Université de Cergy-Pontoise

Maîtrise d'informatique

2004 - 2005

Jean-Philippe Aumasson <jpnossa@free.fr>

Jean-Baptiste Langlois <jean-baptiste.langlois@wanadoo.fr>

DOCUMENT D'ANALYSE

9th January 2005

REALISATION: MODELISATION OBJET AVEC UML, PROGRAMMATION DE L'APPLICATION CLIENT/SERVEUR EN PHP, BASES DE DONNES RELATIONELLES MYSQL.

SUJET: GESTION D'UN ETABLISSEMENT SCOLAIRE DE TYPE UNIVERSITE.

Abstract

Ce document décrit et justifie l'analyse du système d'information étudié, notamment à l'aide des sémantiques UML.

Contents

1 Introduction

L'analyse du système à concevoir est une des premières étapes du développement d'un logiciel, avant de chercher comment programmer nous devons savoir quoi programmer, autrement dit quelles fonctions, au sens large, doit posséder le programme. La détermination de ces fonctions passe par une étude rigoureuse des besoins à satisfaire : identification des spécifications explicites du cahier des charges ou du sujet, ainsi qu'une réflexion personnelle sur le modèle à concevoir. La programmation orientée objet facilite cette tâche, ainsi que l'usage du langage UML.

Nous décrirons la première phase clé de l'analyse ainsi que les aspects pratiques dans la première partie, avant de définir les besoins et formuler les cas d'utilisation de l'application. Alors fixées les limites du logiciel, on pourra s'avancer un peu plus dans la conception, avec tout d'abord l'identification des entités entrant en scène, un diagramme de classe UML formalisera cette étude par la représentation des classes et de leurs associations. Une base de donnée relationnelle sera utilisée pour le stockage de l'information, nous présenterons les tables crées. Enfin, la dernière partie traitera de l'interface réalisée (IHM).

Le travail d'analyse a été réalisé dans ses grandes lignes avant l'implémentation, mais la réflexion sur le modèle, ses remises en question et modifications se sont poursuivies lors de la mise en forme de l'application, celle-ci n'étant pas un simple recopiage du modèle pré-implémentation. Idéalement, dans un milieu professionnel, l'idéal serait sûrement de concevoir le modèle définitif avant implémentation, les concepteurs et développeurs n'étant pas forcément les mêmes personnes.

On se réfèrera au document technique pour des informations détaillées sur l'implémentation, notamment pour la maintenance et l'évolutivité du logiciel.

2 Première approche

2.1 Objectifs

Le sujet nous est disponible dès le vingt-cinq octobre, une certaine liberté nous est offerte pour la conception du modèle, cependant un certains nombre de fonctionnalités restent imposées.

Nous entamons dès lors une première analyse globale du sujet, clairement et volontairement non exhaustive, comme suit : informées l'existence d'un projet à réaliser pour ce cours, nous nous étions mis d'accord pour travailler ensemble, et avions déjà pensé à la possibilité de réaliser l'implémentation en langage PHP, si une application client/serveur était demandée, et si le langage n'était pas imposé.

Des diagrammes de classes et de cas d'utilisations assez grossiers sont rapidement établis, afin de nous permettre une meilleure visualisation du problème. Ces diagrammes ne sont pas présentés dans ce rapport, cette première vue étant bien évidemment très éloignée du résultat final, et ne mérite pas qu'on s'y attarde.

Lors de l'analyse préliminaire nous nous fixons comme limites les fonctions imposées dans le sujet. Les classes candidates sont les suivantes, par ordre alphabétique :

(nous respecterons les conventions de nommage suivantes: nom de classe en minuscules sauf les premières lettres des mots qui le composent, sans séparateur; noms de fonctions et variables selon la même règle, sauf la première lettre qui sera en minuscule. Tous les noms seront en anglais.)

Diploma, Discipline, Employee, Exam, ID, IDs, Mark, Person, SchoolYear, Student, Teacher, University.

On choisit de traiter le simple cas d'une université pluridisciplinaire, préparant seulement aux diplômes du cycle "LMD", soit la licence en trois ans, le master en deux, puis un doctorat en cinq.

Nous raffinons ensuite cette première analyse en commençant à penser aux données à stocker, et à leur structuration (dépendances, clés et index, etc...), de nouvelles classes sont crées, leurs relations précisées.

Nous utiliserons une base de données sous le modèle relationnel, l'application est développée en utilisant un compte du FAI Free : base de données mySQL, support de PHP4. On pourra cependant installer le programme définitif sur une base de donnée locale, ou sur une autre base distante, à l'aide de l'assistant d'installation (page *install.php*).

Pour la suite de l'étude nous suivrons une démarche itérative et incrémentale, consistant à raffiner progressivement la description du modèle en fonction d'une part de notre réflexion sur la conceptualisation, d'autre part des modifications et raffinements apportés durant l'implémentation qui constituera l'expérimentation de nos conceptions et dont on déduira donc au fur et à mesure validation et remises en question du modèle. La difficulté étant de "savoir s'arrêter", pour ne pas plonger dans le détail avant d'avoir une cohérence des éléments essentiels. Manquant d'expérience dans la conception de systèmes d'information, la dissociation totale de l'analyse et de l'implémentation n'est pas envisageable.

Nous devrons veiller à respecter les contraintes fournies par le sujet, réaliser une application utilisable par un utilisateur lambda, c'est à dire simple d'usage, intuitive, ergonomique et conviviale, tout en traitant au mieux le sujet pour obtenir un modèle complet et cohérent, sans trop s'écarter du cadre de l'étude, et tout cela à l'aide d'une implémentation efficace et robuste, facilement compréhensible par un autre développeur amené à effectuer maintenance ou amélioration de l'application.

