Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement

2018-2019

Scraping d'image smart pour portail guide media TV/vidéo

Tuteur académique
Mathieu Delalandre

Étudiant
Louis Babuchon (DI5)

1er avril 2019

Liste des intervenants

Nom	Email	Qualité
Louis Babuchon	louis.babuchon@eut.univ-tours.fr	Étudiant DI5
Mathieu Delalandre	mathieu.delalandre@univ-tours.fr	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Louis Babuchon susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Mathieu Delalandre susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Louis Babuchon, *Scraping d'image smart pour portail guide media TV/vidéo*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
    author={Babuchon, Louis},
    title={Scraping d'image smart pour portail guide media TV/vidéo},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2018-2019}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document polytech.cls V2.2.5 (2017/05/02)

Table des matières

Table des figures

Liste des tableaux

1 Introduction

1 Acteurs, enjeux et contexte

Les acteurs du projets sont :

- MOE : Louis Babuchon
- MOA : Mathieu Delalandre (Enseignant chercheur du département informatique à Polytech Tours)

2 Enjeux et contexte

Nous sommes plus que jamais consommateurs de médias, et depuis toujours c'est la télévision qui est en tête des chiffres de la consommation. Mais maintenant la consommation Internet aussi bien mobile que via les ordinateurs ou même les télé connectées n'a cessé de croître au cours des dernières années. Contrairement aux idées reçues la télévision n'est pas en déclin avec l'évolution d'Internet et des services liés mais en stagnation Figure 1 (Chapitre 1) [WWW0].

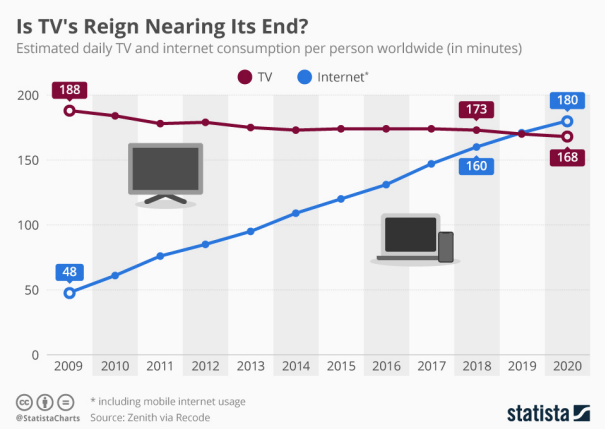


Figure 1 – *Étude montrant l'évolution du temps de consommation des médias par jour selon le support*

L'étude a été réalisée par l'agence Zenith sur 63 pays. On peut voir que internet est en pleine croissance et va dépasser la TV en 2020 avec 180 minutes de visionnage quotidien contre 168 pour la TV.

Cette étude permet de montrer que le temps de visionnage des médias croît. Ceci est expliqué par la multiplication des services multimédia que ce soit TV, services de replay, streaming ou encore vidéos à la demande. De part ce fai,t un certain nombres de guides ont été mit en place sur internet pour cataloguer les contenus. Le problème c'est que le nombre de guides différents est énorme, rien que pour la TNT française on retrouve plus de 20 guides différents rien que sur le deux premières pages de Google. Il y a plusieurs type de guides :

— Les guides qui sont fais par des journalistes, comme Telerama.
— Des guides opérateurs et fournisseur de contenu comme orange.

Dans ces sites on retrouve une description, quelques images ainsi que des vidéos pour présenter le programme, sur la Figure 2 (Chapitre 1) on peut voir sous quel format sont présentées les images décrivant un programme.

Figure 2 – *Présentation d'un guide TV*

Sur tous ces sites il y a quelques différences dans le contenu, mais en général il y a de la redondance de données. On retrouve les mêmes images sur de nombreux sites, car les sites se volent entre eux les images. Il y a deux types d'images pour les guides, soit des affiches de films soit des images tirées du film Figure 3 (Chapitre 1)

(a) *Affiche de la série Mr.Robot*

(b) *Images tirées de la série Mr.Robot*

Figure 3 – *Différentes images proposées par les guides TV*

Des études montrent que nous avons 90 secondes pour capturer l'attention d'un utilisateur,

passé ce délai il y a de grandes chances qu'il perde intérêt et change d'activité. Nous avons donc un temps très court pour capter l'intérêt, l'image a été montrée comme étant la solution la plus efficace pour faire découvrir un titre à un utilisateur car un cerveau humain peut analyser une image en 13 microsecondes.

