# MyDOX : Aide à la classification de documents hétérogènes

**Sujet Proposé par :** Talel Abdessalem

**Réalisé par:**

Rana Bou Jaoudé
Nicolas Boulanger
Adrien Missemer
Céline Monthéard

**Troisième Revue**
**27 Février 2004**

***Résumé :***

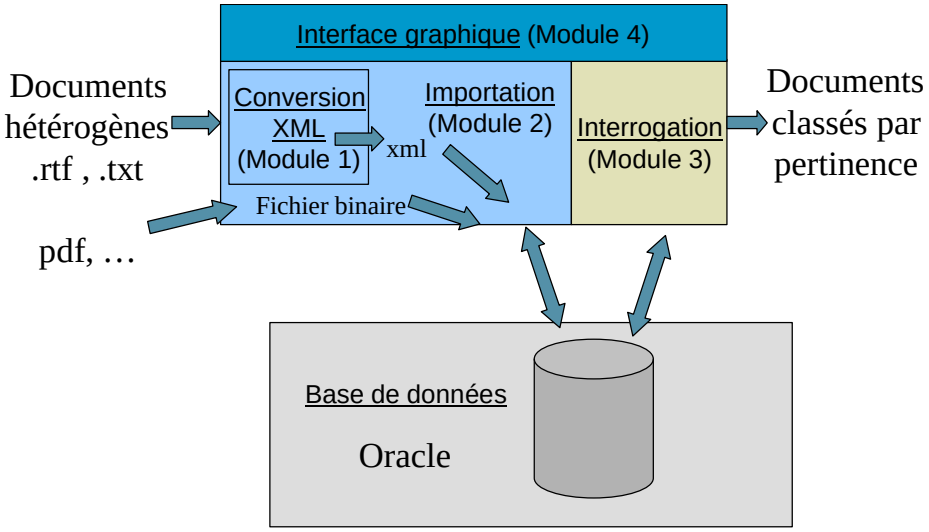
*Dans cette troisième revue, nous effectuons une description globale de MyDOX, puis nous décrivons plus en détail les différents modules de MyDOX (convertisseur en XML, constitution de la base de données, interpréteur pour effectuer les requêtes sur la base de données, noyau et interface utilisateur). Enfin, nous mettons à jour la planification du projet.*

# Table des matières

# I. Description globale de MyDOX

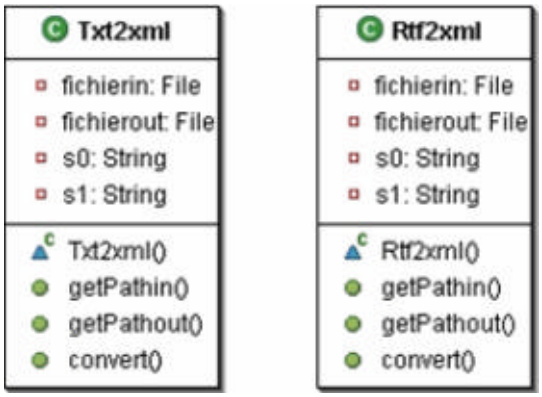
Nous rappelons que l'objectif de ce logiciel est de pouvoir constituer une liste de documents hétérogènes (.rtf, .txt) et d'effectuer des recherches par mot-clé sur cet ensemble de documents. Ainsi, l'utilisateur se constitue sa propre méta-base de données et dispose d'un moteur de recherche personnel pour effectuer des recherches sur celle-ci.

Le premier module a pour rôle de convertir des documents .rtf ou texte en XML. Le deuxième module est chargé de l'importation des fichiers, en particulier il appelle le module conversion pour convertir les fichiers rtf et txt en XML avant d'insérer le contenu XML dans la base de données, ou bien il insère directement les fichiers binaires (Oracle possédant des filtres pour de nombreux types de fichiers). Le troisième module interprète une requête de l'utilisateur et sélectionne les documents correspondant à la requête, classés par ordre de pertinence. Enfin le quatrième module est l'interface graphique.




