

# PROJET FIL ROUGE 2020-2021

## I- Qu'est-ce que le projet « Fil Rouge » ?

Il s'agit d'un projet transversal ayant un double objectif :

- valider des **compétences scientifiques et techniques** liées aux concepts, méthodes, et langages abordés en 1<sup>ère</sup> année de la spécialité Systèmes Robotiques et Interactifs (SRI).
- valider également des **compétences générales** en lien avec la **gestion de projet** aussi bien sur le plan technique (différentes phases de mise en œuvre d'un projet) qu'en terme d'**organisation** (comment fonctionner en groupe et avec le client) et de **communication** (savoir restituer/présenter son travail).

### I-1. Des compétences scientifiques et techniques...

Le détail des compétences techniques mises en jeu dans ce projet, ainsi que les noms des intervenants sont donnés ci-dessous (voir tableau 1).

Tableau 1 : Compétences techniques

UE et Semestres Concernés	Matières – Compétences Objectifs généraux	Enseignants & Heures associées
S5 - Informatique	Programmation en C, commandes et script UNIX (réutilisation de fonctions développées en TP – Gestion d'une pile – Manipulation de fichiers, lecture-écriture formatée, manipulation de chaînes de caractères...) – Traitement de base de documents texte, image et son	Isabelle Ferrané & Julien Pinquier <i>heures projet + travail personnel</i>  Julien Vanderstraeten <i>TD dédiés</i>
S5 - Outils de modélisation informatique	Modélisation UML Mise en pratique UML – diagrammes de classes et de séquences	Christelle Chaudet <i>Application du cours</i>
S6 Gestion de Projet	Spécification, conception, développement, tests et intégration	Julien Vanderstraeten <i>heures projet + travail personnel</i>
S6 Ingénierie des systèmes	Sûreté de fonctionnement - Développement de logiciels sécuritaires	Julien Vanderstraeten <i>Application du cours</i>
S6- Conception orientée objets	Programmation Objet (java, processing) Application directe des concepts Design pattern Modèle-Vue-Contrôleur	Christelle Chaudet & Julien Vanderstraeten <i>Application du cours + travail personnel</i>
S6- Introduction aux Systèmes Robotiques et Interactifs – partie IHM	Prototypage rapide, processing, ... Application directes des concepts IHM	Philippe Truillet <i>Application des concepts + travail personnel</i>
S6- Introduction aux Systèmes Robotiques et Interactifs – partie Robotique, Image, Son	Arduino, capteurs, traitement image, traitement audio, ...	Michel Taix & Michaël Lauer <i>TP dédiés + heures projet + travail personnel</i>

### I-2. ... aux compétences plus générales ?

Ce projet est l'occasion de décloisonner les enseignements et de faire le lien avec d'autres domaines de compétences (voir tableau 2). Dans ce contexte, l'organisation de l'équipe, la gestion du projet, la qualité des présentations

écrite/orale seront prises en compte dans la notation, selon une grille qui vous sera communiquée par les intervenants et mise à disposition sous Moodle.

Tableau 2 : Compétences générales

UE et Semestres Concernés	Matières – Compétences Objectifs généraux	Enseignants & Heures associées
S5 - Sciences humaines et sociales	S'organiser pour réaliser un projet en équipe Savoir convaincre un jury, des clients Réaliser un retour d'expérience et capitaliser ses compétences (techniques et non techniques).	Geneviève Estadiou <i>Cours et TD de communication (S5) + coaching (S6)</i>

### I-3. Organisation générale et évaluation

Ce projet est organisé en 3 parties ou phases, chacune ayant pour objectif de mettre en pratique, d'approfondir et de valider des compétences bien précises (voir tableau 3). La constitution des groupes devra être la même sur les parties I et II (celles-ci étant fortement liées) et pourra changer sur la partie III.

Tableau 3 : Organisation du projet

Décomposition du projet	Focus	Points faisant partie de l'évaluation
Partie I : Indexation de documents et moteur de recherche	Spécifications Développement en langage C, commandes et scripts Unix Programmation modulaire Traitement de documents texte, image et son Intégration, test et validation	<ul style="list-style-type: none"> <li>Document de spécifications</li> <li>Validation sur machine</li> <li>Implication étudiant</li> <li>Respect des contraintes</li> <li>Communication interne/externe</li> <li><b>Rapport/Bilan sur la partie I</b> (Parties technique &amp; communication)</li> </ul>
Partie II : Interface utilisateur pour la recherche d'informations via plusieurs moteurs de recherche	Prototypage de l'interface Interfaçage langage C et java	<ul style="list-style-type: none"> <li>Démonstration</li> <li>Implication étudiant</li> <li><b>Présentation sur la partie II</b> (Parties technique &amp; communication)</li> </ul>
Partie III : Application à la robotique	Développement de fonctionnalités, en lien avec la Robotique	<ul style="list-style-type: none"> <li>Démonstration</li> <li>Présentation</li> <li>Qualité du code</li> </ul>

### I-4. Objectifs à atteindre à l'issue du projet FIL ROUGE

L'ensemble du projet doit permettre aux étudiants d'acquérir des compétences transversales et de s'initier aux domaines de la spécialité SRI (voir tableau 4).

Tableau 4 : Vision globale du projet et des compétences mises en œuvre

PROJET	Gestion de Projet	Développement	Analyse image, texte, son	Interaction & Interface	Robotique & Capteurs	Réutilisabilité Code ou Fonctionnalités	Communication
PARTIE I	X	X	X			X	X
PARTIE II	X	X		X		X	X
PARTIE III	X	X	X	X	X	X	X

Un travail de restitution des compétences acquises individuellement vous sera demandé en fin d'année. Cela s'appuiera sur une formation qui vous sera dispensé en janvier.

