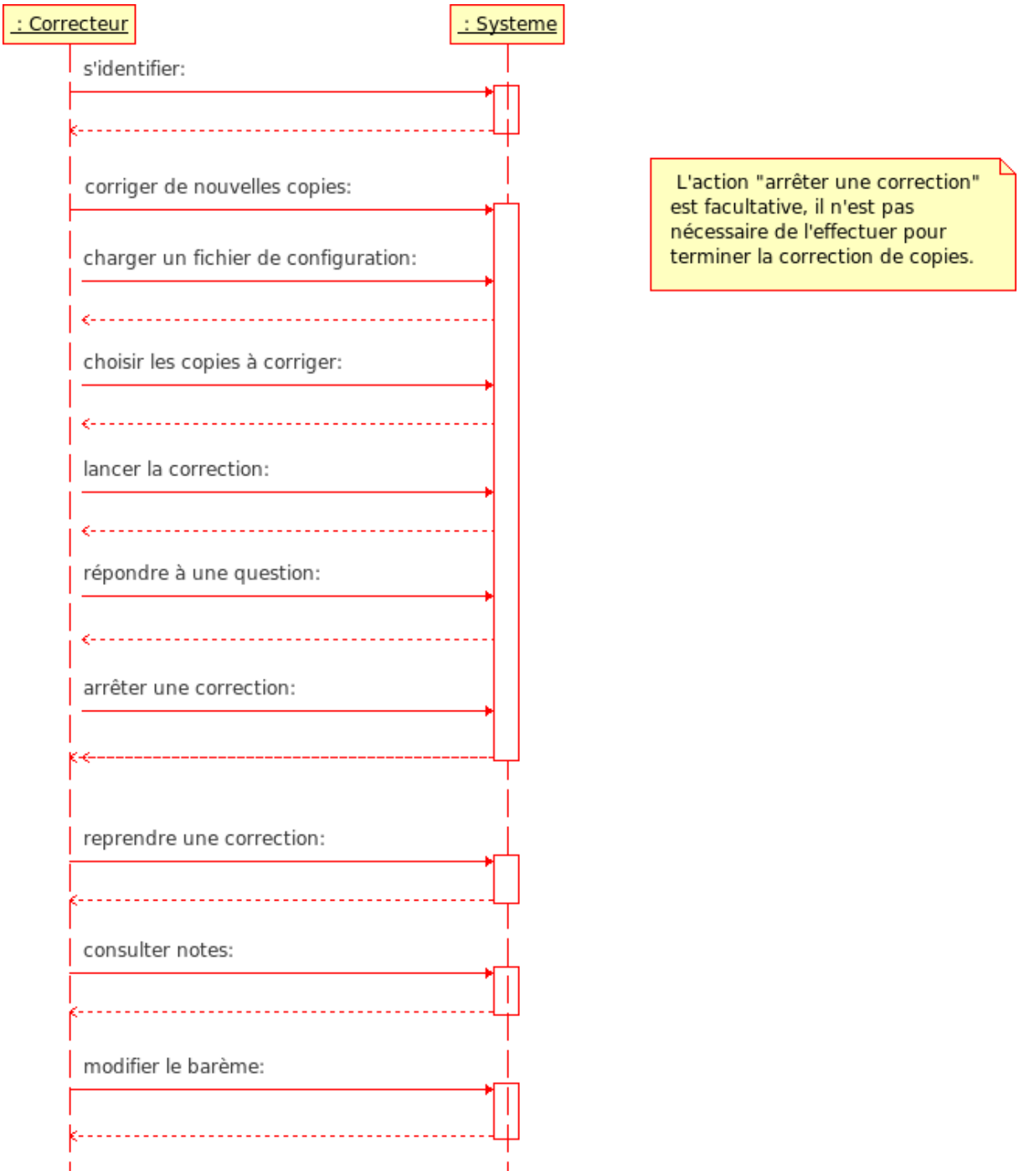


Figure 9 : diagramme de séquence

- *Diagramme d'états :*

Le diagramme d'états est attaché au diagramme de cas d'utilisation. Le déroulement des actions, exposées dans ce diagramme, représente les actions que le correcteur pourra effectuer lors de l'utilisation de notre outil. Ce diagramme met surtout en évidence l'action « Corriger de nouvelles copies » qui comporte en elle-même plusieurs actions nécessaires pour son fonctionnement.