En 2014 Netflix a fait une étude et a montré que l'image "d'artwork" constitue 82% de notre concentration lorsque nous parcourons la plateforme et en moyenne nous analysons pendant 1.8 secondes chaque programme avant de passer au suivant. C'est donc un enjeux énorme d'avoir la meilleur image possible pour décrire un programme et ainsi capter l'utilisateur.

Certains sites utilisent aussi la pré visualisation de vidéos avec des passages précis bien choisis pour permettre de présenter le programme d'une manière encore plus attirante[WWW0] [WWW0] [WWW0][WWW0].

3 Objectifs

Voici une étude faite sur les 17 premiers guide TV en France avec les images qu'ils proposent pour le programme "Charles Azvavour, l'intégrale" Figure 4 (Chapitre 1)

Figure 4 – *Etude réalisé sur le programme "Charles Azvavour, l'intégrale"*

Cette étude montre la redondance des images, en effet il y a seulement 3 images différentes pour près de 17 sites. Une même image peut avoir une moins bonne qualité que les autres, une teinte modifiée un redimensionnement, une compressions différente. De ce constat mes objectifs sont les suivants Figure 5 (Chapitre 1)

Le premier objectif est de former un base de données d'image très grande via un XMLTV pour pouvoir analyser les caractéristiques des images disponible sur les guide Tv.

Dans un second temps l'objectif et de pouvoir fournir un crawler pouvant récupérer les images des programmes sur plusieurs sites. Pour ensuite pouvoir appliquer des méthodes de détection des doublons, via les images restante extraire un certain nombre de metadonnées pour ensuite stocker toutes ces informations dans une base de données. De ce fait on aura une première sélection fiable et robuste.

Nous n'avons pas de contraintes de performances car les programmes étant visibles longtemps à l'avance pour la télé ainsi que les VOD, nous n'avons pas besoin d'un scrapping qui réagit à la microseconde.

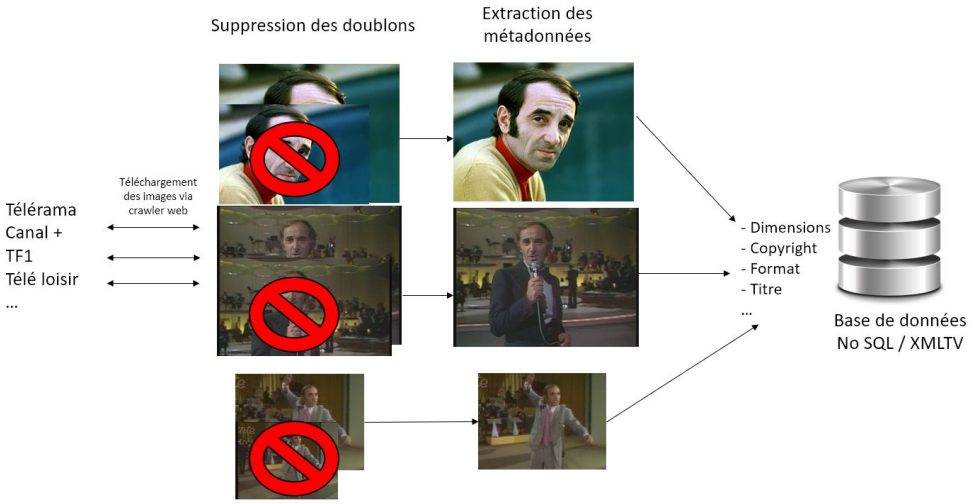


Figure 5 – *Organisation*

4 Hypothèses

Une base de donnée images sera peut être fournie, dans l'éventualité où elle ne pourra pas être fournie j'ai réalisé un programme qui récupère automatiquement les images des programmes du site Télérama [WWW0] pour à la fin du semestre avoir une base assez conséquente pour pouvoir faire la sélection des images. Le langage de programmation choisi sera la Java avec la librairie ImageJ, si cette librairie ne nous permet pas de répondre à tous nos besoins il faudra alors changer de librairie ou alors en ajouter une autre.