# II. Conversion en XML

Pour convertir les documents .rtf ou .txt nous utilisons respectivement les classes Rtf2xml et Txt2xml, dont la modélisation est la suivante :



## II.1. Conversion de .rtf en XML

Pour convertir les documents .rtf en XML, nous utilisons la classe Rtf2xml, qui fait appel au logiciel Majix. Ce logiciel est entièrement écrit en java et nous disposons d'une licence MPL1.1 (Mozilla Public License 1.1) qui nous permet de l'utiliser. Nous décrivons ci-dessous les variables et méthodes de la classe Rtf2xml.

Le constructeur de la classe Rtf2xml prend comme paramètre la variable fichierin. Cette variable est une instance de la classe File, elle représente le document .rtf. Ce constructeur instancie également les variables membres s0, fichierout et s1. Le String s0 est initialisé par le path du fichier fichierin, le String s1 est initialisé par s0, mais en remplaçant les 3 derniers caractères 'rtf' par 'xml'. Le File fichierout est initialisé par un fichier vide de path s1.

Les méthodes de la classe Rtf2xml sont **getPathin()** (elle retourne s0), **getPathout()** (elle retourne s1) et **convert()**. La méthode convert() sert à effectuer la conversion. Les autres méthodes qui renvoient les path du disque dur sont utilisées par la base de données pour avoir accès aux fichiers non modifiés (par exemple pour savoir si une mise à jour est nécessaire).

La méthode convert() utilise une instance de la classe Runtime qui nous permet d'accéder à l'environnement d'exécution. A partir de Runtime, nous lançons une instance de Process afin d'exécuter Majix dans un processus séparé. Ce processus retourne un premier document XML, qui suit la dtd et la xsl de Majix. Ensuite, en utilisant des instances de BufferedReader et BufferedWriter (méthodes de traitement de chaînes de caractères), nous supprimons la xsl fournie par Majix. La méthode convert() modifie donc le File fichierout converti en xml suivant la dtd de Majix, mais sans xsl.

## II.2. Conversion de .txt en XML

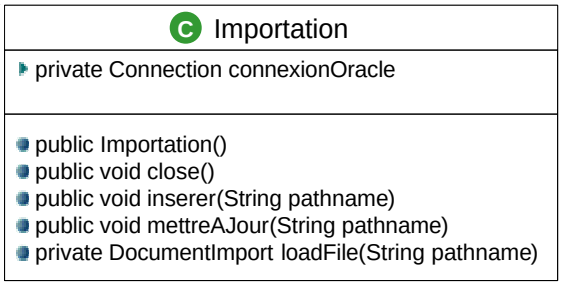
La conversion des documents .txt en XML utilise la classe Txt2xml. Son constructeur, et ses méthodes **getPathin()** et **getPathout()** sont similaires à ceux de la classe Txt2xml.

La méthode **convert()** de Txt2xml transforme le File fichierin en un File fichierout. Fichierin est converti en XML suivant la même dtd que les fichiers convertis par Rtf2xml. Contrairement à la méthode convert() de Rft2xml, nous codons cette méthode sans faire appel à des logiciels extérieurs. Nous utilisons des méthodes de traitement de chaînes de caractère, afin d'ajouter les balises au document .txt (balises <info> <title> … < /title></info>,  …  ou ).

Les documents .txt convertis en XML suivent la même dtd que les douments .rtf convertis en xml.