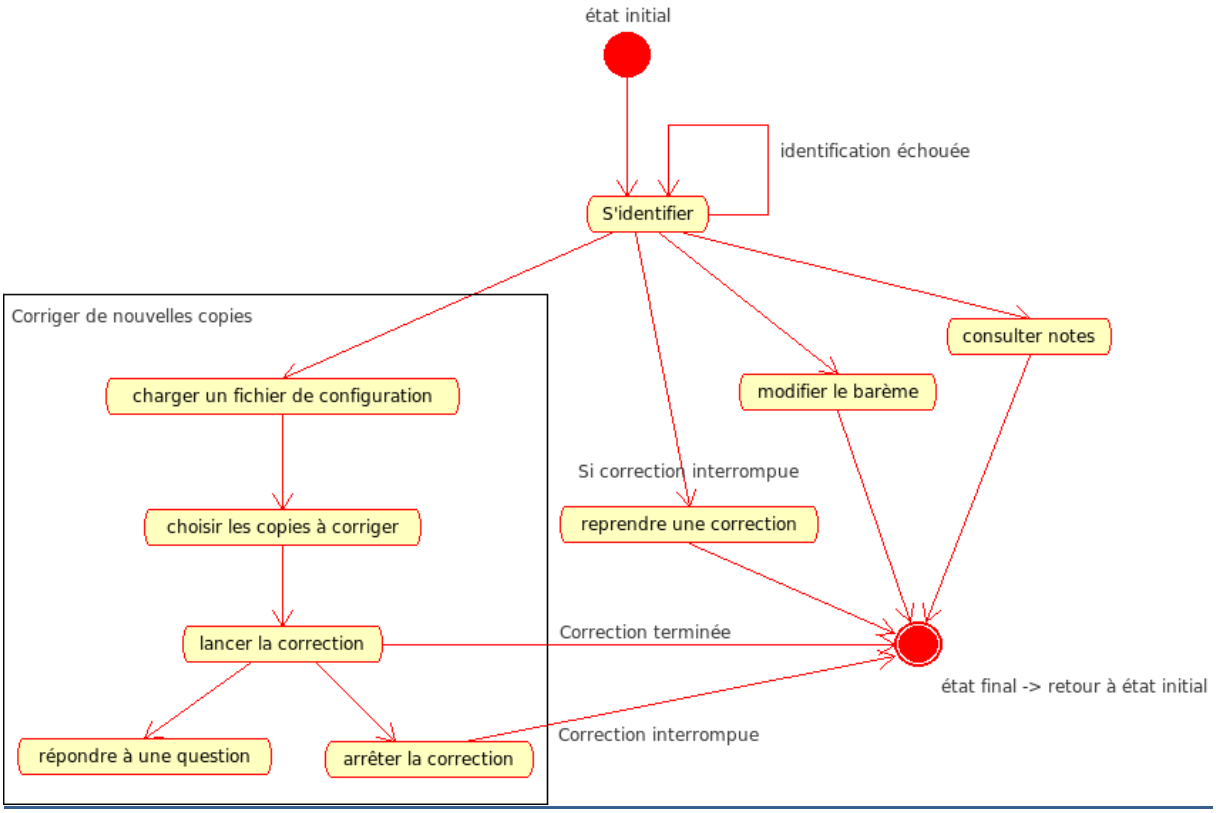


Figure 10 : diagramme d'états

On remarque que l'état final est en fait un retour à l'état initial.
En effet on peut relancer une correction immédiatement et indéfiniment tant qu'on fourni des copies à corriger à notre application.

III. Modélisation du traitement du cas de l'arborescence

C'est la plus grosse partie de notre projet et donc à fortiori la plus compliquée à modéliser.

Nous avons donc le cas parfait d'exercice qui est notre fichier de configuration. Nous devons donc comparer ce qu'a fait l'étudiant avec ce fichier. Deux approches sont apparues lors de nos discussions :

- créer un fichier XML pour l'étudiant similaire au fichier de configuration en se basant sur l'arborescence de l'étudiant, pour pouvoir ensuite comparer directement les deux fichiers XML.

- comparer en temps réel directement le fichier de configuration avec l'arborescence de l'étudiant.

La première idée a été retenue car nous avons estimés qu'il était plus judicieux de comparer deux fichiers du même type plutôt qu'un fichier et une arborescence de dossiers.

Exemple pour la comparaison des fichiers XML : (le contenu des fichiers XML a été volontairement modifié ici).

```
<repertoire_racine nom="NOM_Prenom_Exam" question="0" bareme="1">
    <eval question="1" bareme="0.333333">
        <repertoire nom="Images" question="1">
            <eval question="7" bareme="1">
                <eval question="5" bareme="1">
                    <fichier type_mime="null" nom="null" question="5"/>
                </eval>
                <eval question="6" bareme="1">
                    <fichier type_mime="png" nom="white_rollie.png" question="6"/>
                </eval>
            </eval>
        </repertoire>
        <repertoire nom="Textes" question="1">
            <eval question="2" bareme="0.5">
                <repertoire nom="openoffice_odt" question="2">
                    <eval question="3" bareme="1">
                        <fichier type_mime="odt" question="3" nom="null"/>
                    </eval>
                </repertoire>
                <repertoire nom="pdf" question="2">
                    <eval question="4" bareme="1">
                        <fichier type_mime="pdf" question="4" nom="null"/>
                    </eval>
                </repertoire>
            </eval>
        </repertoire>
        <repertoire nom="Autres" question="1">
            <eval question="8" bareme="1">
                <copie type="repertoire" question="8" nom="Images"/>
            </eval>
        </repertoire>
    </eval>
</repertoire_racine>
```