2.2 Interrogations et hypothèses

Le programme devant être utilisable pour la gestion d'un établissement scolaire de type université, la notion de temps devra être implémentée. Dans ce cas de gestion d'évènements "réels", un programme en temps réel pourrait être adapté, cependant nous devrions définir précisément chaque échéance au préalable. Un écoulement du temps discret nous semble a priori mieux adapté, pour s'en rendre compte nous établissons un premier diagramme de cas d'utilisations, ainsi qu'un *statechart*, rendant seulement compte des fonctionnalités les plus basiques, mais aussi les plus essentielles : inscrire un étudiant à l'établissement, définir un diplôme, mettre à jour les notes, valider ou pas une année scolaire. Nous nous rendons compte qu'une année peut être divisée en quatre périodes, qui correspondent à autant d'états du *statechart* :

- Le début d'année, la pré-rentrée : inscription d'étudiant, recrutement d'enseignants, création de filières et de diplômes, attribution des charges.
- La fin du premier semestre : sont collectées les notes correspondant aux divers modes de validation. Et éventuellement la désinscription d'étudiants ou le changement d'orientation, mais ces dernières options sont encore des détails à ce stade de l'analyse.
- La fin du second semestre, ou fin d'année: les notes du semestre sont enregistrées, car où on devra gérer d'éventuelles mentions, mais encore une fois ceci reste un détail pour l'instant.

- Le rattrapage, pour les étudiants n'ayant pas validé leur année, et ayant des notes suffisantes pour espérer être repêchés, et n'étant pas victimes de notes éliminatoires.

On distingue alors la frontière entre le niveau statique et le niveau dynamique du programme, car on pourrait réaliser une application permettant d'ajouter et supprimer des étudiants, recruter des enseignants et leur attribuer des cours en charge, et même insérer des notes à un étudiant en fonction de ses inscriptions, mais elle se révelerait vite incohérente, par exemple de part la possibilité de faire passer plusieurs années scolaires à un étudiant tandis qu'un autre n'en est qu'à son premier semestre.

Bien que ce modèle à quatre états ne traite pas tous les cas particuliers et situations extravagantes possibles, il nous semble après réflexion un bon départ, sujet à améliorations.

Il est tout de suite nécessaire d'émettre des hypothèses, faire des concessions par rapport à la réalité, et définir des règles. On émet pour l'instant une hypothèse simplificatrice : tout le monde "rentre" au même moment, en début d'année. On passera d'un état au suivant par exemple en cliquant sur "Validation du semestre", et on se trouvera ensuite dans l'état suivant, ne pouvant revenir dans le passé pour entrer d'éventuelles notes oubliées. Celà revient à poser implicitement l'hypothèse d'un respect des délais de la part des correcteurs.

Ce choix a pour avantage de ne pas pouvoir "trafiquer" des notes *a posteriori*, et pour l'utilisateur d'éviter des confusions. Cependant il est assez rigide car il serait alors impossible d'entrer des notes délivrées plus tard, de les modifier ou corriger si nécessaire.

Nous avions dans nos premiers diagrammes émis l'hypothèse que seul le gestionnaire de la scolarité serait l'utilisateur du programme, ainsi que toutes les personnes auxquelles il peut déléguer ses responsabilités. Mais dans le cas de personnes mal intentionnées, ou d'utilisation par une tierce personne non autorisée, on imagine déjà des situations d'utilisation frauduleuse.

Des considérations sécuritaires sont donc nécessaires, nous pensons d'abord à la plus simple, la plus utilisée, mais certainement pas la plus sûre : l'authentification par un couple login/mot de passe.

Nous songeons alors à étendre l'accès au système à d'autre utilisateurs, dont l'accès au système peut être utile, dans une certaine mesure. Nous sommes ainsi amenés à définir des classes d'utilisateurs du programme, des profils définissant un ensemble de restrictions, dont un super-utilisateur qui aurait tous les droits. Les étudiants pourront aussi consulter leur notes, en mode lecture-seule, et les enseignants inscrire les notes de leurs enseignements.

2.3 Organisation du travail

JB étant plus familier du PHP et des bases de données, il s'occupera d'abord des structures de données à concevoir et à implémenter, et complétera l'analyse UML statique et dynamique élaborée par JP, en fonction des contraintes de développement. Une fois le modèle établi nous travaillerons tous les deux sur l'implémentation et l'interface de l'application. Munis des idées et hypothèses définies plus haut, et d'un aperçu global du sujet, nous pouvons commencer l'analyse.

L'analyse ne s'arrêtant pas après la première ligne de code, nous avons constamment débattu les choix effectués, quitte à modifier le modèle du document d'analyse et/ou les sources du programme.

3 Besoins, conceptualisation et cas d'utilisation

3.1 Analyse des Besoins

L'application s'adresse aux classes d'individus suivants:

- Les responsables, par exemple le personnel du service de scolarité de l'établissement, les secrétaires de départements, qui auront les droits les plus larges, et l'accès à toutes les fonctions du logiciel, et les plus grandes responsabilités.
- Les enseignants, qui pourront (la tâche pourra être déléguée à une secrétaire) entrer les notes des examens de leur enseignement, consulter la base des étudiants et des enseignants.
- Les étudiants, qui consulteront leur notes, et seulement les leurs. On considère comme étudiant un individu inscrit à l'université pour au moins un diplôme. On ne gérera pas d'éventuelles contraintes judirico-sociales ou biologiques, par contre il pourrait être nécessaire de fixer une limite d'âge, inférieure, fonction de l'année courante. Les étudiants pourront consulter les listings des étudiants et enseignants.