5 Bases méthodologiques

5.1 Les outils

— Pour la gestion de projet, je vais réaliser un diagramme de gantt pour contrôler le déroulement du projet via le site "Team gantt" ainsi que le site "Trello" pour gérer les tâches.
— Pour la modélisation logicielle je vais utiliser les diagrammes UML.
— Pour le rapport de projet, je vais utiliser LaTeX
— Pour la gestion de version je vais utiliser Github

5.2 La méthode de gestion de projet

Pour mener à bien ce projet je vais utiliser la méthode agile. Car le projet peut être découpé facilement en 3 lots qui composeront mes sprints.

2 Description générale

1 Environnement du projet

Le projet ne comporte pas d'éléments déjà existant. L'environnement de développement est le suivant :

- Le système d'exploitation : Windows.
- Le langage de programmation : Java 8.
- L'IDE : Eclipse.
- Le matériel pour le développement : Intel core i5 6300HQ, 8Go de Ram, carte graphique GTX 960m.
- La connection internet : Fibre avec un débit de 300Mb/s montant et descendant.

2 Caractéristiques des utilisateurs

Les utilisateurs du système seront moi et mon encadrant.

	Connaissance de l'informatique	**Expérience de l'application**	**Type de l'utilisateur**	**Droit d'accès utilisateurs**
Réponse	Oui	Oui	Occasionnel	Total
Commentaire	Enseignant chercheur	Encadre le projet	Permet de tester les méthodes mises en place	Toute les opérations sont possibles

3 Fonctionnalités du système

N'ayant qu'un type d'utilisateur voici le diagramme des cas d'utilisations de mon système associé à celui-ci. (Figure 1 (Chapitre 2))

Mon système est divisé en deux programmes indépendants, le crawler et le sélecteur. L'utilisateur ne possède que des actions simple, c'est à dire lancer le programme, ensuite le reste est automatique.

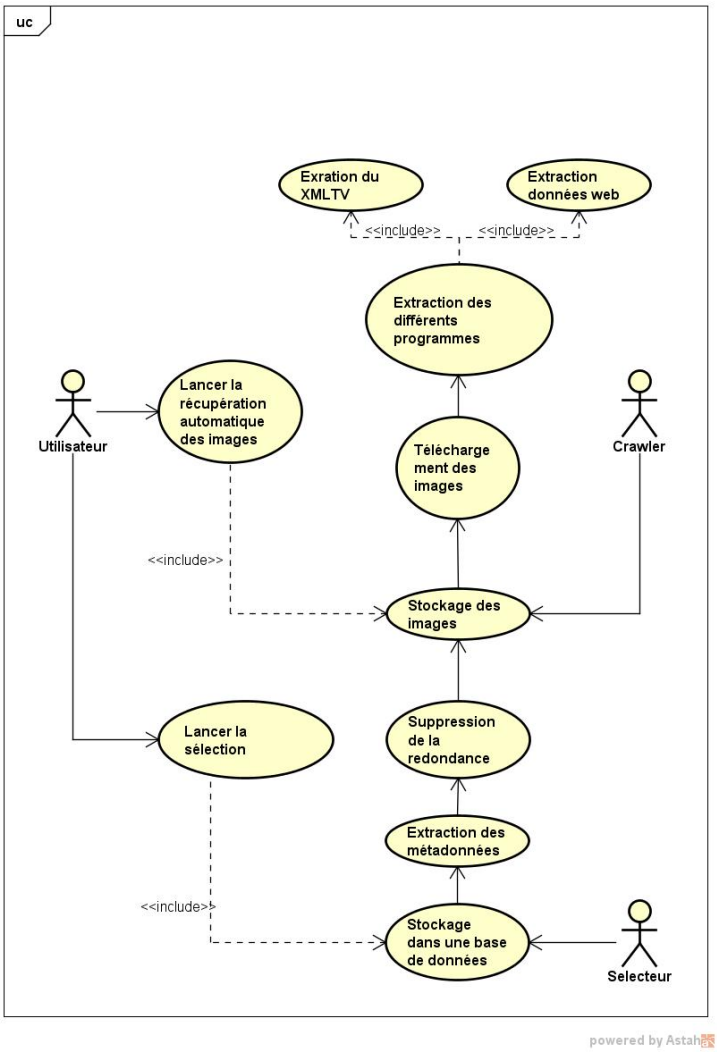


Figure 1 – *Diagramme des cas d'utilisation*

Le crawler permet d'extraire des programmes sur plusieurs sources et d'en extraire les informations. Ensuite il télécharge les différentes images.