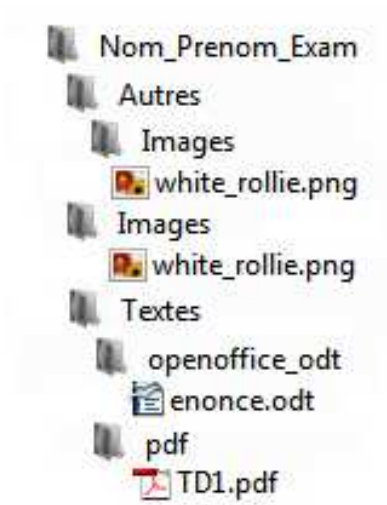
Comparaison plus simple

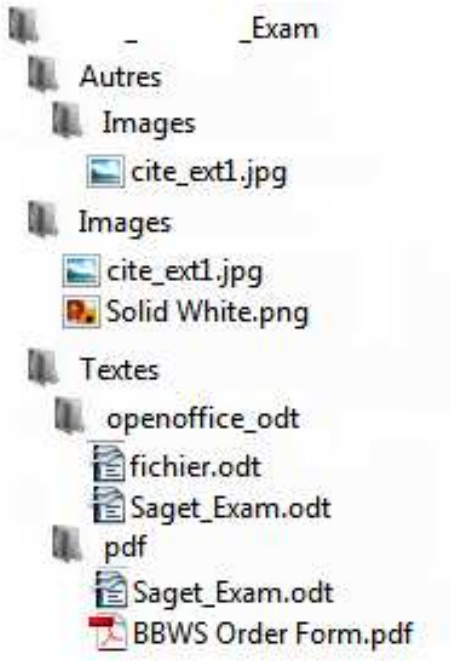
```
<repertoire_racine nom="      _       _exam">
    <repertoire nom="Autres">
        <repertoire nom="Images - copie">
            <fichier nom="cite_ext1.jpg" type_mime="jpg"/>
        </repertoire>
    </repertoire>
    <repertoire nom="Images">
        <fichier nom="cite_ext1.jpg" type_mime="jpg"/>
        <fichier nom="Solid White.png" type_mime="png"/>
    </repertoire>
    <repertoire nom="Textes">
        <repertoire nom="openoffice_odt">
            <fichier nom="fichier.odt" type_mime="odt"/>
            <fichier nom="Saget_Exam.odt" type_mime="odt"/>
        </repertoire>
        <repertoire nom="pdf">
            <fichier nom="BBSW Order Form.pdf" type_mime="pdf"/>
            <fichier nom="Saget_Exam.odt" type_mime="odt"/>
        </repertoire>
    </repertoire>
</repertoire_racine>
```

Fichier de configuration

Arborescence de l'étudiant

<u>Figure 11</u> : comparaison des fichiers XML : à gauche l'arborescence attendue, à droite celle de l'étudiant





Développement

Le développement est divisé en 3 parties que nous allons détailler :

- Extraction et traitement des copies : récupérer les informations à corriger
- Correction des copies et notes
- Présentation et interface

Pour chacune de ces tâches nous présenterons le principe, les entrées/sorties, les principaux algorithmes, les technologies utilisées et le design.

I. Extraction et traitement des copies

La première étape lors d'une correction de copies est de récupérer les données à corriger. En effet, le correcteur fournit en entrée une archive zip regroupant le contenu de chaque machine d'une salle à un instant donné. Chaque machine contient plusieurs répertoires mais un seul nous intéresse : celui où l'étudiant a déposé son travail et donc, celui à corriger. Eventuellement, on vérifiera le contenu de la corbeille car certains étudiants se trompent et suppriment leur travail en fin d'examen.

I.1 Entrée / Sortie

Entrée	Sortie
Fichier zip des copies	Chaque copie décrite par un fichier xml
Copie décrite par un fichier xml	Sous arbre xml représentant le travail de l'étudiant

Figure 12 : entrée/sortie du module d'extraction et traitement des copies

I.2 Technologies utilisées

Les appels système étant désactivés sur le serveur de l'UM3, il était donc impossible d'extraire une archive zip et de l'explorer. Ainsi, il fallait donc dans un premier temps trouver un moyen pour analyser le contenu de l'archive.

La librairie PHP pclzip offre une fonction permettant de récupérer des informations sur un fichier zip, sous la forme de textes placés dans des tableaux associatifs : répertoire par répertoire, fichier par fichier, sont récupérées des données comme le nom, la taille, la taille compressée… Le problème est que, parmi ces informations, beaucoup ne sont pas utiles ou utilisables. Les informations intéressantes concernent le nom, la taille et une variable de type booléen précisant si c'est un répertoire ou non. Il faut donc, à l'aide de PHP, fouiller les textes, garder ce qui est utile et le mettre en forme, c'est-à-dire dans un fichier XML afin de l'exploiter par la suite. Lors du parcours de l'archive grâce à pclzip, lorsqu'on tombe sur un nouvel étudiant, tout l'arborescence de sa machine est transcrite en XML et est encadrée par une balise <eleve>.

A ce moment là, l'ensemble des données de chaque machine est regroupé sous forme de données XML au sein d'un même fichier.

Dans un second temps, une fois l'archive analysée et formatée sous forme d'un fichier XML, on doit parcourir ce XML et le corriger. Le parcours s'effectue grâce à la librairie SimpleXML de PHP5. Lorsqu'on rencontre un nouvel étudiant, grâce à la balise <eleve>, on analyse les noms de chacun de ses répertoires afin de savoir si éventuellement ce sont des racines et donc, si le sous arbre extrait de cette racine doit être corrigé.

La racine correspond au répertoire supposé contenir le travail de l'étudiant. En effet, on attend de l'étudiant qu'il soit capable de créer un répertoire de la forme NOM_Prenom_Exam et d'y placer les autres répertoires qu'il crée.

Pour savoir si le répertoire est une racine, on compare à l'aide d'une expression régulière, son nom et la valeur de l'attribut "fin_nom" du fichier de configuration. L'attribut "fin_nom" a pour valeur par exemple "Exam".

Quand un répertoire racine est trouvé, on extrait le sous arbre issu de cette racine et on l'écrit dans un fichier XML temporaire qui est passé en paramètre au module de correction.

Si on ne trouve pas de racine, on propose au correcteur de choisir manuellement une racine grâce à une interface où l'arborescence de l'étudiant est affichée.

Technologies utilisées
PHP5
XML

Figure 13 : technologies utilisées par l'extraction et le traitement des copies

I.3 Algorithmes utilisés

Pour extraire les informations utilisables d'une archive zip :

```
Extraire_info_zip(archive_a_extraire)
{
    Ouvrir l'archive
    Récupérer les informations
    Parcourir toutes les informations
    {
        Si l'information est utile
        {
            Tester si c'est un nouvel étudiant
            Formater les données (balisage xml, attributs, etc.)
        }
    }
    Ecrire  dans un fichier XML
}
```

Figure 14 : algorithme d'extraction des informations

I.4 Design

Lorsque le logiciel n'arrive pas à trouver un répertoire racine pour un étudiant, il interroge le correcteur qui choisit ce répertoire manuellement. Une fois que le dossier a été choisi, un bouton pour confirmer et passer à l'étudiant suivant apparaît.