## II- FIL ROUGE PARTIE I - Indexation de documents

Ce projet demande **du travail personnel à réaliser en dehors des séances encadrées**, il est donc indispensable de bien le spécifier et de se répartir équitablement le travail. Le travail de spécifications doit s'appuyer sur le cours **Outils de modélisation Informatique (UML)**. Il sera complété par les interventions de **gestion de projet (TD)** pour vous aider à appréhender les parties *Spécification, Conception et Intégration de la première partie*. Le travail à réaliser se base également sur du **code développé dans le cadre des TP UNIX & C** et vise à réutiliser le code déjà produit.

### II-1. Contexte général

L'accès à l'information est un enjeu essentiel. Pouvoir exploiter le plus efficacement possible la grande quantité de documents mis à disposition (Internet, archives...) nécessite d'abord d'avoir une description synthétique du contenu de chacun d'eux. Cela correspond à une phase d'**indexation**. Celle-ci peut être faite manuellement lorsque l'ensemble des documents accessibles est limité et facilement gérable. Au-delà, il est indispensable de disposer de méthodes d'**indexation automatique**.

Les documents disponibles sont généralement de **nature diverse** (*document texte ou sonore, image...*) et enregistrés suivant le cas dans un **format standard** prédéfini (*XML, JPEG, WAVE...*). La **description synthétique d'un document** sera donc fonction de sa nature et de son contenu. Cette description sera utilisée ensuite par un **moteur de recherche** pour récupérer les documents répondant le plus possible à la requête d'un utilisateur.

L'**utilisateur** ne sait pas comment fonctionne le moteur de recherche ni comment les documents sont indexés. Les choix concernant les méthodes d'indexation, les paramètres associés et leur configuration relèvent des **administrateurs**.

### II-2. Types de documents traités

Un **document** correspond à un **fichier** de données. Il se caractérise par son **nom** et son **extension**. Plusieurs **types de documents** sont pris en compte dans ce projet. Pour simplifier la lecture de certains fichiers, ceux-ci peuvent être accompagnés d'une version textuelle (extension .txt), voir tableau 5.

Un **ensemble de documents** (appelé également *corpus*) sera mis à votre disposition. Ces documents seront organisés en **répertoires**, chacun étant dédié à un type de document.

Tableau 5 : Type de fichiers à traiter

Type de document	Extension	Version .txt disponible
Texte	.xml	Non
Image couleur	.jpg	Oui (matrices intensités des pixels - RGB)
Image noir & blanc	.bmp	Oui (matrice intensités des pixels niveau de gris)
Audio	.wav .bin	Oui (valeurs des échantillons du signal numérisé)

### II.3- Processus d'Indexation automatique

Pour **rechercher** et **comparer** des documents il est nécessaire d'**indexer** leur contenu pour produire des *représentations simplifiées*, appelée **descripteurs**. La nature d'un descripteur dépend du type du document indexé ainsi que d'un ensemble de **paramètres**. Les valeurs des paramètres propres à chaque processus d'indexation sont données dans un **fichier de configuration** (fichier .config).

Une fois qu'un document est indexé, son descripteur est sauvegardé dans la **base de descripteurs** correspondant au type de document traité.

La phase d'**indexation automatique de documents** consiste à produire un **descripteur** pour chacun des fichiers disponibles. Un fichier n'est indexé qu'une seule fois. L'ensemble des descripteurs est mis à jour en fonction de l'ajout et la suppression de documents dans les différentes bases. Ainsi, la suppression d'un fichier entraîne la suppression de son descripteur. L'ajout d'un nouveau fichier entraîne l'indexation de celui-ci, son descripteur étant ajouté aux descripteurs existants.

L'objectif de cette partie est donc de **spécifier, concevoir et développer des fonctionnalités** permettant d'exploiter les documents mis à disposition, et ce, le plus efficacement possible. Des **fonctionnalités d'indexation et de comparaison** devront être développées pour chaque type de documents traités. Des indications sur les méthodes d'indexation élémentaires et de comparaison de descripteurs sont données en annexe.

### II.4- Comparaison de documents

**Comparer** deux documents de même type entre eux revient à comparer leurs descripteurs. Deux documents sont considérés comme **identiques** si leurs descripteurs sont strictement les mêmes.

La notion de **similarité** est plus large. Deux documents de même type sont considérés comme similaires si leurs descripteurs ont plus de X% de caractéristiques en commun. Le calcul de similarité repose généralement sur le calcul d'une « distance » entre descripteurs. Si la distance est faible les documents sont considérés comme similaires (si cette distance est nulle alors les documents seront considérés comme identiques). Plus cette distance est importante, plus les documents sont considérés comme différents. Le **critère de décision** utilisé correspondra donc à une valeur de **seuil** ou à une **proportion** au-delà de laquelle deux documents seront considérés comme proches.

Comparer deux documents c'est donc aussi considérer le **taux de similarité de leurs descripteurs et le comparer à un seuil**.

### II.5- Moteur de recherche basique

Pour valider les fonctionnalités d'indexation et de comparaison, il est nécessaire de développer un embryon de **moteur de recherche, éventuellement en mode administrateur / mode utilisateur**.

**En mode administrateur** (mode debug) : il s'agit de permettre de **configurer** chaque processus d'indexation, de **lancer l'indexation** et de **visualiser** les valeurs des descripteurs et de faire différents types de tests.

**En mode utilisateur** : il s'agit de recueillir une **requête utilisateur** et d'explorer les bases de documents, via les descripteurs (obtenus en phase d'indexation) et leurs **comparaisons**. L'objectif est de ramener une liste de documents répondant à la requête formulée, de les ordonner en fonction du taux de similarité (similarité décroissante) si celui-ci peut être calculé, et d'afficher le résultat de la requête.

Cette **interface utilisateur sommaire** (mode alphanumérique, saisie et affichage de caractères) permettra de saisir les éléments nécessaires, d'afficher les résultats et les messages d'erreurs éventuels.