Le sélecteur utilise les images qui ont été téléchargées par le crawler, il effectue ensuite la détection de la redondance, l'extraction des métadonnées des différentes images pour ensuite stocker les programmes, les images associées ainsi que leur métadonnées dans une base de données XMLTV.

4 Structure générale du système

Le diagramme de composant (Figure 2 (Chapitre 2)) permet de décrire la structure interne du système. On peut découper le système en 5 composant principaux découpés dans deux logiciels.

4.1 Le crawler

Deux composants sont présents dans le crawler :

— Extraction des programmes : ce composant permet suivant la source (XMLTV [WWW0] ou Web) de récupérer les informations du programmes ainsi que le lien vers les différentes

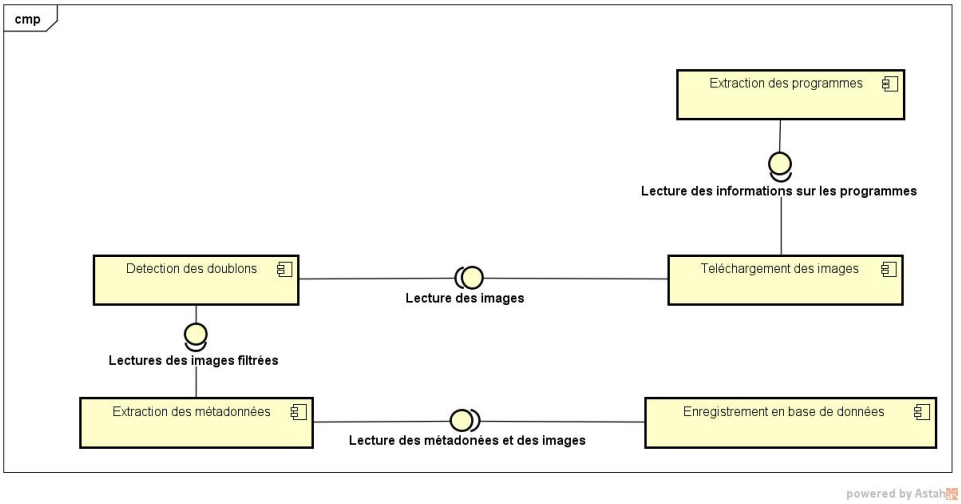


Figure 2 – *Diagramme de composants*

images. Il envoi ensuite au composant 2 la liste de ces informations de manière interne pour qu'il puisse les traiter.

— Téléchargement des images : Ce composant permet à partir de la liste des informations fournies par le composant 1 de télécharger l'ensemble des images et de les stocker en local selon le système de fichier définie 1.1 (Annexe C).

4.2 Le sélecteur

Le sélecteur est composé de 3 composants :

— Détection des doublons : Ce composant permet de lire les images stockées sur le disque par le crawler, ensuite détecte les doublons et supprime celles qui ont la moins bonne qualité (résolution/compression).

— Extraction des métadonnées : Ce composant prend en paramètre les images précédemment filtrées et en extrait un certain nombre de métadonnées (Copyright, dimensions, auteur ...)

— Enregistrement en base de données : On récupère les images filtrées et annotées et on les enregistre dans une base de données XMLTV.

3 État de l'art/veille

1 Crawler

1.1 Qu'est ce qu'un crawler ?

Un web crawler, ou robot d'indexation, est un logiciel permettant d'explorer automatiquement le web. Il est généralement conçu pour récupérer différentes ressources (contenu de pages web, images, vidéos...) pour des usages variés :

— Indexation pour moteur de recherche (google)
— Création de bases de données spécialisées (imdb, allociné)
— Comparateurs de prix (https://www.quiestlemoinscher.leclerc/)

1.2 Les principes de l'indexation

Un crawler parcourt le web de manière récursive à partir d'une page de départ, c'est-à-dire qu'il récupère tous les liens sortant de chaque page pour ensuite les visiter. Il se compose généralement de plusieurs éléments (Figure 1 (Chapitre 3))[0].