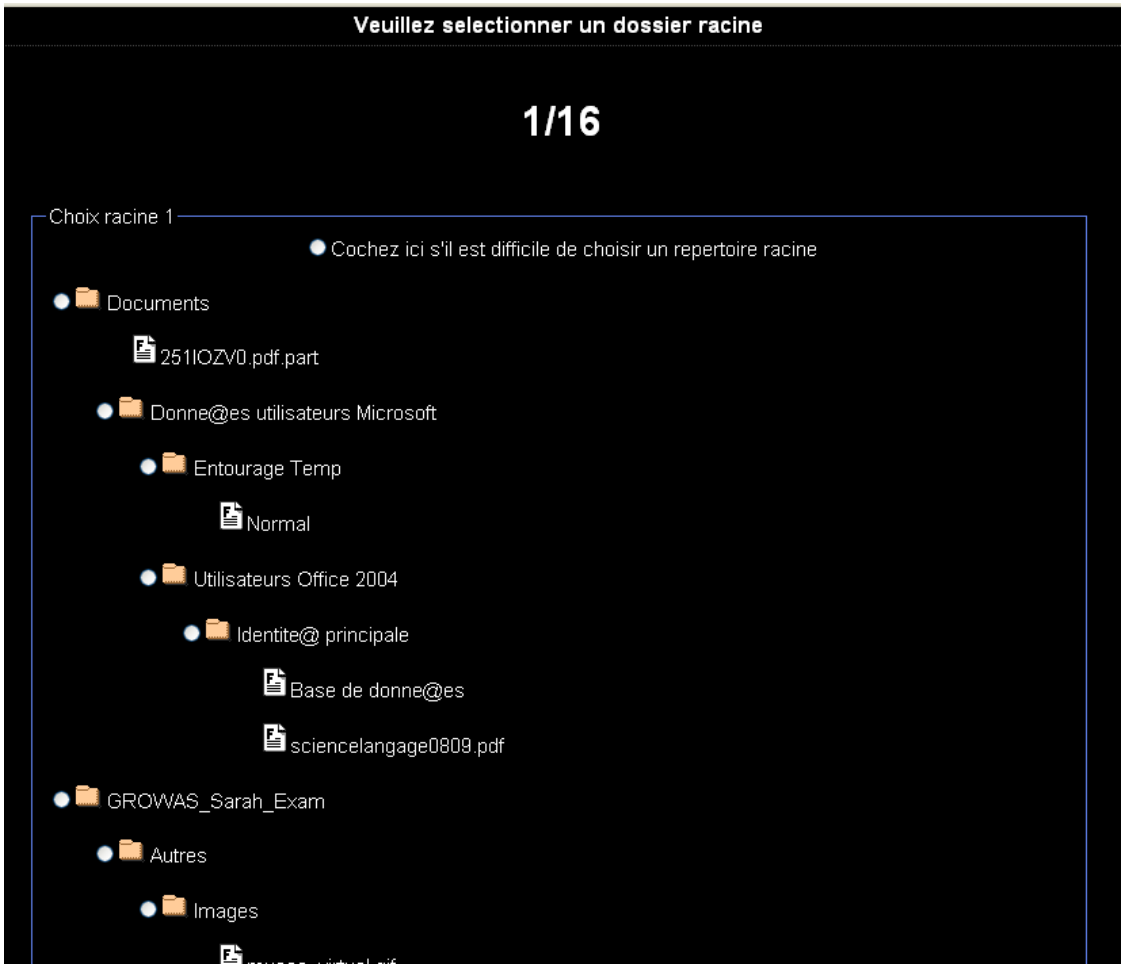


Figure 15 : interrogation du correcteur pour choisir manuellement une racine

II. Correction des copies

A ce stade nous avons donc un ensemble de copies d'étudiants générées au format XML ainsi qu'un fichier de configuration XML représentant l'énoncé de l'exercice. Le but de cette partie est donc de comparer les copies des étudiants avec le fichier de configuration et d'attribuer une note à chaque étudiant.

II.1 Entrée / Sortie

Entrée	Sortie
Chaque copie décrite par un fichier xml	Les notes de chaque étudiant
Une liste de copies pour chaque examen	Les notes de chaque examen

Figure 16 : Entrées/Sorties Correction de copies

II.2 Technologies utilisées

Pour effectuer cette tâche nous avons donc utilisé une extension de PHP5 permettant de parser des fichiers XML : SimpleXML.
Nous avons créé une classe nommée SXE qui hérite de SimpleXML afin de faciliter la manipulation des fichiers XML générés.
Nous nous sommes servis du logiciel libre EasyPHP qui intègre PHP/MySQL/Apache.

Technologies utilisées
PHP5
XML
SimpleXML
MySQL/SQL

Figure 17 : Technologies utilisées pour la correction

II.3 Algorithmes utilisés

Nous avons utilisé un raisonnement par cas de type RAPC comme présenté dans l'analyse (figure 4)
Voici notre algorithme général simplifié :

```
Parcours du fichier de configuration()
{
   Pour chaque question
   {
      Parcours du fichier de l'étudiant()
      {
         Si bonne réponse => Maximum de points
         Sinon si réponse approximative
         {
            Vérification si le cas existe dans la base de cas()
            {
               Si oui => attribution des points
               Sinon => Intervention du correcteur, attribution des points et
                  insertion du nouveau cas dans la base de données
            }
         }
         Sinon si aucune réponse => pas de point
      }
   }
}
```

Figure 18 : algorithme de correction

Pour certains traitements nous avons utilisé l'algorithme de LEVENSHTEIN. Cet algorithme renvoie le nombre de transformations nécessaires pour passer d'une chaîne de caractères à une autre :

Exemple :

Levenshtein(Images, Image) renvoie 1 (Le « s »).