On définit les besoins à satisfaire pour chacune de ces classes d'utilisateurs, ainsi que les contraintes à imposer:

- Le responsable, souvent du personnel peu qualifié en informatique, attend un programme simple d'utilisation, intuitif, convivial, avec un minimum de considération techniques et de choix ou ordres incompréhensibles ("Pour activer cette fonction veuillez recompiler votre noyau en ajoutant le module correspondant à votre modèle de périphérique..."). Nous devrons

donc conserver la plus grande simplicité d'utilisation, qui réside d'une part dans les actions à effectuer (par exemple, éviter d'avoir à parcourir quinze écrans pour quitter), et d'autre part à la façon de les effectuer (composants d'interactions, boutons, etc...faciles d'accès, explicites, de taille raisonnable...). L'aspect esthétique et confort ne doit pas être négligé, si un utilisateur trouve le logiciel laid il ne se plaira pas à travailler dessus, sa productivité s'en verra amoindrie. Enfin, toutes les opérations utiles à la gestion du système scolaire devront être réalisables, et on tâchera de limiter les appels à l'aide auprès du "service informatique".

- Les enseignants, ayant beaucoup moins de contact avec le système, seront moins regardants sur l'esthétisme mais exigeront du programme qu'il ne leur fasse pas perdre leur temps en redondances de saisies, bugs, et futilités. Leur utilisation se limite à la saisie de notes et à la consultation.

- Les étudiants, qui se connecteront pour voir leur notes dès leur disponibilité (en supposant que le format archaïque du papier accroché au mur ait disparu dans ce monde), pourront être tentés de modifier leurs notes. Cette fonction leur sera évidemment interdite, mais on pourrait émettre l'hypothèse que dans cette université il existe des pseudo-hackers souhaitant accéder illicitement à la base de donnée (SQL injection, etc...), la prise en charge de ce détail est cependant hors-sujet. Les étudiants sont limités à la consultation des listes d'étudiants et d'enseignants, des cours, et de leurs notes.

A priori on n'aura pas d'interaction directe à gérer entre les étudiants, enseignants et le responsable, mais on pourrait imaginer l'intégration d'une messagerie interne. Dans le programme l'utilisateur devra au préalable s'identifier, chaque étudiant ou enseignant aura son propre compte, étant donné que ce sont des individus reconnus par le système, avec des informations personnelles. Par contre un seul compte sera crée pour le ou les responsable, qui sera muni de tous les droits, par analogie avec l'utilisateur *root* d'un système Unix.

3.2 Conceptualisation et cas d'utilisation

3.2.1 Acteurs et signaux

On identifiera chaque classe d'individu à un acteur, on a trois acteurs donc : le responsable, l'enseignant, l'étudiant. On ajoutera aussi un acteur *John Doe* (nom utilisé pour les personnes inconnues, ici indescriptibles, n'appartenant à aucune catégorie définie), pour décrire les actions réalisables par une tierce personne n'ayant pas de droits d'accès au système.

Un cas d'utilisation du système par un acteur correspond à un ensemble de signaux d'interaction émis par l'acteur et la ou les réactions du système. Par exemple l'ensemble de signaux d'interactions { *cliquer sur le menu déroulant*

Students, selectionner add a student, remplir les champs, valider } correspond à l'écriture dans la base de donnée d'une nouvelle entrée, la réaction du système étant l'ajout dans sa mémoire des données entrées par l'utilisateur via le clavier pour l'entrée effective et l'écran pour le contrôle. L'utilisateur interagira directement via la souris et/ou le clavier. On suppose que les ensembles de signaux sont cohérents, et ont un objectif, par exemple le cas d'un utilisateur cliquant n'importe où n'importe quand n'a pas de sens, on pourra éventuellement traiter ce cas comme cas particulier. Le cas d'utilisation incohérente est détaillé plus loin dans l'analyse.

On peut diviser en trois classes les ensembles de signaux cohérents (vue des cas d'utilisation) :

- Entrée de données (écriture) : saisie par l'utilisateur, sauvegarde en mémoire du système.
- Accès à des données (lecture) : demande spécifiée par l'utilisateur, lecture et impression de données du système.
- Modification de l'état du système : l'utilisateur passe à la période suivante, éventuellement modification de données par le système, calculs.

Soit du point de vue des composants :

- L'utilisation de champs de saisie de texte et de boutons de validation (on utilisera ce modèle, le plus répandu, familier des utilisateurs).
- Le choix des données à afficher dans des menus déroulants, et l'affichage sous forme de tableau, triable selon divers critères.
- L'action d'un "super-utilisateur" de passer à la période suivante en cliquant sur le bouton destiné à cet effet.

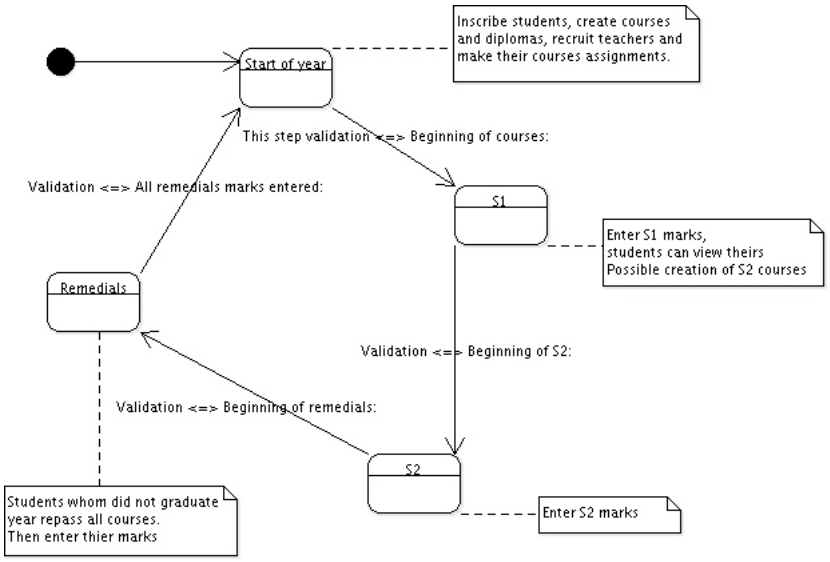
La classe d'utilisateur des responsables sera donc la seule à pouvoir modifier l'état du système, l'étudiant aura l'accès le plus restreint, on devra soit lui rendre impossible l'émission des ensembles de signaux correspondant aux actions interdites, soit rendre le système insensible à ces signaux. Celà correspondrait par exemple au choix entre rendre invisible certains menus, ou les griser. La seconde possibilité a l'inconvénient d'accroître la tentation et l'interrogation, il est donc plus *secure* de choisir la première.