Un crawler utilise la combinaison des principes suivants :

— Principe de sélection, qui définit la liste des pages à visiter ;
— Le principe de re-visite, qui définit la fréquence de vérification des changements dans les pages ;
— Le principe de politesse, qui définit comment éviter les surcharges des pages web ;
— Le principe de parallélisation, qui définit comment coordonner les robots d'indexations distribués.

1.3 Éléments définissant un crawler

Un crawler possède un certain nombre de fonctionnalités ; certaines sont requises, d'autres sont optionnelles.[0] [WWW0][WWW0]

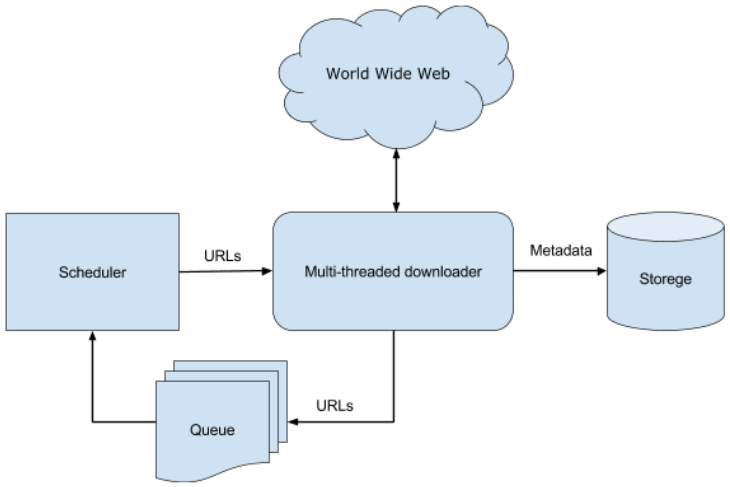


Figure 1 – *Principe de base d'un crawler*

1.3.1 Robustesse

Certains sites contiennent ce qu'on appelle des "Spider traps", c'est-à-dire des pièges à crawlers. Les serveurs génèrent des pages qui trompent les crawlers et les incitent à parcourir certaines pages en boucle, indéfiniment. Les crawlers doivent donc être résistants à ce genre de pièges. Tous les pièges ne sont pas intentionnels, ce sont la plupart du temps des erreurs de développement. Par exemple, un site peut mener à l'exploration de dossier infiniment profonds (http ://exemple.fr/a/b/a/b/...), ou générer des pages contenant un nombre indéfini de liens à suivre. La robustesse est l'un des éléments les plus importants pour un crawler.

1.3.2 Politesse

Les serveurs web ont des politiques implicites et explicites qui permettent de réguler à quel taux un crawler peut les visiter. Un fichier robots.txt, présent à la racine d'un site, fournit des informations pour indiquer quelles pages sont accessibles ou quels robots sont autorisés. Il est important de respecter la politique de politesse : dans le cas contraire, un bannissement temporaire est souvent observé.

1.3.3 Qualité

Sachant qu'une fraction importante des données d'un site est inutile, pour répondre au mieux aux besoin des utilisateurs, le crawler devrait se diriger en premier vers les pages utiles, et extraire uniquement les informations pertinentes.

1.3.4 Distribution

Le crawler a la possibilité de s'exécuter de manière distribuée à travers plusieurs machines : grâce à cela, il est possible de s'affranchir des limitations d'une machine unique en en faisant fonctionner plusieurs en parallèle. Cela implique une certaine rigueur dans le développement du crawler, car un système distribué mal conçu peut obtenir des performances médiocres comparé à un système centralisé

1.3.5 Scalabilité

L'architecture du crawler doit permettre d'augmenter le plus linéairement possible la puissance du crawling en ajoutant de nouvelles machines ainsi que de la bande passante supplémentaire, en perdant le moins possible en efficacité.

1.3.6 Performance et efficacité

Le crawler doit utiliser de manière efficace l'utilisation des diverses ressources telles que le processeur, les accès disque ainsi que la bande passante. Plus un robot est efficace, plus il est capable d'obtenir une grande quantité d'informations avec le même matériel.