**ATTENTION :** La réalisation d'une interface graphique fera l'objet de la partie II du projet (voir tableau 3). Pour que l'intégration de votre noyau indexation/comparaison développé ici en C soit utilisable depuis une interface java, grâce à JNI, il faudra être rigoureux sur les « entrées-sorties » de vos modules et gérer l'affichage des résultats et des messages d'erreurs au niveau du moteur MAIS pas des autres fonctionnalités.

## II.6- Organisation des équipes

Une équipe de 5 **personnes** sera constituée pour réaliser cette partie. Le travail de **spécification et conception** sera fait en commun, pour que l'ensemble des membres du groupe parte sur les mêmes bases (réflexion sur les structures de données, sur le processus d'indexation, le format des fichiers de sauvegarde, les « entrées-sorties » de différents modules, ...).

Le **développement et les tests** seront ensuite répartis entre les différents membres du groupe. Les **fonctionnalités développées devront être testées (unitairement) et intégrées** pour valider l'ensemble de l'application. L'objectif est de constituer une seule et même application intégrant les traitements réalisés sur les différents types de documents. Le suivi des activités se fera en utilisant l'outil collaboratif Trello (<https://trello.com/>). Le « client » devra avoir accès à ces informations.

Des détails supplémentaires sur cette partie sont donnés en annexe de ce document ainsi que les principes relatifs aux méthodes d'indexation à développer.

## III- Moteur de recherche avancé (partie II)

Un moteur de recherche avancé sera développé en java. Celui-ci intégrera une **interface graphique** développée suivant l'approche MVC (Modèle-Vue-Contrôleur). Cette interface permettra de saisir des éléments nécessaires pour effectuer une recherche de documents dans la base préalablement indexée (voir partie I). Elle proposera différents types d'aide à la saisie de la requête (menus déroulant, liste prédéfinies, ...).

**Plusieurs moteurs de recherche** pourront même être accessibles via l'interface. Dans ce contexte, une **première possibilité** sera donc de sélectionner **un moteur** et de lancer la recherche via ce moteur. Le résultat obtenu sera alors affiché via l'interface. Une **seconde possibilité** sera de lancer la même recherche sur **plusieurs moteurs**. Le résultat final dépendra alors de la fusion des résultats obtenus par chacun d'eux (système de vote par exemple). Cette approche s'appuiera par exemple sur des méthodes utilisées pour le développement de logiciels sécuritaires (vues en S6).

Un **premier prototype basse fidélité** sera réalisé pour valider les choix des modes d'interaction, ainsi que la structure de l'interface (choix fonctionnels, modes de saisie de l'information nécessaire à la recherche, choix pour l'affichage des résultats et l'accès aux documents sélectionnés, affichage des messages d'erreur, ...). Plusieurs types de recherche pourront être envisagés et seront décrits ultérieurement. Les éléments concernant l'IHM seront abordés en début de S6.

Le moteur de recherche avancé sera implémenté en java (processing, JNI). Le lien sera fait avec le moteur que vous aurez réalisé dans la première partie du projet (interface avec le code C) et avec les autres moteurs sélectionnés.

Un cahier des charges plus détaillé vous sera donné en janvier 2021.

## IV- Vers d'autres applications (partie III)

D'autres applications pourront vous être proposées qui nécessiteront peut-être le recours à des traitements développés dans les phases précédentes (traitement d'images, traitement de l'audio...). Il s'agira donc de savoir exploiter ce qui a déjà été développé pour le réutiliser ou l'adapter.

Le cahier des charges de cette 3<sup>ème</sup> partie sera donné dans le courant du second semestre 2020-2021.

## V- Validation et Évaluation de la Partie I

Ceci se fera en plusieurs étapes suivant le planning donné ci-après...

**ETAPE 1 :** Constituer un groupe de 5 personnes. Analyser le problème posé, le décortiquer pour bien le comprendre ou identifier les points que vous ne comprenez pas. Confrontez vos points de vue. S'il reste des éléments à clarifier prenez contact avec le client. Chaque groupe aura un référent.

**ETAPE 2 :** Rédiger un dossier de spécification fonctionnelle qui sera rendu selon le planning prévu (voir tableau 5). Ce dossier décrira le **projet en terme de fonctionnalités** (= *Que doit faire le système demandé par le client ?*). **Il ne doit pas décrire comment le problème sera résolu** (=pas de code). Les détails sur les objectifs d'un tel document vous seront apportés en TD de Gestion de Projet (avec Julien Vanderstraeten).

L'équipe encadrant le projet devra valider ces spécifications et donner le **feu vert** pour le passage en phase de conception/développement/tests/intégration. Vous vous appuyerez sur les séances de gestion de projet planifiées en support du projet.

**ETAPE 3 :** Les développements se feront en C uniquement sous Linux/Unix (utiliser une machine virtuelle Linux si vous travaillez sous Windows). Il est recommandé d'utiliser des commandes ou scripts Unix pour simplifier certains traitements (récupération de listes de fichiers, tri alphabétique, faire la différence entre deux listes, ...).

Des compléments concernant le langage C, les commandes ou l'écriture de scripts UNIX sont disponibles sous Moodle, sur l'espace réservé au projet.

**CONSIGNES :** tout programme doit être écrit de façon à **prévoir tous les cas limites**. Il est **vivement** conseillé d'écrire l'algorithme correspondant avant de coder et de s'entendre sur le type de données échangées entre les modules ou fonctions, soit leurs « entrées-sorties ». Le programme doit ensuite être **testé de façon à s'assurer de son bon comportement dans tous les cas d'utilisation** (bonne ou mauvaise).

*Remarque :* afin de faciliter le débogage, il sera, sans doute, intéressant d'avoir des fonctions permettant de visualiser les descripteurs.