Levenshtein(Images, image) renvoie 2 (Le « I » majuscule et le « s »).

L'algorithme de Levenshtein est en O (m*n) où n et m sont la taille de chaque chaîne passée en paramètre.Voir l'algorithme ci-dessous.

```
entier Levenshtein(caractere chaine1[1..longueurChaine1], caractere
chaine2[1..longueurChaine2])

  declarer entier d[0..longueurChaine1, 0..longueurChaine2]

  declarer entier i, j, coût

  pour i de 0 à longueurChaine1
     d[i, 0] := i
  pour j de 0 à longueurChaine2
     d[0, j] := j

  pour i de 1 à longueurChaine1
     pour j de 1 à longueurChaine2
        si chaine1[i] = chaine2[j] alors coût := 0
                        sinon coût := 1
        d[i, j] := minimum(
                        d[i-1, j  ] + 1,     // effacement
                        d[i, j-1] + 1,     // insertion
                        d[i-1, j-1] + coût   // substitution
                    )

  retourner d[longueurChaine1, longueurChaine2]
```

Figure 19 : algorithme de Leveinshtein

Cet algorithme nous a permis de trouver les réponses approximatives des étudiants et d'attribuer un barème adéquat à des fautes dans les noms de fichiers. En effet nous avons fait un rapport (résultat de Levenshtein) / (taille du mot) et nous avons acceptés 33% d'erreurs. Exemple :

« image » au lieu de « Images » :

La taille du mot est de 6 et le Leveinshtein renvoie 2 : 2/6 = 0.33 => 33% d'erreurs

En dessous de ce seuil nous regardons seulement si le mot est contenu dans la chaine de caractères.

Exemple :

« Opennoffice_pdf » au lieu de « pdf ».

Beaucoup trop d'erreurs par rapport à la taille du mot donc on regarde seulement si pdf est contenu dans Opennoffice_pdf ce qui est le cas ici.

Notre logiciel doit se rapprocher d'une correction humaine et donc doit être clément sur certains points. La où systématiquement le correcteur humain attribue quelques points il faut que notre application le fasse aussi.

III. L'interface

Dans cette partie nous allons vous expliquer les choix que nous avons faits pour la création de l'interface, ainsi que les outils utilisés. Nous vous expliquerons notamment l'utilité d'avoir une interface dont l'élément principal n'occupe pas toute la page.

III.1 Le Design

Pour créer notre interface, nous avions trois éléments principaux à prendre en compte. Il fallait, d'une part que l'interface soit agréable pour le correcteur. Nous avons donc voulu créer une interface à la fois sobre et élégante.

D'autre part, nous devions afficher les différentes fonctionnalités de notre logiciel. Ces fonctionnalités ne devaient, de plus, être affichées qu'à certaines conditions, comme par exemple si l'utilisateur s'est logué.

Enfin, notre phase d'analyse a révélé que le contenu à afficher n'occuperait que peu de place.

Figure 20 : page d'accueil de l'application

III.2 Les outils et les contraintes

Pour créer notre interface, nous avons utilisé le freeware The GIMP. Cet outil a permit la réalisation des différentes parties de l'interface, de l'image de fond, aux boutons du menu général.

L'intégration de cette interface à l'aide d'une feuille CSS a rendu notre site compatible avec l'explorateur Firefox, compatibilité qui nous avait été demandée.

III.3 Une interface en trois parties

L'interface se compose de trois parties : un menu général à droite qui permet d'accéder aux fonctionnalités du logiciel ; le cadre central est la partie principale de l'interface, et à gauche on retrouve un menu spécifique à la page consultée.

Les trois parties se trouvent dans des fichiers différents que l'on intègre aux différentes pages à l'aide de la fonction **include** de PHP. Ainsi, il est très facile de cacher le Menu général et/ou le menu de gauche, ce qui est notamment nécessaire au cours d'une correction.

III.3.1 Le Menu de Gauche

Le menu de gauche intègre un lien permettant de se déconnecter proprement du site. Rappelons que pour accéder au logiciel de correction il faut au préalable se loguer.

C'est également dans cette partie de l'interface que l'on trouvera le sous-menu concernant la partie consultée par l'utilisateur :

Figure 21 : présentation du menu de gauche de l'application

III.3.1 La page principale

C'est ici que l'on indiquera au correcteur les éléments importants tels que les nouveaux cas d'erreur.