Plus l'ensemble de signaux correspondants à une action donnée sera réduit, plus le système sera efficace : moins de signaux implique moins d'erreurs potentielles, moins de cas particuliers à gérer, moins de risques de failles de sécurité. Cependant nous verrons dans l'étude des cas d'utilisation que cette simplicité peut être source d'insécurité. Les choix d'éléments d'interface sont évoqués dans la partie "Interface utilisateur".

3.2.2 Etats et types d'utilisation

Le système pourra se trouver dans quatre états distincts, correspondant chacun à un ensemble de cas d'utilisations. L'état initial est bien sûr le premier chronologiquement, la pré-rentrée, unique période d'inscription d'étudiants.

`Diagramme d'états ( statechart )`

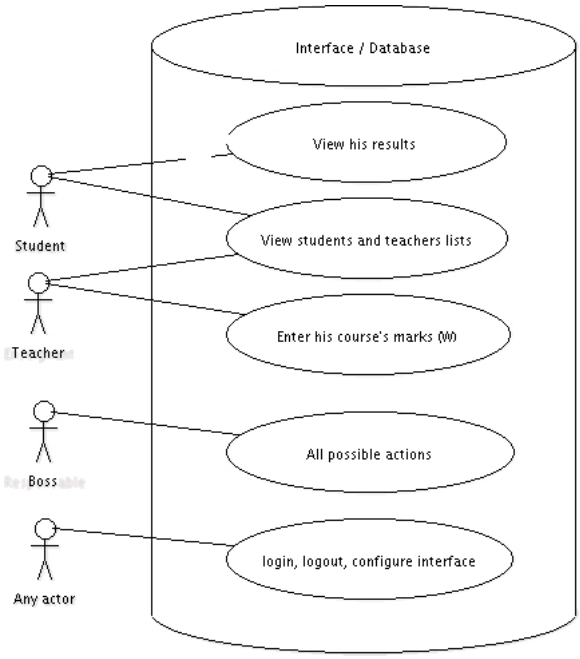


Chaque état ne peut donc être atteint que depuis son prédecesseur direct, l'interfaçage de cette procédure pourrait s'effectuer par un bouton particulier, ou un choix particulier parmi les autres actions. Le caractère irréversible de l'action implique des avertissements (messages, doubles validations...) pour éviter toute fausse manoeuvre. Ceci sera particulièrement important pour l'entrée des notes de la période courante, qui ne pourra être effectué dans un état ultérieur.

On distingue deux autres principaux types d'actions : entrer des informations (écriture dans la base), accéder à des informations (lecture seulement, accès à la base). Chaque type d'utilisateur peut aussi forcément se connecter à la base, mais on n'inclura pas cette action évidente à notre analyse, car elle est la condition *sine qua non* de toutes les autres actions, et n'est conditionnée par aucun état.

Les actions administratives sont limitées à la modification des privilèges d'un utilisateur, réalisables par tout responsable.

`Présentation simple des cas d'utilisation:`



3.2.3 Cas d'utilisation

Etat "pré-rentrée" (*Start of year*):

1. Responsable :
   1/ Entrer dans la base de données un nouvel étudiant, lui attribuant automatiquement un numéro d'étudiant unique (le SID, Student ID), et créant un mot de passe. Si l'étudiant est déjà inscrit, sa réinscription consistera à lui attribuer un ou des diplômes pour la nouvelle années scolaire.
   2/ Entrer un nouveau diplôme disponible aux étudiants.
   3/ Supprimer un diplôme, si aucun étudiant n'y est inscrit (on considèrera que le responsable agit de façon cohérente, et qu'il ne supprimera pas le diplôme avant de saisir les inscriptions. Si jamais cela arrive il pourra toujours recrée le diplôme).
   4/ Ajouter une matière, et éventuellement lui attribuer des enseignants en charge, et des cursus l'incluant.
   5/ Supprimer une matière, qui ne sera plus enseignée.

- Etudiant :
  Rien de significatif.
- Enseignant :
  Rien de significatif.

Etat "fin du premier semestre" (*End 1st semester*):

- Responsable :
  1/ Entrer les notes des étudiants, pour chacun de leurs enseignements du premier semestre. La valeur par défaut sera zéro. Ce sera donc la même sanction pour une absence que pour un résultat nul. La probabilité d'une absence à l'examen étant négligeable nous ne traiterons pas le cas d'un rattrapage d'un examen particulier, l'étudiant se présentera alors au rattrapage « de septembre » si nécessaire.
- Etudiant :
  1/ Consulter ses notes du premier semestre, si elles ont été enregistrées.
- Enseignant :
  1/ Entrer les notes de son enseignement, quels que soient les élèves.

Etat "fin du second semestre" (*End 2nd semester*):

- Responsable :
  1/ Entrer les notes des étudiants, pour chacun de leurs enseignements du second semestre.
- Etudiant :
  1/ Consulter ses notes du premier et second semestre, si elles ont été enregistrées.
- Enseignant :
  1/ Entrer les notes de son enseignement, quels que soient les élèves.

Etat "rattrapage" (*Passing exams period*): lors de la transition entre la fin d'année et le rattrapage le système aura calculé les moyennes des étudiants, en aura déduit s'ils ont validé leur année, et dans le cas échéant les aura inscrits au rattrapage. On ne gérera pas le cas d'étudiants dits "ajac" (grossièrement, dont l'année n'est pas validée mais pouvant tout de même accéder à l'année suivante).