### ATTENTION :

- Il est important de veiller à la **portabilité du code**, le système doit être fonctionnel **dans un environnement Linux sur une seule et même machine**.
- Si vous développez sur vos ordinateurs personnels il faut que le système fonctionne sur TOUTES les machines de votre groupe.
- Ne pas se lancer dans des choses trop compliquées qui ne fonctionnent que d'un côté et pas de l'autre. De plus, comme vous aurez à croiser les systèmes (utiliser ceux des autres et faire utiliser le vôtre), veillez à la **qualité et la pertinence des commentaires**.

## ETAPE 4 : EVALUATION

Le **jour de la validation**, chaque groupe fera la démonstration du fonctionnement de son système d'indexation et de son moteur de recherche.

Un **rapport décrivant l'architecture logicielle** (organisation en modules et sous modules), exemples de descripteurs construits, bilan technique (ce qui marche ou pas) et individuel (quelle a été la contribution et le retour d'expérience de chacun) sera remis dans les jours suivant la validation. Vous rendrez également l'intégralité du **code commenté et testé et validé**.

En plus de la validation machine, la prise en compte des consignes, le respect des délais, la qualité de la rédaction et de la programmation et l'investissement de chacun contribueront également à la note finale.

Ce projet est l'occasion d'élargir vos compétences UNIX et en programmation C. Certains aspects non abordés en TP devront être intégrés dans le projet et seront évalués.

Compétences	Fonctions C et Commandes Unix	Objectifs
Fichiers	fopen, fscanf, fprintf, fclose, feof, ...	Savoir ouvrir et fermer un fichier ; lire et écrire des données en utilisant les formats %s, %d, %f, ... (fsacnf/fprintf)
Pointeurs	Test à faire après un malloc, un calloc, un fopen, ...	Eviter d'accéder à des données à partir d'un pointeur NULL ⇒ Limiter les erreurs segmentation Fault - Core Dumped
Chaînes de caractères	strcpy, strcmp, strcat, strchr, ... autres fonctions de <string.h>	Manipulation des chaînes de caractères lire, écrire et construire une chaîne (nettoyage, tri, concaténation ...) Pas de lecture caractères par caractères
Commandes unix	system, ls, sort, unique, diff, ...	Appeler une commande Unix depuis un programme C et exploiter son résultat
Script Unix	System, redirection dans un fichier, ...	Simplifier les traitements en utilisant les commandes Unix ou combinaisons de commandes existantes
Gestion des Piles de descripteurs	Modules Pile.c Pile.h Element.c Element.h, ...	Réutiliser les modules développés en TP pour gérer les descripteurs (créer, stocker, sauvegarder).
...	...	...

## VI- Planning Partie I

Des séances de TD de gestion de projets sont planifiées pour vous aider à préparer les différentes étapes de réalisation de cette partie du projet. Les dates de livraison des différents documents à rendre sont déjà fixées (voir tableau 6).

**CONSIGNES PRATIQUES A SUIVRE** : Les dépôts de dossiers/archive code se feront sous Moodle. Eviter l'envoi par mail pour ne pas saturer nos boîtes mail. Les noms de groupes doivent figurer dans les noms de fichiers.

Les consignes par rapport aux évaluations des deux autres parties du projet seront données ultérieurement.

**Etat d'avancement avant la phase d'intégration** : Chaque membre du groupe doit remplir le tableau Trello régulièrement pour montrer l'avancement et la répartition du travail. L'équipe encadrante consultera régulièrement ce tableau.

Tableau 6 : Organisation détaillée de la partie I – Livrables et dates clés

Étapes clés	Dates
Présentation du projet et constitution des groupes	19/10/2020 - 14h
TD Gestion de Projet	23/10/2020 – 13h45
Constitution des groupes	21/10/2020 (envoi de la liste par les délégués)
TD Gestion de Projet	6/11/2020- 13h45
Spécifications	Dossier à rendre pour le 13/11/2020
Conception et développement	Du FEU VERT au 8/01/2021
Intégration au plus tôt (prévoir 1 semaine)	Intégration à finaliser <b>avant le 15/01/2021</b>
TD Gestion de Projet	30/11/2020 – 7h45 ; 16/12/2020 – 13h45 ; 5/11/2021 – 7h45
Consultation des enseignants possible	10/12/2021 ; 17/12/2020 ; 7/01/2021 De 7h45 et 10h suivant les groupes.
Validation finale sur machine (45 mn par groupe)	22/01/2021 – 13h (selon ordre de passage) + livraison du code
TD Gestion de projet : débriefing	25/01/2021 - 10h
Clôture de la partie 1 du projet Fil Rouge	Remise du rapport <b>le 29/01/2021</b>

## ANNEXE – PARTIE I

Une **base de documents** sera mise à disposition. Cette base contiendra des documents de type **Texte**, de type **Image** et de type **Audio**.

### 1. Indexation de contenu

Chaque type de document disposera d'une **fonctionnalité d'indexation** et d'une **fonctionnalité de comparaison**. La première construira le descripteur associé au document à indexer et la seconde aura pour objectif de comparer 2 descripteurs de même type pour déterminer s'il s'agit de documents similaires ou non.

### 2. Recherche de contenu

La **fonctionnalité de recherche** consistera à partir d'un **document initial** (et de son descripteur) ou bien d'un **critère de recherche** et à rechercher dans la base de documents ceux qui sont similaires au document initial ou bien qui répondent le plus au critère de recherche donné par l'utilisateur. La recherche pourra donc se faire de différente manière :

- **suivant un critère de recherche donné par l'utilisateur** (ex : recherche d'un mot clé dans des documents de type **Texte**, recherche d'une couleur dominante dans un document de type **Image**, recherche d'un mot prononcé ou d'un son dans un document **Audio**...)
- **suivant la similarité** de leur contenu : savoir dire si deux documents sont similaires ou non sur la base de calcul de « distance » entre descripteurs et de seuils. Ainsi, on pourra rechercher les documents présents dans la base qui sont les plus proches (les plus similaires) du document donné en exemple (éventuellement non présent dans la base, dans ce cas il devra être indexé automatiquement au préalable).