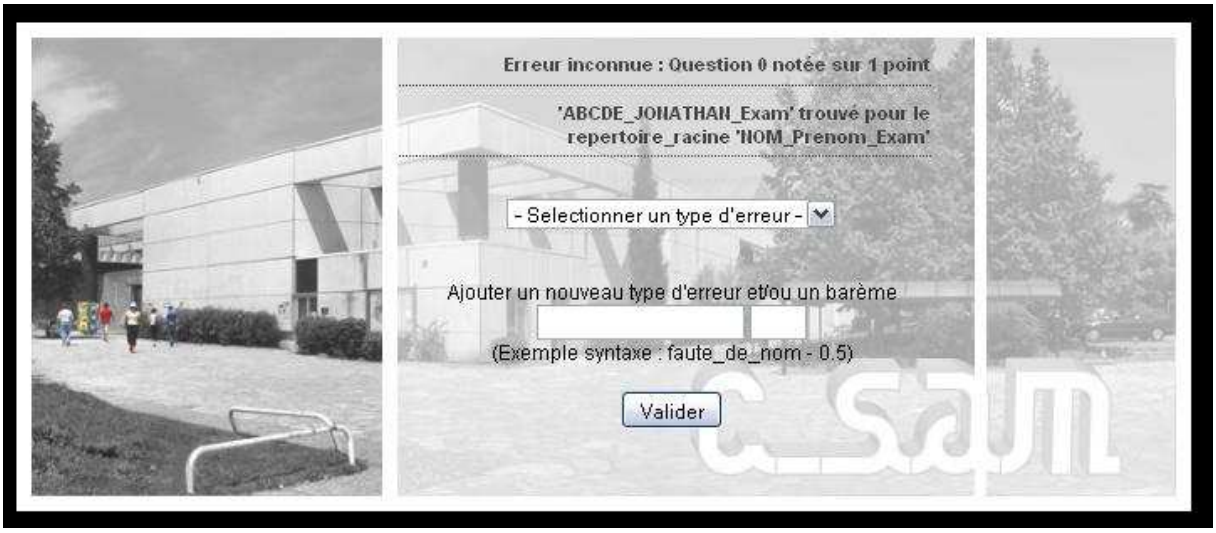


Figure 22 : affichage d'un exemple d'erreur sur la page principale

On constate d'ailleurs ici que les menus, de gauche et droite, sont cachés. En effet pendant une correction, il ne doit pas être possible, par exemple, d'en lancer une nouvelle. L'interface permet ainsi de contrôler les actions de l'utilisateur.

Le fond a été rendu légèrement opaque pour améliorer la lisibilité du texte qui figure au premier plan.

III.3.1 Le Menu de droite

A droite on retrouve donc le menu général avec un lien pour revenir à l'accueil, et un lien pour chaque fonctionnalité du logiciel. C'est à partir de ce menu que l'on va pouvoir notamment lancer une nouvelle correction en cliquant sur « Nouvelle Correction ».

Figure 23 : présentation du menu de droite, permettant la navigation à travers le site

III.3 Une interface évolutive

Nous avons imaginé notre interface en gardant en tête le fait que des futures fonctionnalités pourraient être intégrées à notre application.

Ainsi, le menu de droite comporte les éléments principaux du logiciel. Les éventuelles applications futures se grefferont alors dans l'une de ces parties.

Elles pourront être facilement ajoutées dans le menu de gauche qui est associé à cette partie.

IV. Les notes

Un des avantages de notre application est d'avoir les notes totales et les notes par question de chaque étudiant dans une Base de Données. Ainsi la partie « Note » permet de consulter de manière aisée ces notes. Si le correcteur estime avoir mal évalué le barème d'une ou plusieurs questions, il est également possible de les modifier.

IV.1 Consultation des notes

Nous avons donc un script de connexion à la BD qui permet ensuite d'effectuer des requêtes SQL sur les tables.

On trouve notamment un moteur de recherche qui permet de restreindre la liste des étudiants. Il est possible d'effectuer une recherche sur le nom de l'étudiant, sur son prénom, ou sur le numéro de machine sur laquelle il a passé son examen.

- *Liste de notes d'étudiants*

Par défaut la liste de tous les étudiants avec leur note totale sont affichés au correcteur. S'il le souhaite, il peut consulter le détail des notes d'un étudiant en cliquant sur son nom. Un lien lui permet ensuite de revenir à sa recherche ou à la liste complète des étudiants.

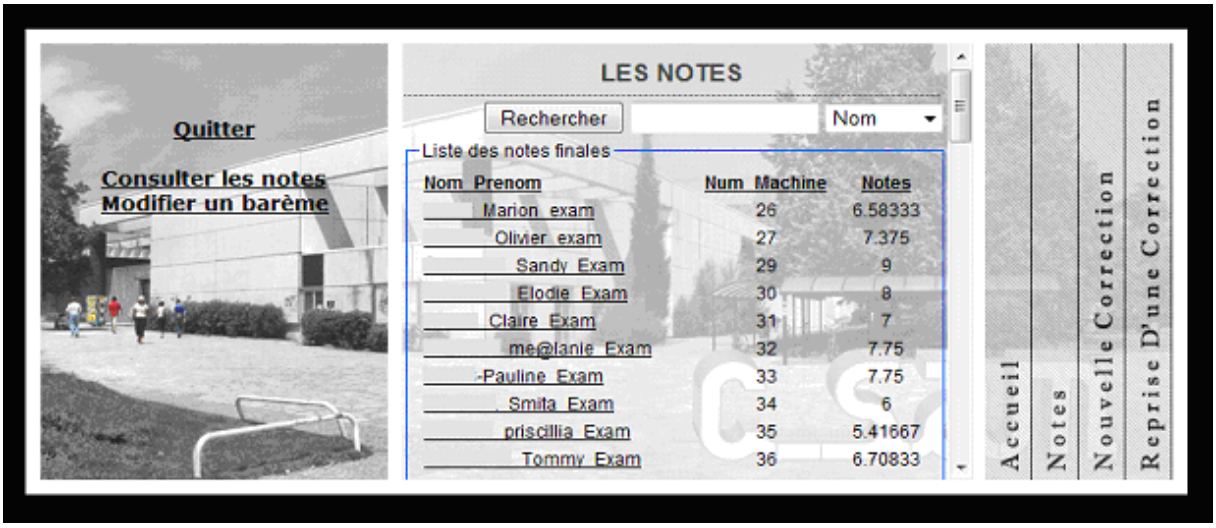


Figure 24 : Liste de notes des étudiants

- *Détail des notes d'un étudiant*

Le correcteur a la possibilité de consulter le détail des notes d'un étudiant, c'est-à-dire question par question. Ce qui peut s'avérer être utile si l'étudiant souhaite discuter de sa note avec le correcteur.

- *Détail des notes d'un étudiant*

Le correcteur a la possibilité de consulter le détail des notes d'un étudiant, c'est-à-dire question par question. Ce qui peut s'avérer être utile si l'étudiant souhaite discuter de sa note avec le correcteur.