- Responsable :
  1/ Entrer les notes des étudiants au rattrapage, pour chaque enseignement "rattrapé".
- Etudiant :
  1/ Consulter ses notes du premier et second semestre et du rattrapage, si elles ont été enregistrées.
- Enseignant :
  1/ Entrer les notes de son enseignement au rattrapage, quels que soient les élèves.

Une fois cet état validé, on passera au suivant, la pré-rentrée; les étudiants pourront avoir obtenu un diplôme, ou un droit d'accès dans l'année suivant de leur diplôme en préparation. Cependant leur réinscription n'est pas automatique. L'archaïque procédure d'inscription par minitel n'est pas gérée, la tâche en reviendra au responsable d'inscrire les étudiants. Toutefois un étudiant réinscrit n'est pas forcément un nouvel étudiant, s'il était déjà inscrit il conserve son SID, ainsi que son historique dans la base de donnée.

Pour tous les états:

- Reponsable :
  1/ Consulter la base de données, de toutes les années, pour tous les étudiants.
  2/ Supprimer un étudiant, soit le désinscrire, pour de multiples raisons; non-assiduité, fraude aux examens, décès, etc...
  3/ Modifier les privilèges d'un compte du système.
  4/ Ajouter un enseignant, lui attribuant un identifiant automatiquement et un mot de passe, ainsi que les opérations annexes (ajout et suppression d'enseignements - *Course* - pour cet enseignant).
  5/ Passer à l'état suivant, selon l'ordre défini plus haut dans l'analyse. L'action effective sera précédée d'un avertissement sur le caractère irréversible de l'opération.

- Etudiant :
  1/ Consulter ses notes de l'année, ses diplômes obtenus depuis son inscription dans l'établissement.
  2/ Consulter la liste des élèves et des enseignants.
  3/ Consulter la liste de diplômes et filières proposés par l'établissement.

- Enseignant :
  1/ Consulter les informations (dont les notes, de tout enseignement) des étudiants.
  2/ Consulter la liste de diplômes et filières proposés par l'établissement.

Pour toutes les entités, à n'importe quel instant :
1/ Se déconnecter, pour éventuellement permettre à quelqu'un d'autre de se connecter. Si la session est fermée brutalement (fenêtre fermée, machine éteinte), l'authentification sera demandée à la réouverture de l'interface.
Chaque utilisateur pourra personnaliser son interface, et sauver ses préférences (conservées pour les sessions suivantes), il sera alors libre de:
2/ Modifier la langue de l'interface.
3/ Modifier son mot de passe.
4/ Modifier l'apparence (*skin*).

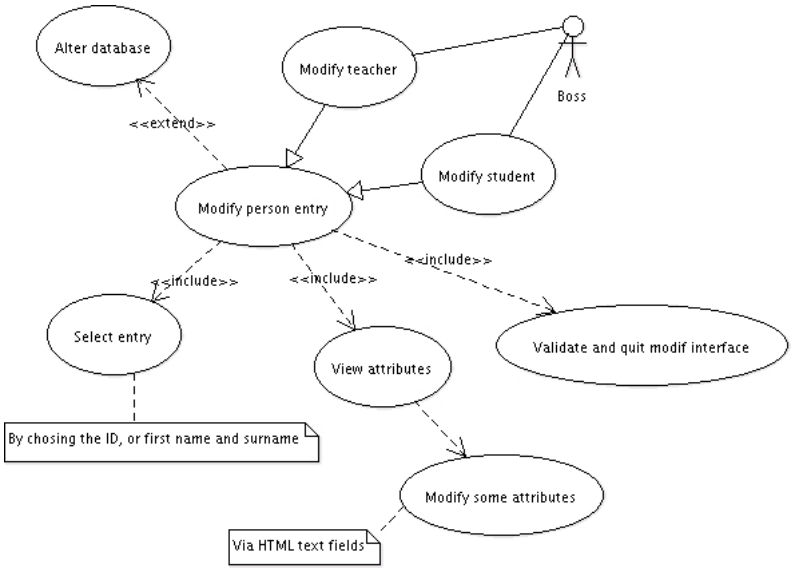
Identification Le login de tout utilisateur sera son identificateur, suite de huit chiffres pour les étudiants et enseignants, respectivement avec '1' et '2' comme premier caractère.

Les mots de passe sont générés par cryptage (algo DES) de la chaîne concaténée du nom et du prénom de l'étudiant, avec comme "sel" le numéro de l'étudiant, ce qui garantie l'unicité des mots de passe. La longueur de la chaîne est de huit caractères, pour à la fois être aisément mémorisable, et garantir un minimum de sécurité, ce compromis étant assez subjectif...

Validation La validation concerne les étudiants, et conditionnera leur obtention du diplôme préparé. On choisit d'appliquer la règle suivante : si la moyenne des notes est supérieure ou égale à 10 (les notes étant positives, sur 20), l'année est validée, sinon l'étudiant est ajourné pour ce diplôme. Les notes dans une matière peuvent avoir trois statuts :

- Facultative : cette note est comptabilisée dans la moyenne seulement si elle est supérieure ou égale à 10.
- Eliminatoire : si on obtient moins de 10, l'année ne peut être validée, aurait-on 19 de moyenne.
- Standard : la note est comptabilisée dans la moyenne sans condition, quelle que soit sa valeur.

`Exemple de cas d'utilisation:`



3.2.4 Acteur John Doe - utilisation anormale

Utilisation illicite Deux cas se présentent:

- John Doe n'a pas de couple login / password.
- John Doe possède un couple login / password, qu'il a "emprunté" à un utilisateur.

Dans la première situation, John Doe pourra effectuer des tentatives aléatoires de se connecter, si elles se révèlent infructueuses il essaiera sans doute d'accéder directement à une page en recherchant directement une URL ; par exemple si cherche à atteindre la page *index2.php*, lui sera affichée la page d'erreur de mot de passe, et le code source de la page ne l'informera pas plus. Il pourra tenter d'afficher la page *addstudent.php*, mais rencontrera une erreur de type 401, car il ne s'est pas identifié.