### 3. Contraintes techniques / Exigences client

Les fonctionnalités nécessaires à la phase d'indexation et de recherche seront développées **en C sous UNIX/Linux** et utiliseront les concepts de programmation associés, dont la plupart seront vus en TP d'informatique au S5. L'objectif du projet est non seulement de **réutiliser certains développements réalisés en TP** (*pile statique ou dynamique, histogramme de couleur,...*) mais aussi de **compléter ces apprentissages** en explorant d'autres aspects non abordés (*manipulation de fichiers, lecture/écriture formatée, commande système, écriture de scripts unix, ...*). Des documents seront mis à disposition sur moodle pour vous aider sur ces aspects non vus en TP.

Toutes les **fonctions** développées devront être conçus pour être **génériques**, et **réutilisables**. Elles devront être **testées unitairement** avant d'être **intégrées**. Le **travail préliminaire de spécification et de conception** est donc primordial. Les **descripteurs** extraits de chaque document sont au cœur de l'application. Le choix des **structures de données** correspondant aux descripteurs est donc essentiel.



**Exigences client** : cette première partie valide des compétences en Algorithmique et en système UNIX/Linux et langage C. En l'occurrence il est indispensable de respecter ces contraintes (pas de Windows, ni de C++...).

#### 4. Organisation en Sous-Projets

La définition, l'extraction et la comparaison de descripteurs fera l'objet du sous-projet 1 (SP1) « Indexation automatique », lui-même divisé en autant de modules que de type de documents à traiter (**Texte, Image, Audio...**).

La recherche d'information et l'exploitation du contenu des bases disponibles feront l'objet du sous-projet 2 (SP2) « Moteur de recherche basique ». De même, différents modules permettront d'effectuer la recherche sur différents types de documents.

##### 4.1. SP1 : Indexation automatique

Chaque type de document a ses spécificités et devra disposer d'une fonction d'indexation et de comparaison propre. Il vous appartiendra de bien identifier ce qui relève d'un traitement commun (valable dans tous les cas) et d'un traitement spécifique à la nature du document. Le travail de spécification réalisé en UML vous aidera à décrire correctement le problème.

Principe général : le processus d'indexation consiste à traiter successivement tous les fichiers disponibles, non déjà indexés. Lorsqu'un document est traité, son descripteur est créé et empilé dans une structure de type Pile (cf TP sur le sujet). Lorsque tous les documents ont été traités, le contenu de la pile est sauvegardé dans un premier fichier. La liste des fichiers traités et le lien avec le numéro du descripteur créé est stockée dans un second fichier.

Afin de cadrer le projet, des indications sur les méthodes d'indexation à implémenter sont données ci-après.

##### A- Document de type Texte

**Méthode** : une méthode de base consiste à repérer la présence de mots clés ou bien à identifier les mots les plus fréquents du texte (taux d'apparition ou fréquence supérieure à un seuil donné) pour les proposer comme index potentiels.

**Structure d'un document de type texte** : les documents à traiter sont au format XML et seront fournis en temps utile sous Moodle.

Il faut distinguer les **balises**, qui sont des marqueurs de la structure du texte, du **texte** proprement dit constitués de **mots**. Une **balise** est une suite de caractères entre '<' et '>' :

**<balise ouvrante>** *texte* **<balise fermante>**

Les noms des balises sont prédéfinis et figurent en gras dans l'exemple suivant.

##### Exemple de contenu de type texte

```
<?xml version="1.0" encoding="iso-8859-1"?>
<article>
  <date>Mon, 12 Sep 2005 11:52:20 GMT</date>
  <auteur>Adeline Cordon</auteur>
  <titre>Agassi plie face au meilleur joueur qu'il ait affronté</titre>
  <resume>Lisez l'intégralité de l'article pour plus d'information.</resume>
  <texte>
    <phrase>Roger Federer saute de joie.</phrase>
    ...
    <phrase>Une victoire sur la terre battue lui permettrait de rejoindre
    Andre Agassi, seul et unique joueur à avoir gagné les quatre tournois
    du Grand Chelem sur des surfaces différentes depuis le début de l'ère
    open, en 1969.</phrase>
  </texte>
</article>
```

**A NOTER** : il n'est pas demandé d'utiliser des bibliothèques de manipulations de fichiers XML. L'enjeu ici est de savoir les traiter comme des fichiers texte et de savoir lire dedans ce qui est utile pour le traitement. Lire et traiter à la volée pour nettoyer le texte et construire les descripteurs.

**A EVALUER** : Lecture/écriture formatées (**fscanf / fprintf**) et manipulation de chaînes de caractères (fonctions dédiées dans la bibliothèque C **<string.h>**).

##### Unité de base : le token

Un **token** est une chaîne de caractères (tableau de caractères terminé par un '\0' en langage C). Un texte est une suite de tokens séparés par des espaces. On considère qu'un **token** a une taille maximale **TAILLE\_TOKEN\_MAX**. Seuls les tokens correspondant à des mots pertinents seront retenus.

Avant d'analyser un texte pour construire son descripteur il est donc nécessaire de faire une première passe de prétraitement pour nettoyer le texte.

**Phase de nettoyage** : lors de cette phase de prétraitement, un texte au format .xml sera transformé en fichier .clean. Il sera donc nécessaire d'enlever les balises, de séparer les mots pas un espace. Ainsi, la chaîne de caractère l' arbre deviendra l' arbre, soit 2 mots (l' et arbre) séparés par un espace. La ponctuation (',', ',', '!', '?', ':', ...) sera enlevée et les majuscules remplacées par la lettre minuscule correspondante. Suivant le cas, vous pourrez conserver les contenus des titres et/ou des résumés.