1.3.7 Actualisation

Dans de nombreuses applications, un robot devrait fonctionner en continu : il devrait obtenir par lui-même les nouvelles versions des pages visitées précédemment. Par exemple, les moteurs de recherche peuvent ainsi garantir que l'index du moteur contient une version récente des pages indexées. Ce faisant, le crawler devrait être capable d'analyser une page à une fréquence qui se rapproche au taux de changement de cette page.

1.3.8 Extensible

Le crawler doit permettre d'ajouter de nouveaux formats de données ainsi que de nouveaux protocoles de récupération, afin de s'adapter aux changements qui peuvent arriver sur les sites. Cela nécessite que l'architecture du crawler soit modulaire.

1.3.9 Crawling large et concentré

Quand on crawl un seul domaine (par exemple Amazon) on est essentiellement limité par la politique de politesse. On ne peut pas submerger le serveur de milliers de requêtes par seconde, autrement le robot se retrouvera bloqué ; nous avons alors besoin de limiter notre nombre de requêtes. À cause de cette limite la plupart des ressources seront en attente. C'est pour cela que pour un crawling concentré, avoir à sa disposition un crawler distribué avec de nombreuses machines ne sera pas plus rapide que de le lancer sur un ordinateur personnel.
Dans le cas d'un crawler large, le goulot d'étranglement est la performance et la scalabilité. Lorsque l'on attaque plusieurs serveurs en même temps, on peut se permettre d'exécuter plusieurs centaines de milliers de requêtes par seconde sans risque de submerger les serveurs. On est donc limité par notre nombre de machines ainsi que leurs ressources (CPU, bande passante...).

2 Extraction des programmes

Les programmes sont disponibles sur des sites différents, ils sont donc stockés d'une manière différentes. Il peuvent être sous la forme d'un XMLTV, d'une page web html/css ou même javaScript. Il faudra alors adapter le crawler suivant la source.

Voici la présentation de JSoup (Permet d'extraire le contenu des pages WEB) et XMLTV (permet de stocker les programmes dans un fichier XML)

2.1 JSoup

Jsoup est une librairie open source qui a pour but de parser, extraire et manipuler les données contenue dans les pages web. Cette librairie va me permettre de créer le crawler qui servira à construire la base de donnée image.[WWW0]

Voici l'exemple qui est fournit sur le site de Jsoup :

```
Document doc = Jsoup.connect("http://en.wikipedia.org/").get();
log(doc.title());
Elements newsHeadlines = doc.select("#mp-itn b a");
for (Element headline : newsHeadlines) {
  log("%s\n\t%s",
    headline.attr("title"), headline.absUrl("href"));
}
```

La première ligne permet de créer un objet `Document` à partir de l'url en paramètre ici c'est la page d'acceuil de wikipédia.

La deuxième ligne utilise la méthode `select()` qui est disponible sur les classe `Document, Element, Elements` cette méthode permet de rechercher des éléments à la manière du CSS. Un sélecteur est un enchainement de sélecteurs séparé par des combinateur, ils sont insensible à la casse.

Voici un liste non exhaustive de sélecteurs :

Modèle	Eléments
*	Tous les éléments
tag	Les éléments avec le tag donné
#id	Les Elements avec comme ID "id"
.classe	Les éléments avec comme nom de classe "classe"

Ainsi que des combinateurs :

Modèle	Eléments
E F	Les éléments E qui sont dans F
E, F, G	Tous les éléments E, F ou G
E > F	Un enfant direct F de E

La deuxième ligne de l'exemple ce dessus indique donc que l'on veut récupérer toutes les balises `<a>` qui sont dans la balise `<b>` et dans des éléments qui portent l'id `mp-itn`.