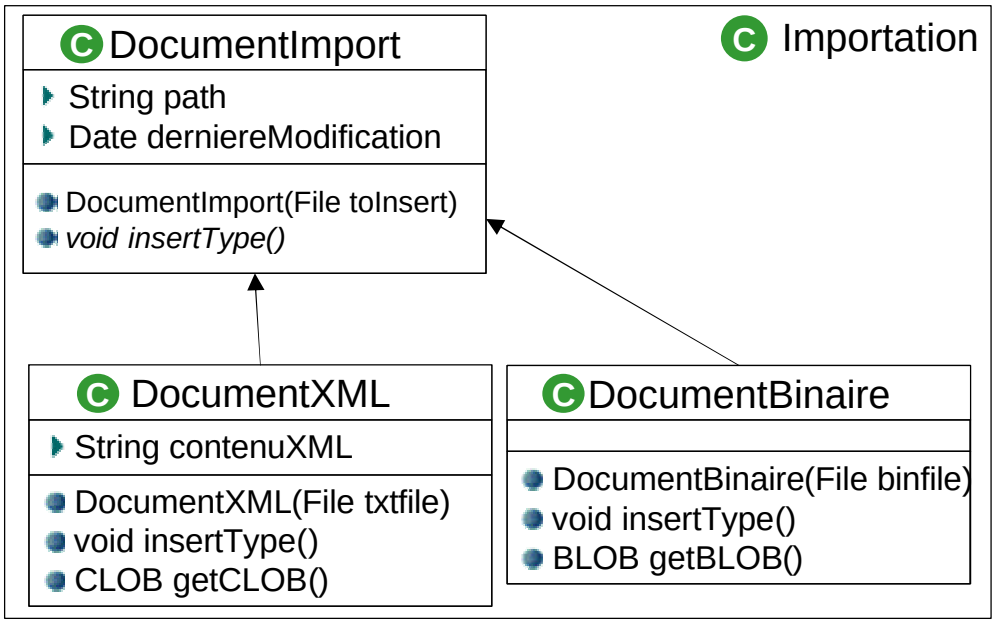
# III. L'importation des documents

La classe principale a pour interface :



Pour l'utilisateur, la partie utile est donc réduite à sa plus simple expression : `inserer(nom_du_document)` ou `mettreAJour(nom_du_document)`. Pour `inserer()` une exception est levée au cas où le fichier est déjà dans la base. Ce sera alors au module interface de proposer une mise à jour du contenu de la base.

De façon interne, la classe Importation contient plusieurs sous-classes :




La classe `DocumentImport` est une classe abstraite, parente de toutes les classes représentant des documents insérables dans la base de données. Les documents convertibles en XML seront chargés dans des objets de classe `DocumentXML` et les autres dans des objets de classe `DocumentBinaire`.

La méthode `insertType()` (abstraite dans `DocumentImport`, définie dans les sous-classes) demande l'insertion du document dans la base.

# IV. La base de données

| ***NDOC*** *Integer* | URL *Varchar(256) unique* | CONTENT *Clob* | MODIFIED *Date* | TREE *Varchar(256)* |
|---|---|---|---|---|
| | | | | |

Clés

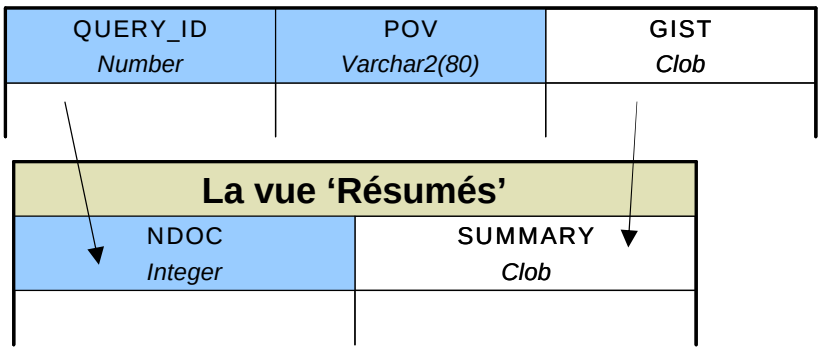
Arborescence virtuelle

La table des documents possède 2 clés : NDOC est la clé primaire, et est déterminée automatiquement par Oracle à l'insertion de chaque document (par l'utilisation d'un déclencheur et d'une séquence qui sert de compteur). L'URL est utilisée comme une seconde clé sur la table.

A chaque fois qu'un document est inséré, on pourra laisser la possibilité à l'utilisateur de spécifier un chemin virtuel dans lequel il veut ranger son document. Ce chemin est stocké dans le champ **TREE**. Il est virtuel dans le sens où il ne correspond à aucun emplacement

physique, ni sur le disque, ni dans la base. Il reste néanmoins possible de trier la table selon l'attribut **TREE**, ce qui permet de lister cette arborescence virtuelle. On pourra également récupérer l'ensemble des documents se trouvant dans tel ou tel dossier virtuel. On voit donc que cet attribut permet de hiérarchiser la base de documents.