Bien évidemment nous prenant seulement en compte le cas d'un utilisateur voulant "faire le malin", et pas un "vrai hacker". De plus notre implémentation de l'identification par mot de passe n'est certainement pas un modèle de sécurité.

Dans la seconde situation, l'interface n'a pas les moyens de vérifier si l'utilisateur se fait passer pour un autre et sera donc vulnérable. Le seul moyen est d'informer l'utilisateur du caractère précieux de ses informations de connexion, et l'inciter à ne pas les exposer au grand jour.

Utilisation incohérente On qualifiera d'incohérente une utilisation du programme où les signaux émis par l'utilisateur ne répondent pas à un objectif particulier, le cas extrême pouvant être simulé par une machine émettant des signaux aléatoires (clic de souris, pressions sur des touches du clavier) à des instants aléatoires. Pratiquement une utilisation incohérente pourra survenir lors de signaux émis involontairement (on s'assoit sur le clavier, déplace la souris sans s'en rendre compte), de réaction agressive de l'utilisateur face à l'incomprehension du système, du manque de patience lors d'un calcul de la machine paralysant le système, ou plus anecdotiquement après un pot bien arrosé...

Le faible nombre d'interaction pour réaliser une action donnée (inscrire un étudiant, supprimer un enseignant, etc...) offre une certaine simplicité d'utilisation et une plus grande efficacité, mais dans le cas d'une utilisation incohérente augmentera les risques d'actions indésirées. Par exemple il peut suffir de deux clics pour supprimer un étudiant. Rajouter des avertissements du type *"Êtes-vous bien certain d'être sûr de vraiment vouloir réaliser cette action ?"* ne ferait que déplacer le problème, et agaçerait sans doute très rapidement les utilisateurs quotidiens du programme.

On paye donc la simplicité par une augmentation des risques lors d'utilisation incohérente, il sera alors bon de mettre en garde les utilisateurs.

Tentative d'accès illégal Si un utilisateur quelconque tente d'accéder directement à une page, sans passer par l'étape de l'authentification, il sera confronté à la page d'erreur 401, correspondant à un accès non autorisé. De plus nous n'utilisons pas d'adresse contenant le nom d'un fichier, qui pourraient poser des problèmes de sécurité.

4 Modélisation statique orientée objet

Une fois les cas d'utilisation définis, on peut formaliser le système par la vision de ses objets, leurs composants et relations, à l'aide des sémantiques UML de modélisation statique, et notamment celles du diagramme de classes, duquel découlera directement l'implémentation, celle-ci étant orientée objet avec PHP.

Les classes retenues sont les suivantes (on note *table* quand elles correspondront à une table de la base de données, celle-ci sera détaillée dans un paragraphe suivant):

Career (*table*) : Les filières d'études (informatique, sciences humaines, etc..), chacune munie d'un identificateur dans la table.

Course (*table*) : Un cours, identifié par un ID, qualifié par un titre, et pouvant être associé à plusieurs couples Career/Diploma, avec pour chaque couple un coefficient, et la période (par exemple un cours de calculabilité, pour le diplôme master I, de la filière informatique, au premier semestre).

Cursus (*table*) : Le cursus d'un étudiant, soit l'ensemble des diplômes obtenus depuis sa première inscription à l'université.

Diploma (*table*) : Les différents titres pouvant être obtenus à la fin d'une année d'études, indépendamment de la filière concernée (par exemple pour la licence on aura les diplômes licence I, licence II et licence III).

Mark (*table*) : Une note, ayant pour attributs une valeur, une période (par exemple semestre 1 de l'année 2046), et un enseignement.

Person (*table*) : Surclasse de Student et Teacher, la table contenant les informations propres à une personne; état civil et coordonnées.

Privileges (*table*) : Les types de privilèges accordés aux utilisateurs, de trois types (responsable, étudiant, enseignant).

Sid : Identificateur d'un étudiant, cette classe définira des opérations relatives à un étudiant car on pourra identifier l'étudiant à son Sid.

Tid : Analogue au Sid, pour un enseignant.

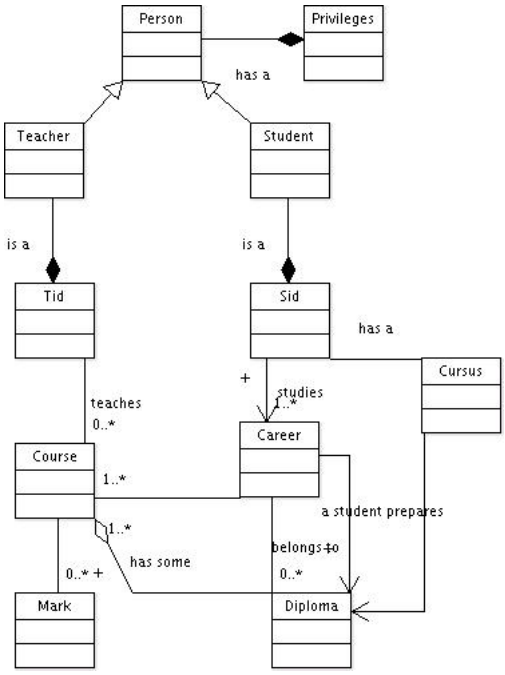
Student (*table*) : Etudiant de l'établissement, identifié par son numéro d'étudiant (Sid). La table contiendra les identificateurs de l'étudiant, de ses diplômes et de leur filière, ainsi que son numéro de groupe.

Teacher (*table*) : Enseignant de l'établissement, identifié par son numéro d'étudiant (Tid). La table contiendra l'identificateur de l'enseignant, ceux de ses enseignements (Course), et le nombre d'heures effectuées pour chaque enseignement.