##### Exemple de contenu après nettoyage (fichier .clean)

```
agassi plie face au meilleur joueur qu'il ait affronté
lisez l'intégralité de l'article pour plus d'information
roger federer saute de joie ... une victoire sur la terre battue lui permettrait de
rejoindre andre agassi seul et unique joueur à avoir gagné les quatre tournois du
grand chelem sur des surfaces différentes depuis le début de l'ère open en 1969
```

Une seconde phase de préparation est ensuite nécessaire pour analyser la fréquence des **tokens** d'un texte et construire le descripteur associé.

**Phase de filtrage :** les **tokens** correspondant à des mots outils sont supprimés. Ils correspondent à des mots relativement fréquents, mais qui n'apportent pas d'information sur le contenu du texte (*il, le, un, quand, ...* n'informent pas sur le thème abordé). Ces mots (*stopwords*) sont listés dans des ressources appelées *stoplist*, nombre d'entre elles sont disponibles sur internet<sup>1</sup>. Cette phase consiste à éliminer ces tokens, pour ne conserver que ceux dont la fréquence peut avoir du sens. A chaque texte au format `.clean` sera associé un fichier `.tok` après filtrage.

**Exemple de contenu après filtrage (fichier .tok)**

```
agassi plie face meilleur joueur affronté
lisez intégralité article information
roger federer saute joie ... victoire terre battue permettrait rejoindre andre agassi
seul unique joueur gagné quatre tournois grand chelem surfaces différentes depuis
début ère open en 1969
```

**Premier niveau de description : la notion de Terme**

Tout mot pertinent (token restant après nettoyage et filtrage) sera associé à son nombre d'occurrences dans le texte et formera un **terme**. La valeur {"olympiques", 15} signifie que le token « olympiques » est présent 15 fois dans le texte traité.

**Descripteur d'un document de type Texte :** le but est de représenter de manière synthétique le contenu de chaque texte. Après traitements (nettoyage et filtrage), chaque texte sera représenté par un descripteur contenant au moins :

- l'identifiant unique du document traité,
- la liste des **termes** (token, nombre\_occurrences) les plus significatifs,
- le nombre total de termes retenus,
- ainsi que la taille du texte en nombre de tokens (exemple : 1000 tokens dans le fichier `.tok` traité).

Descripteur associé à un texte après nettoyage, filtrage et comptage du nombre d'occurrences.

1	
« jeux »	20
« olympiques »	15
« natation »	10
« médaille »	7
4	
1000	

D'autres informations pourront y être ajoutées si nécessaire. La taille de la liste de termes sera *a priori* variable, fonction soit (A) d'une **valeur limite** (ex : X mots les plus présents) ou (B) d'un **seuil** (ex : les mots qui apparaissent plus de X fois) ou bien d'une combinaison des 2. Cependant la modalité devra être spécifiée au démarrage de l'application sous forme par exemple de **paramètres de configuration**.

<sup>1</sup> <https://github.com/stopwords-iso/stopwords-fr/blob/master/stopwords-fr.txt>

**Sauvegarde des descripteurs :** pour éviter de traiter plusieurs fois les mêmes documents, chaque descripteur calculé sera sauvegardé dans le fichier nommé **base\_descripteur\_texte**. Celui-ci sera chargé en début de traitement s'il existe déjà ou bien il sera créé si aucun document n'a encore été indexé. La nouvelle version sera sauvegardée en fin de traitement (ajout de nouveaux descripteurs par exemple).

**Lien descripteur/document traité :** la liste des documents traités (dont le descripteur se trouve dans le fichier **base\_descripteur\_texte**) sera mémorisée dans un seul et même fichier **liste\_base\_texte**. Il fera le lien entre le nom ou chemin du fichier correspondant au document traité et l'identifiant unique qui lui est associé dans le descripteur.

**Table d'index des mots-clés :** pour optimiser la recherche de documents en fonction d'un ou plusieurs mots-clés, une **table d'index** sera construite et enrichie en parallèle. Chaque cellule contiendra un mot clé (mot présent au moins dans un des descripteurs existants), une liste comportant l'identifiant de chaque fichier contenant ce mot-clé ainsi que le nombre d'occurrences trouvé ou bien la fréquence associée (Nombre d'occurrence / Nombre total de token) ce qui peut faciliter les comparaisons ensuite.

Ainsi dans l'exemple suivant, on saura que le mot « olympiques » est présent 15 fois dans le fichier numéro 1, 4 fois dans le fichier numéro 3, etc.

« olympiques »	
1	15
3	4
25	6
...	...

Connaissant le mot-clé, on accèdera facilement à la liste des documents qui le contiennent sans avoir à parcourir tous les descripteurs existants. Cette table pourra être sous forme de « table de hachage » (indice début-fin du sous ensemble de mots-clés à balayer → si le mot clé commence par « a », inutile de chercher dans les mots commençant par « b », « c », ...) ou bien d'arbre binaire (sous arbre gauche → mot avant dans l'ordre alphabétique, sous arbre droit → mots après dans l'ordre alphabétique).

A l'issue du processus d'indexation de documents de type **Texte**, cette table sera sauvegardée dans le fichier **table\_index\_texte**.

Lors de la première session d'indexation, ce fichier sera créé. Lors des sessions suivantes, il sera chargé en début de session d'indexation et sa nouvelle version sauvegardée en fin de session.

**Pour résumer :** à l'issue d'une phase d'indexation de documents de type **Texte**, les 3 fichiers **base\_descripteur\_texte**, **liste\_base\_texte** et **table\_index\_texte** sont créés ou bien mis à jour.

## B- Document de type Image

**Méthode** : une méthode de base consiste à calculer l'histogramme des couleurs en comptabilisant le nombre d'occurrences d'une même valeur d'intensité, suivant ce qui a pu être déjà fait en TP (cf. TP Image).