Oracle Text Server propose une fonctionnalité de résumé, à travers la procédure `ctx_doc.gist`. Celle-ci stocke le résumé dans une table devant contenir les 3 attributs suivants : QUERY_ID, POV et GIST. A partir de cette table, on définit la vue `RESUMES(NDOC,SUMMARY)`, dans laquelle `NDOC` est encore une clé primaire, et référence le `NDOC` de la table des documents.




On rappelle que l'indexation des documents est gérée automatiquement par Oracle TextServer. Elle repose sur 3 tables :

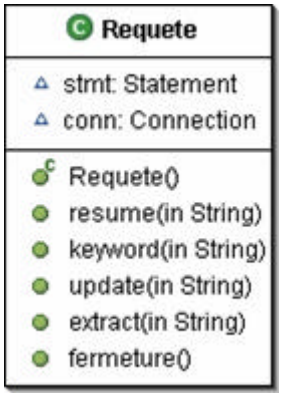
- Table des mots (comporte le mot, son n° et son type pour l'indexation(STOP ou PASS))
- Table de localisation (comporte le n° du mot et une chaîne de bit qui indique la présence ou non du mot dans le document)
- Table annexe (comporte le n° d'un mot, le n° d'un document dans lequel il apparaît, le nombre d'occurrences du mot dans le document, et une indexation fine au sein du document)

…et sur des algorithmes spécifiques :

- STOP Mode (indexation de tous les mots sauf des mots préalablement notés STOP, ou notés dans la STOP List)
- PASS Mode (indexation des mots notés PASS)
- Column specific (indexation avec différenciation sur les colonnes de la table de document en plus de la différenciation ligne)

# V. Requêtes sur la base de données

Pour effectuer les requêtes, nous utilisons la JDBC (*Java Data Base Connectivity*). Elle permet à tout programme Java d'accéder à une base de données relationnelles. L'API JDBC est implémentée au sein du package `java.sql`. Les requêtes sur la base de données se feront par l'appel d'instances de la classe Requete.




Cette classe possède comme variable membre une connexion à la base de données Oracle et un statement qui nous sert de support pour effectuer des requêtes sur la base de données et obtenir les résultats des requêtes. La classe Requete possède des méthodes de classe qui permettent d'effectuer différents types de requêtes sur la base de données :

- La méthode **keyword(String mot)** permet d'effectuer une requête de recherche par mot-clés sur la base de données. Elle lit le mot à rechercher, tapé par l'utilisateur, le traduit en requête (select ndoc from documents where …), effectue cette requête, puis traduit le résultat de la requête en un tableau de String. Elle retourne ce tableau de String (chaque String représente le path d'un document et des paramètres qui décrivent la pertinence de ce document).
- La méthode **resume(String resu)** lance une requête de recherche par mots-clés contenus dans le resumé d'un certain document de la base de données et retourne un tableau de String. Elle est implémentée de manière similaire à keyword(String).
- La méthode **update(String urldufichier)** permet d'effectuer une requête sur la base de données pour connaître la dernière date de modification du document dans la base de données, compare cette date avec la dernière date de modification du document sur le disque dur. Cette méthode renvoie le booléen 'false' si la dernière date de mise à jour est postérieure ou égale à la date de la dernière modification sur le disque dur, elle retourne 'true' dans l'autre cas (une mise à jour sera alors nécessaire).
- La méthode **fermeture()** ferme le statement puis la connection à la base se données.

# VI. L'interface utilisateur

L'interface utilisateurs permettra d'effectuer différentes opérations :

- Importer des documents dans la base de données (cette opération fait appel aux deux premiers modules).
- Effectuer des recherches par mot-clé, ou sur les résumés (appel du troisième module).
- Mise à jour de la base de données (Pour savoir si la mise à jour est nécessaire : appel de la méthode update de la classe Requete, si la mise à jour est nécessaire un appel aux modules 1 et 2 sera effectué).

# VII. Mise à jour de la planification

Le nombre d'heures de travail estimé et passé pour l'ensemble du groupe et pour les différentes phases du projet est représenté par le tableau qui suit : (en heures / groupe)

| Etape | Temps estimé | Temps passé |
|---|---|---|
| Planification | 25 | 25 |
| Analyse | 90 | 80 |
| Conception | 155 | 90 |
| Codage & Tests | 135 | |
| TOTAL | 405 | |