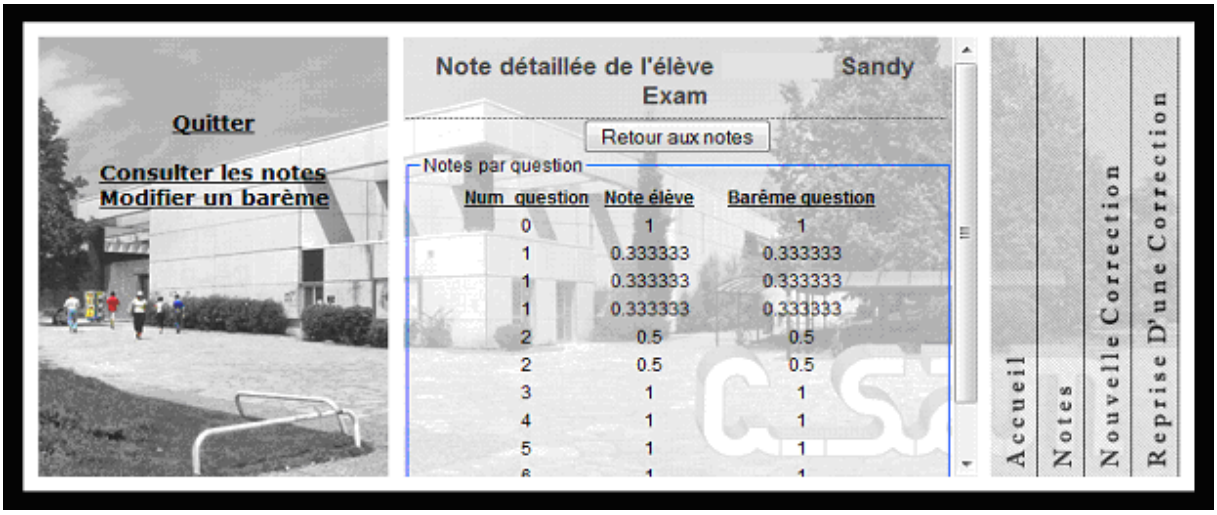


Figure 25 : détails des notes d'un étudiant

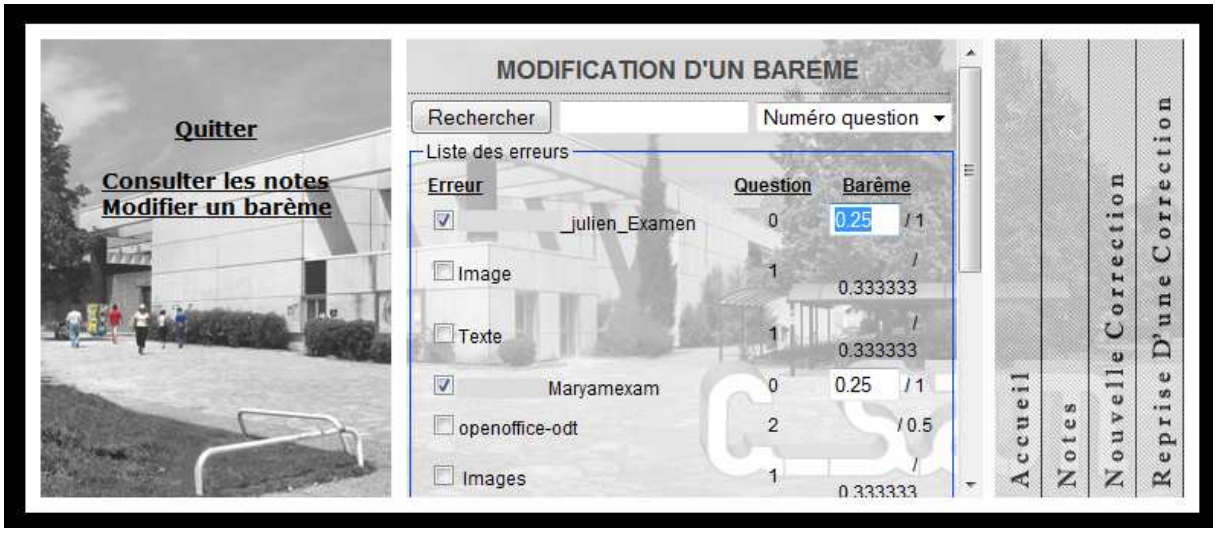
IV.2 Modification d'un barème

L'accès à cette section se fait par le menu de gauche de la partie Note, en cliquant sur le lien « Modifier un barème ».

Ici le principe est le même que pour la consultation des notes. On se connecte dans un premier temps à la base de données, puis on effectue des requêtes SQL sur ses tables.

Là encore on retrouve un moteur de recherche car par défaut on affiche toutes les erreurs. S'il le souhaite, le correcteur peut ainsi restreindre l'affichage à quelques erreurs ou même à une seule. Il peut faire une recherche par numéro de question, par barème, ou par erreur.

- *Modification d'un barème*



Une fois la modification du barème prise en compte, le correcteur est invité à relancer la correction afin de conserver une correction équitable de toutes les copies.

Cette nouvelle correction sera très rapide puisque toutes les copies ont déjà été corrigées, et donc les erreurs sont déjà toutes dans la base de données. On peut ainsi dire que c'est quasi instantané.

Discussions

I. Difficultés

Tout au long de notre projet, notre principale difficulté a été de se plier aux contraintes imposées par le serveur webtest, serveur de test mis à notre disposition par l'Université Montpellier 3. Ce serveur dispose des mêmes configurations que le serveur de production qui accueillera notre application. En effet par mesure de sécurité, celui-ci fixe des limites sur de nombreux points. Citons par exemple un temps d'exécution de trente secondes maximum pour un script, et un upload de fichier inférieur à 2 Mo.