### Unité de base : le pixel


Une image est une matrice. Chaque élément correspond à un pixel et possède une intensité (valeur entre 0 et 255). On dispose de deux types d'images :

- images couleurs ayant 3 composantes : **red**, **green**, **blue** (RGB)
- images en niveaux de gris (une seule composante).

### Prétraitement : la quantification

Au préalable, et afin de limiter les calculs, une quantification sera nécessaire afin de ne retenir que les **n** bits de poids forts (**n** premières puissances de 2) pour chaque valeur d'intensité dans ces composantes couleurs.

Exemple de traitement :

Soit le pixel suivant : , ayant les valeurs (255, 128, 64), c'est-à-dire une intensité de 255 sur la composante rouge (R), de 128 sur la composante verte (G) et de 64 sur la composante bleue (B).

La quantification sur les 2 premiers bits donne :

Pour 255 =  
 $1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$   
 → 1<sup>er</sup> bit R1: 1, 2<sup>ème</sup> bit R2: 1

Pour 128 =  
 $1 \times 128 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 \Rightarrow$  1<sup>er</sup> bit  
 → 1<sup>er</sup> bit G1: 1, 2<sup>ème</sup> bit G2: 0

Pour 64 =  $0 + 1 \times 64 + 0 + 0 + 0 + 0 + 0 + 0 + 0$   
 → 1<sup>er</sup> bit B1: 0, 2<sup>ème</sup> bit B2: 1

On se retrouve avec 6 bits (que l'on organise par exemple ainsi) :

**R1 R2 G1 G2 B1 B2**

Avec R1 1<sup>er</sup> bit de la composante rouge, R2 2<sup>ème</sup> bit de la composante rouge, G1 1<sup>er</sup> bit de la composante verte, G2 2<sup>ème</sup> bit de la composante verte, B1 1<sup>er</sup> bit de la composante bleue, B2 2<sup>ème</sup> bit de la composante bleue.

Sur notre exemple, nous avons : **1 1 1 0 0 1**, soit le nombre : **57**.

$57 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

Donc avec une quantification sur 2 bits, nous obtenons des pixels ayant une valeur entre 0 (les 6 bits à 0) et 63 (les 6 bits à 1).

### Descripteur d'un document de type Image : histogramme

Pour chaque pixel de l'image, nous réalisons le prétraitement (quantification) traitement puis nous calculons l'histogramme correspondant aux 64 valeurs

possibles. L'histogramme comptabilise le nombre de pixels de l'image ayant une valeur d'intensité donnée.

### Structure d'un document de type Image :

Les images à traiter sont présentes au format **JPEG** (.jpg) afin de les visualiser **et** au format texte (.txt, dit « brut ») afin d'effectuer les traitements. Ils seront fournis en temps utile sous moodle.

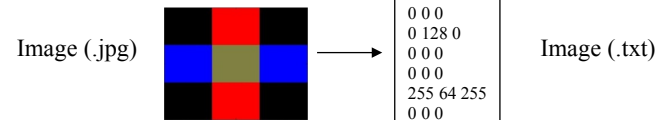
Chaque fichier texte est composé de la façon suivante :

- 1<sup>ère</sup> ligne : taille de l'image,
- matrice représentant les valeurs des pixels dans la composante « rouge »,
- matrice représentant les valeurs des pixels dans la composante « verte »,
- matrice représentant les valeurs des pixels dans la composante « bleue ».

Remarques :

1. La taille de l'image est composée de 3 valeurs (dimensions) : la longueur (**l**), la hauteur (**h**), le nombre de composantes (**d**), par exemple « 3 » pour une image couleur RGB, en « rouge, vert, bleu ».
2. Chaque matrice possède « h » lignes et « l » colonnes.
3. Les valeurs de l'image sont comprises entre 0 et 255.

Exemple de document :



**Descripteur d'un document de type Image** : le but est de représenter de manière synthétique le contenu de chaque image. Après les étapes de quantification et de calcul de l'histogramme, chaque image sera représentée par un descripteur contenant au moins :

- l'identifiant unique du document traité,
- un vecteur de taille maximale  $2^{4n}$  représentant l'histogramme, avec **n** le nombre de bits utilisés pour la quantification et **d** le nombre de composantes de l'image.

Pour une image couleur (3 composantes, RGB) quantifiée sur 2 bits, nous aurons alors :

	Id1
Valeur 0	12
1	5
...	...
21	8
...	...
63	20



Ceci signifie qu'après quantification, 12 pixels de l'image ont une valeur 0, 5 ont une valeur de 1, 8 ont une valeur de 21, etc.

**Sauvegarde des descripteurs** : pour éviter de traiter plusieurs fois les mêmes documents, chaque descripteur calculé sera sauvegardé dans le fichier **base\_descripteur\_image**. Celui-ci sera chargé en début de traitement et sa nouvelle version sauvegardée en fin de traitement.

**Lien descripteur/document traité** : la liste des documents traités (dont le descripteur se trouve dans le fichier **base\_descripteur\_image**) sera mémorisée dans le fichier **liste\_base\_image** et fera le lien entre le nom du fichier correspondant au document traité et l'identifiant unique utilisé ensuite comme référence dans le descripteur.

**Pour résumer** : à l'issue d'une phase d'indexation de documents de type **Image**, les 2 fichiers **base\_descripteur\_image**, **liste\_base\_image** sont créés ou bien mis à jour.

### C- Document de type Audio

**Méthode** : une méthode naïve consiste à découper le signal en plusieurs fenêtres de taille **n** échantillons (par exemple  $n=1024, 2048, 4096$  ou  $8192$  points), puis de calculer pour chacune des fenêtres un histogramme composé de **m** intervalles (par exemple  $m=30, 50, 100$  barres entre -1 et 1).