Si on affiche touts les éléments de `newsHeadlines` on obtient :

```
<a href="/wiki/Soyuz_MS-10" title="Soyuz MS-10">Soyuz MS-10</a>
<a href="/wiki/Hurricane_Michael" title="Hurricane Michael">Hurricane Michael</a>
<a href="/wiki/William_Nordhaus" title="William Nordhaus">William Nordhaus</a>
<a href="/wiki/Paul_Romer" title="Paul Romer">Paul Romer</a>
<a href="/wiki/Portal:Current_events" title="Portal:Current events">Ongoing</a>
<a href="/wiki/Deaths_in_2018" title="Deaths in 2018">Recent deaths</a>
<a href="/wiki/Wikipedia:In_the_news/Candidates" title="Wikipedia:In the ↩
        news/Candidates">Nominate an article</a>
```

Donc les deux dernières lignes permettent de récupérer seulement l'attribut `Title` et le lien dans l'attribut `href`. La méthode `Element.absUrl("href")` est très pratique car ça permet de reconstruire automatiquement le lien absolue à partir du lien relatif.

2.2 XMLTV

Le format XMLTV permet de représenter les programmes de télévision via un format XML. Sa structure est décrite dans une DTD [1].[WWW0] Le site xmltv france utilise l'API [2] de télérama, elle permet de récupérer le programme TV sur plusieurs semaines avec un certains nombre d'informations sur les programmes, elle est mise à jour régulièrement.

Voici un exemple de la structure du XMLTV :

```
<!-- Description d une chaine -->
<channel id="C111.api.telerama.fr">
    <display-name>Arte</display-name>
    <icon src="http://television.telerama.fr/tv/500x500/111.png" />
</channel>

<!-- Description d un programme -->

 <programme start="20181008111000 +0200" stop="20181008115500 +0200" showview="" ↩
         channel="C111.api.telerama.fr">
<icon src="http://television.telerama.fr/169_EMI_693810.jpg" />
    <title>Sur les toits des villes</title>
    <sub-title>Rome</sub-title>
    <desc lang="fr">Saison : 2 - Les toits des villes accueillent des aventures humaines ↩
            hors norme, entre métiers insolites, expériences artistiques ou reconquête ↩
            de la faune et de la flore.</desc>
    <credits>
      <director>Xavier Lefebvre</director>
    </credits>
    <date>2018</date>
    <category lang="fr">documentaire : découvertes</category>
    <length units="minutes">45</length>
    <country>français</country>
    <episode-num system="xmltv_ns">1..</episode-num>
    <video>
      <aspect>4:3</aspect>
      <quality>HDTV</quality>
    </video>
    <previously-shown />
    <rating system="CSA">
      <value>Tout public</value>
    </rating>
</programme>
```

La structure permet de décrire une liste de chaine de télévision via un nom et un icone. Ainsi que d'une liste de programme avec une heure de début, une heure de fin ainsi qu'une certain nombre d'information décrivant le programme. Le plus important est la balise `<icon>` car c'est ici que l'on va récupérer le lien des photos pour créer la base de donnés.

Lors du projet j'utilise le XMLTV pour créer la base de données images, je récupère ainsi la liste des programmes ainsi que les images associées.

3 Détection des duplicata

La détection des duplicatas est un point essentiel car les nombreuses sources proposent souvent les même images ayant subit de légère modifications (découpage, recadrage, compression ...).

1. Document Type Definition, est un fichier qui décrit la grammaire du XML, la liste des éléments, les attributs ainsi que leur agencement
2. Application programming interface

Pour détecter ces doublons on va utiliser des méthodes de "Template matching".

Template Matching

Le "Template Matching" est de manière conceptuel un processus simple. Il permet de faire correspondre une image à un modèle, où le modèle est une sous partie de l'image. En conséquence le processus est de centrer le modèle sur un point de l'image et on compte le nombre de points qui correspondent à l'image. Le processus est répété pour toute l'image. On renvoi alors le point qui possède le nombre maximum de correspondance [0][WWW0].