De plus, les appels système étant bloqués, il nous a été impossible de pouvoir effectuer des opérations comme dézipper une archive.

Pour contourner ces problèmes nous avons notamment utilisé des librairies (pclzip et SimpleXML). Il a donc fallu apprendre celles-ci afin de les utiliser au mieux.

En ce qui concerne l'interface, elle devait être claire et faciliter la prise en main du logiciel. De plus cette interface devait être compatible avec l'explorateur Mozilla Firefox, à la demande de nos tuteurs.

La réalisation du projet se faisant en même temps que les cours, il a fallut également avoir une bonne organisation pour ne pas négliger le travail quotidien et avancer dans le développement du logiciel.

II. Améliorations

Nous n'avons pas pu implémenter la partie permettant la correction de documents structurés. Cette partie a, dès le début, été définie comme optionnelle et nous nous sommes donc concentré sur la correction d'exercices de gestion d'un espace de travail, afin de fournir l'outil le plus efficace possible. Cependant, notre logiciel permet d'intégrer la correction d'autres exercices comme celui cité précédemment.

La partie graphique et la base de données sont de plus suffisamment modulables pour une intégration facile de telles corrections.

III. Par rapports aux objectifs

Les objectifs suivant ont été atteints :

- Permettre la correction de l'exercice sur l'arborescence par un seul correcteur.
- Réduire le temps de correction
- Conserver une généricité permettant d'adapter notre application à d'autres exercices en minimisant les modifications à effectuer.

Nous avons donc livré notre application dans les délais malgré les contraintes imposées.
Notre logiciel permet de corriger de façon équitable, rapide et précise.
Les tableaux ci-dessous illustrent ces qualificatifs.

NOTES

Humain	Machine
8,5	8,25
8,5	7,6
7,5	7,75
5,5	5
7	7
5	3,9
7,5	6
8,5	7,5
9	8,5
8,5	8,5

TEMPS

	Humain	Machine
20 min	**16 copies**	**100 copies**
1h	**50 copies**	**384 copies**
2h	**100 copies**	**2000 copies**

De plus lorsque le correcteur corrige un tas de copies il ne sait pas quelle erreur correspond à quelle copie donc on obtient une correction anonyme.

Enfin le barème des erreurs appliqué est constant. La subjectivité humaine si elle intervient au moment de l'association de la note à l'erreur sera la même pour toutes les copies où cette erreur sera rencontrée.

Conclusion

Nous avons trouvé ce projet très intéressant car on nous offrait l'opportunité de participer à une application qui serait un support d'aide pour les professeurs de l'Université Montpellier III.

Ce projet a nécessité de nombreuses recherches afin de mieux maîtriser le langage PHP, langage que nous avons qu'entre aperçu en troisième année de Licence. Il a donc fallut se prendre en main totalement pour se former, en quelques sortes, à ce langage, ce qui n'a été que bénéfique pour nous puisque le PHP est très couramment utilisé dans le développement d'applications web.

Nous tirons entière satisfaction de ce projet puisque notre objectif premier est atteint : nous avons obtenu un logiciel de correction semi-automatique qui permet de corriger de manière aisée des copies du C2I, et de consulter les notes des étudiants ainsi attribuées.

Un utilisateur lambda peut facilement naviguer dans le site et effectuer toutes les actions possibles.

Cette expérience très enrichissante tant sur le plan technique qu'au niveau de l'organisation nous a permis d'avoir une première vue sur la façon de mener un projet sur une période relativement longue, et des difficultés que l'on peut rencontrer. Il nous a également éclairés sur l'importance des choix que l'on a à faire.

Nous d'avons pas pu réaliser, par manque de temps, la correction d'exercices portant sur la structuration de documents complexes. Mais la correction d'exercices de ce type pourra parfaitement être développée et intégrée à notre projet puisque nous laissons un logiciel générique et clairement structuré, et qui peut ainsi être facilement repris par une autre personne étrangère à notre projet.

Cela pourrait d'ailleurs faire éventuellement l'objet d'un futur projet.

Glossaire

- L -

Loguer (Se) : Anglicisme du verbe « log », signifie s'identifier.

Login : Identifiant d'un membre.

- M -

MySQL : Gestionnaire de bases de données open source, majoritairement utilisé par les webmasters de projets de petite ou moyenne envergure.

- P -

Pseudo : Abréviation de pseudonyme, nom choisi par le membre qui sera visible par les autres.

PHP : HyperText PreProcessor, est un langage de scripts libre principalement utilisé pour être exécuté par un serveur HTTP. Ce langage peut servir de lien entre le HTML et les bases de données *MySQL*.

-R-

RAPC : Raisonnement A Partir de Cas.

- S -

Session : Mot désignant l'exécution d'un programme pour un utilisateur donné. L'exécution du programme est alors paramétrée par les informations du profil de l'utilisateur (droits d'accès par exemple).

Source : Les sources d'un site sont tous les fichiers qui constituent ce premier. Une fois tous assemblés, on obtient le site fonctionnel.

- U -

UML : Unified Modeling Language, langage de modélisation objet unifié. Méthode de modélisation utilisée dans les entreprises pour modéliser les systèmes des applications à mettre en place.

UML est aujourd'hui un standard incontournable.

Annexes

Références :

- Utilisation du programme client VPN CISCO, outil d'accès sécurisé au réseau de l'université de Montpellier III :
  - Ø http://www.univ-montp3.fr/crit/vpn
- Tutoriel : utilisation de l'extension SimpleXML :
  - Ø http://fr.php.net/simplexml
  - Ø http://metacites.metawiki.com/php/simplexml
- Distance de Levenshtein :
  - Ø http://fr.wikipedia.org/wiki/Distance_de_Levenshtein

Manuel utilisateur :