#### Structure d'un document de type Audio :

Les fichiers sonores seront fournis en temps utile sous moodle selon plusieurs formats :

- **WAVE** (.wav) afin de les écouter,
- **Binaire** (.bin) afin d'effectuer les traitements.
- **Texte** (.txt) afin de vérifier votre lecture du fichier binaire.

Chaque fichier binaire est composé de la suite des échantillons (points) du signal au format **double**.

Si vous souhaitez travailler directement sur le fichier **WAVE**, il vous faudra enlever l'entête du fichier, soit 56 octets...

#### Unité de base : la fenêtre

Le nombre **n** d'échantillons correspondant à une **fenêtre d'analyse** sera un paramètre de configuration.

**Premier niveau de description** : **histogramme sur une fenêtre d'analyse**  
L'**histogramme** comptabilise le nombre d'échantillons de la fenêtre d'analyse dont la valeur est comprise dans un intervalle donné. Le nombre **m** d'intervalles est un paramètre de configuration.

**Descripteur d'un document de type Audio** : le but est de représenter de manière synthétique le contenu de chaque fichier son. Après les étapes de lecture, découpage en fenêtre et de calcul d'histogrammes, chaque son sera représenté par un descripteur contenant au moins :

- l'identifiant unique du document traité,
- une matrice de taille maximale **k\*m** représentant l'histogramme, avec **k** le nombre de fenêtres d'analyse (obtenues lors du découpage) et **m** le nombre d'intervalles (utilisé lors du calcul de chaque histogramme).

Nous obtenons alors :

		Id1		
Fenêtre	0	12	...	440
	...			
k	...	...		
		1	...	m
		Histogramme		

**Sauvegarde des descripteurs** : pour éviter de traiter plusieurs fois les mêmes documents, chaque descripteur calculé sera sauvegardé dans le fichier **base\_descripteur\_audio**. Celui-ci sera chargé en début de traitement et sa nouvelle version sauvegardée en fin de traitement.

**Lien descripteur/document traité** : la liste des documents traités (dont le descripteur se trouve dans le fichier **base\_descripteur\_audio**) sera mémorisée dans le fichier **liste\_base\_audio** et fera le lien entre le nom du fichier correspondant au document traité et l'identifiant unique utilisé ensuite comme référence dans le descripteur.

**Pour résumer** : à l'issue d'une phase d'indexation de documents de type **Audio**, les 2 fichiers **base\_descripteur\_audio**, **liste\_base\_audio** sont créés ou bien mis à jour.

### 4.2. SP 2 : Moteur de recherche

La recherche dépend du critère spécifié par l'utilisateur. Elle est basée sur une exploitation des descripteurs générés lors de l'indexation. Si aucun descripteur n'est disponible, l'indexation automatique des données sera lancée au préalable.

#### A- Document de type Texte

Dans le cas d'une recherche de documents de type texte, l'utilisateur aura le choix entre :

- **réaliser une recherche par mot-clé**. L'utilisateur devra saisir le mot correspondant (ex : « sport », « tennis », « politique »...) et le système renverra la liste des fichiers (ou chemins) contenant ce mot-clé. Cette liste sera triée par ordre décroissant du nombre d'occurrences du mot-clé dans ces fichiers.

Exemple :

```

Requête mot-clé : "musique"

Résultats (fichier -> occurrences) :
28-Festival__Ososphère,_un_week-end.xml      -> 6
05-Musique_électronique__l_élan_collectif.xml -> 5
12-Musiques_du_monde__les.xml                -> 4
17-La_voix_envoûtante_de_Giriya.xml          -> 4
22-Danse__quarante_jeunes_de.xml             -> 3

```

- **réaliser la recherche des textes les plus proches d'un texte donné** sachant qu'un texte proche aura un maximum de mots clés en commun avec le texte choisi. Exemple :

```

Requête fichier : "28-Festival__Ososphère,_un_week-end.xml"

Résultats (fichier -> nombre de mots-clés communs) :
05-Musique_électronique__l_élan_collectif.xml -> 10
12-Musiques_du_monde__les.xml                -> 8

```

Remarque : pour ces deux types de recherche, le document en tête de liste sera ouvert automatiquement à l'aide d'un éditeur de textes.

## B- Document de type Image

Dans le cas d'une recherche de documents de type image, l'utilisateur réalisera **une recherche à partir d'une image couleur (composantes RGB) ou bien d'une image en niveaux de gris**. L'utilisateur devra saisir le nom (ou chemin) de l'image choisie ou le sélectionner dans une liste affichée et le système renverra la liste des chemins vers les fichiers qui sont les plus proches (par exemple après calcul de l'intersection des histogrammes représentant les images...).

Exemple :

```

Requête image : "45.jpg"
Résultats -> 13.jpg 47.jpg 44.jpg 46.jpg

```

Remarque : pour ce type de recherche, l'image la plus proche sera ouverte automatiquement à l'aide d'un logiciel de visualisation d'image (**xv**, **eog**...).

## C- Document de type Audio

Dans le cas d'une recherche de documents de type son, l'utilisateur réalisera **une recherche à partir d'un fichier son (contenu dans le répertoire « requête »)**. L'utilisateur devra saisir le nom (ou chemin) du jingle choisi ou le sélectionner dans une liste affichée et le système renverra la liste des chemins vers le (ou les) fichier(s) qui sont le(s) plus proche(s) : par exemple, après calcul des intersections des histogrammes représentant le corpus et ceux représentant le jingle.

Une plus-value consistera à localiser temporellement (en secondes) chacune des apparitions du jingle dans le corpus. Exemple :

```

Requête son (jingle) : "jingle_m6.wav"
Résultats (corpus) -> corpus_m6.wav : 45s 87s 135s ...

```

Remarque : pour ce type de recherche, le fichier sonore (corpus) le plus proche sera écouté automatiquement à l'aide d'un lecteur audio.