On crée des classes Sid et Tid respectivement pour les identificateurs (clés primaires dans leurs tables respectives) d'un étudiant et d'un enseignant, et pas pour les identificateurs des diplômes ou d'une matière (Course), d'une part car étudiants et enseignants sont des acteurs internes du systèmes, jouent plus de rôles qu'un diplôme, et d'autre part, et surtout, car ces identificateurs sont explicites dans l'environnement, c'est à dire qu'un étudiant connaît son Sid, de même un enseignant doit savoir son "numéro de matricule", on peut alors faire une analogie avec le modèle objet en qualifiant ces identificateurs de publics, contrairement aux autres, seulement utilisés pour la conception et l'implémentation, mais dont l'existence même reste inconnue "en pratique".

4.1 Diagramme de classe

Présenté sans les attributs et méthodes pour une meilleure lisibilité :



4.2 Détails d'associations

4.2.1 Classes Sid, Career, Cursus, Diploma, Mark, et Course

On identifie l'étudiant à son Sid (unique). Chaque étudiant, pourra être inscrit à une ou plusieurs filières (*careers*). Par hypothèse on considère comme étudiant tout individu inscrit à au moins une filière, le cas échéant l'individu ne sera pas reconnu comme un étudiant par le système, même si il a auparavant étudié dans l'établissement. Une filière vue comme liée à un étudiant permet à ce dernier de préparer un seul diplôme de cette filière. Vu au sens le plus large, une filière sera composée de plusieurs diplômes (par exemple les trois premières années de licence, puis deux années de master). Dans tous les cas, une année de préparation à un diplôme possèdera un ensemble non-vide d'enseignements, chacun ayant un ensemble de notes pour chaque étudiant inscrit à ce diplôme. De plus chaque enseignement (*course*) est lié à une filière (l'enseignement "macroéconomie" appartiendra à la filière "economie et gestion", et seulement à celle-ci), chaque filière ayant en général plusieurs enseignements. De plus chaque Sid possède un cursus retraçant son parcours, soit l'énumération des diplômes qu'il a obtenus.

4.2.2 Classes Person, Student, Teacher, Privileges

Pour chaque étudiant et enseignant on doit connaître son état-civil et ses coordonnées, propres à toute personne, on crée donc naïvement une classe *Person* pour stocker nom, prénom, addresse, code postal, date et lieu de naissance. Nous aurions très bien pu intitulé cette classe User, car on associe à chaque personne reconnue par le système un compte utilisateur, spécifié dans la classe *Privileges*, composée de *Person*. On y recencera les informations propres au compte utilisateur et aux privilèges : mot de passe, langue de l'interface, apparence, et niveau de privilège, l'association utilisée doit être une composition. L'indexation étant réalisée sur l'identificateur de la personne, soit le *Sid* d'un étudiant, et le *Tid* d'un enseignant.

4.2.3 Association Sid / Student et Tid / Teacher

On identifie l'étudiant et l'enseignant respectivement à leurs numéros d'identification unique, Sid et Tid. Les fichiers sid.php et tid.php définiront toutes les opérations propres à un étudiant et à un enseignant. On garantit l'unicité des Sid, mais pas des Students, dans le cas d'homonymie et de même inscriptions, de même pour les enseignants.

Cette association est donc une agrégation forte (les cycles de vie sont liés), on utilise donc l'association de composition dans le diagramme de classes.

5 Base de données

5.1 Aspects techniques

Nous utilisons une base de donnée du fournisseur d'accès Free de type MySQL (version 4.0.20), avec laquelle on interagit grâce au langage PHP4. Une connexion typique à la base s'effectuant à l'aide des commandes:

```
include "config.php"; // dans ce fichier les valeurs des trois variables ci-dessous
$dblink=mysql_connect($servername,$username,$password);
if (!$dblink)
{
die("Connection error :".mysql_error());
}
mysql_select_db($database,$dblink);
```

Et une requête suivant cette connexion sera de la forme:

```
$query=mysql_query(< la requête SQL >,$dblink);
```

Plus de détails dans la documentation technique de l'application.

5.2 Conception et optimisation des tables

Chaque classe correspond à une table de la base de donnée, exceptées Sid et Tid, car elles sont implicitements utilisées en tant qu'identificateur des étudiants et des enseignants.

Pour les identificateurs publiques, Sid et Tid, à huit chiffres, on utilisera le type SQL *int*, mais pour les identificateurs de cours et filières un *tinyint unsigned* sera largement suffisant (jusqu'à 255 entrées).

Spécification des tables utilisées, dépendances fonctionnelles et normalisations: Pour simplifier on notera "DF" pour "dépendance fonctionnelle", "FN" pour "forme normale" (et par extension "3FN" pour troisième forme normale, "BCFN" pour "forme normale de Boyce-Codd-Kent", etc...), la liste des attributs correspondra aux éléments entre parenthèses, la clé primaire étant précédée d'un astérisque.

Career *(*ID, title) DF: ID –> title BCFN.* Présence de DF élémentaires. L'intérêt d'avoir une BCFN permet l'élimination sans condition des redonances et répétitions.

Course *(*ID, title, career, diploma, career, coef, period) DF : ID –> title, career, diploma, coef, period.* BCFN car les attributs dépendent de la clé primaire, représentant une DF élémentaire. Cela permet l'unicité de chaque entrée dans la table, bien qu'une multivaluation des données soit possible.

Cursus : *(student, yearn diploma, career) DF : student, year -> diploma, career.* 3FN. La clé étant multivaluée la table n'est pas en 4FN.

Diploma *(*ID, title, greatest) DF : ID –> title, greatest. BCFN (raison : cf Course).* Unicité des entrées.

Mark *(*ID, *course, *period, value) DF : ID, course –> value.* BCFN.

Person *(*ID, sname, fname, addr, pcode, dob, pod) DF : ID –> sname, fname, addr, pcode, dob, pob.* BCFN car on a un ensemble de DF élémentaires garantissant l'unicité des entrées.

Place *(*postalcode, town) DF : postalcode –> town.* Voir Career

Privileges *(*ID, password, lang, skin, level) DF : ID –> password, lang, skin, level.* BCFN, ensemble de DF élémentaires garantissant l'unicité de la clé primaire.

Student *(*ID, *diploma,*career, group) DF : ID, diploma, career –> group.* 3FN : L'attribut group dépend pleinement des trois clés.

Teacher *(*ID, *course, hours) DF : ID, course –> hours.* Voir Student

Year *(*currentyear) DF : aucune.* 5FN : tous les attributs sont des clés, pas de multivaluation ni naturelle ni par jointure.

YearPeriod *(*ID, year, period) DF : ID –> year period.* BCFN : cf. Privileges.

Préservation de l'intégrité référentielle: Lors de la suppression d'une entrée dont la clé est clé étrangère d'une autre table nous serons tenu d'effacer le tuple contenant la clé étrangère afin de préserver toutes les références intactes. Par exemple la suppression d'un diplôme implique la suppression de ses enseignements, et de leurs assignements aux enseignants. De même, un niveau au dessus, supprimer une carrière en conservant ses diplômes et enseignements rompt l'intégrité de la base.

6 Interface utilisateur (IHM)

6.1 Conception - aspect technique

L'interface réalisée en HTML est relativement simple, s'adaptera facilement à n'importe quel naviguateur, et les modifications visuelles seront très min-

imes d'un naviguateur à un autre (design des menus, marges). Le choix de l'apparence revient à choisir un fichier *css* définissant les caractèristiques des éléments graphiques (couleurs, type et taille des fontes), et les différentes langues correspondent simplement à choix du fichier consignant les variables de type *$TXT_**, définissant les parties textuelles de l'interface. Plus de détails dans la documentation technique.

6.2 Réalisation - aspect utilisateur

Nous avons opté pour une interface "tout en un", c'est à dire que la seule fenêtre principale permet d'accéder à toutes les fonctionnalités, en l'occurence via les menus déroulants. Ainsi l'utilisateur n'aura pas à naviguer dans une arborescence de sous-menus, et y gagnera en rapidité de maîtrise du logiciel, la barre de menu étant toujours présente il ne sera pas dérouté et pourra toujours "retomber sur ses pieds" en cas de mauvais choix. La barre de menu horizontale sera placée en haut de l'écran, ainsi l'utilisateur habitué aux logiciels courants de bureautique ne sera pas surpris de voir les contrôles du programme à cet endroit.

Comme objets réalisant la sélection nous avons utilisés les menus déroulants, dont l'avantage principal est la faible occupation d'espace, et l'utilisation intuitive. Leurs intitulés décrivent leurs fonctions, ainsi le menu Students permets d'accéder à toutes les fonctions directement relatives aux étudiants, Teacher pour les actions directement relatives aux enseignants, Studies pour les actions relatives aux enseignements et filières, et Preferences pour les options de personnalisation l'interface (mot de passe, langue, *skin*).

La possibilité de personnaliser l'interface sera appreciée d'une part pour la convivialité qu'elle apporte, et aussi car un utilisateur préfère toujours avoir le choix entre plusieurs apparences, rien que pour l'aspect ludique et découverte de toutes les essayer, même si il conserve celle par défaut. Moins superflus sont les modifications de la langue de l'interface et surtout du mot passe.

7 Conclusion

Après cette première expérience d'analyse et programmation nous avons pu réaliser l'importance de l'analyse pré-implémentation, et l'attention à porter à sa rigueur. La difficulté étant peut-être le niveau d'abstraction, qui fait qu'après avoir programmé une partie du logiciel on se représente mieux les composants d'un point de vue analyse. De plus la démarche incrémentale n'est pas forcément triviale, dans la mesure où il faut savoir différencier chaque niveau de détail, établir une hiérarchie des besoins, et ne pas se lancer dans un point particulier du programme simplement parcequ'on a "des idées".

A l'échéance, les principaux objectifs que nous nous étions fixés ont été respectés, les fonctionnalités étant certes limitées, mais l'ensemble est fonctionnel,

cohérent, aucun bug majeur n'a été constaté, et la prise en main par un utilisateur lambda serait certainement aisée. Réaliser un gros logiciel aux multiples fonctionnalités, mais inutilisable par les utilisateurs ciblés constitue un échec.

L'estimation du temps de travail nécessaire n'a pas été dépassée, et nous n'avons pas souffert de difficultés d'organisation particulières, n'étant que deux à travailler sur ce projet.

Credits

L'application a été réalisée sur les systèmes Mandrake 10.0 GNU/Linux, Debian GNU/Linux Sarge 3.1, MacOSX, et testé sur des versions récentes des naviguateurs Firefox, Konqueror, Safari, Internet Explorer. L'installation en local a été testée sur Debian, MacOSX, et Microsoft Windows XP avec plus ou moins de facilité (cf.User's guide). Pour les modélisations UML nous avons utilisé argoUML, pour l'édition des sources : JEdit 4.2, kwrite, emacs et vi. Les documents ont été tapés dans l'éditeur Latex wysiwyg Lyx 1.3.3, la conversion en pdf effectuée par pdflatex 3.14159-1.10b, pour la visualisation on déconseille xpdf ou gnome pdf viewer. Le programme restera installé à l'adresse suivante, afin d'en avoir un aperçu sans être obligé d'installer en local :
http://jpnossa.free.fr/files/maitriseInfo/Objet/projet_04_05/ums/ , l'accès restera restreint jusqu'à la fin des soutenances du projet.
Pour tout contact, critique, proposition d'embauche, donation, etc...nos emails sont en entête de ce fichier ;)

References

[1] L.Atkinson, *La programmation en PHP4*, Campus Press

[2] M.Dreyfus, *Livre d'or HTML4*, Sybex

[3] *http://uml.free.fr*