Le problème est lorsque l'image est tourné ou agrandie la méthode ne fonctionne plus. Pour palier à ce problème on peut utiliser plusieurs modèle qui eux sont agrandis et ayant subit une rotation. L'avantage du "Template Matching" et qu'il est résistant au bruit ainsi qu'au recadrement de l'image ce qui est très courant sur le sites de média. (Figure 2 (Chapitre 3))

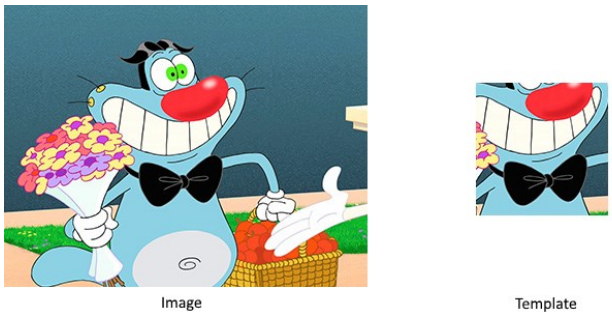


Figure 2 – *Exemple de "Template Matching"*

3.1 Corrélation croisée ou somme des différences absolues

Une approche simple serait d'utiliser une image en niveau de gris. Le résultat de la corrélation croisée sera l'endroit où le modèle correspond le plus à l'image.

On appelle l'image de recherche S(x,y) Où (x,y) représentent les coordonnées de chaque pixel de l'image. De la même manière on appelle le modèle T(x_t,y_t) où (x_t,y_t) représentent les coordonnées de chaque pixels dans le modèle. La méthode consiste à bouger l'origine du modèle T(x_t,y_t) sur tous les (x,y), on calcul alors la somme des produits entre les coefficients de S(x,y) et T(x_t,y_t) sur tout l'espace occupé par le modèle.

Un autre calcul pourrait être de comparer l'intensité des pixels en utilisant **La somme des différences absolues**. Chaque pixel de l'image avec les coordonnées (x_s,y_s) possède une intensité $I_s(x_s,y_s)$ ainsi que les pixels du modèle (x_t,y_t) a une intensité $I_t(x_s,y_t)$. On définie alors la somme des différences absolues (SAD) comme :

$$SAD(x,y) = \sum_{i=0}^{Tlignes} \sum_{j=0}^{Tcolonnes} Diff(x+i,y+j,i,j)$$

où

$$Diff(x_s,y_s,x_t,y_t) = \left|I_s(x_s,y_s) - I_t(x_s,y_s)\right|$$

Lors d'une approche basique le temps de calcul est très important. Si le modèle est carré et de taille m x m et que l'image est de taille N x N. Alors m^2 pixels sont associés sur tout les points de l'image, le temps de calcul est alors de $O(N^2m^2)$. C'est un point négatif, une implémentation basique est très lente. De ce fait, il est alors intéressant d'utiliser des techniques qui fournissent le même résultats mais avec un temps de calcul beaucoup plus rapide comme l'utilisation de la transformation de Fourier et particulièrement la FFT(Fast Fourier Transformation).

3.2 Transformation de Fourier

On peut implémenter le "Template Matching" via la transformation de Fourier en utilisant la dualité entre la convolution et la multiplication. Cette dualité indique que la multiplication dans l'espace correspond à une convolution dans le domaine des fréquences. La transformation de fourrier utilise le domaine de fréquence, on peut alors calculer la corrélation croisé par une multiplication dans le domaine des fréquences.

I représente l'image et T la modèle et $\otimes$ la corrélation, on définie alors :

$$I \otimes T = \sum_{(x,y)\epsilon W} I_{x',y'} T_{x'-i,y'-j}$$

où

$$x' = x + i \text{ et } y' = y + j$$

La convolution est définie comme :

$$I * T = \sum_{(x,y)\epsilon W} I_{x',y'} T_{i-x',j-y'}$$

On peut alors dire que :

$$I \otimes T = I * T' = \sum_{(x,y)\epsilon W} I_{x',y'} T_{i-x',j-y'}$$

où

$$T' = T_{-x,-y}$$

T' a alors subit un retournement horizontal et vertical. Comme dit ci-dessus dans le domaine des fréquences la convolution correspond à une multiplication on peut donc dire :

$$I \otimes T = I * T' = F^{-1}(F(I) * F(T'))$$

Ici F représente la transformation de fourrier et F^{-1} représente l'inverse de la transformation de fourrier.

Grâce à cette équation on peut résoudre le même problème d'une manière beaucoup plus rapide. L'inconvénient est que les deux images doivent être de la même taille et comme par définition le modèle est plus petit que l'image on doit alors utiliser ce qu'on appelle le "Zero padding" c'est à dire remplir le reste du modèle de zero pour qu'il fasse la même taille que l'image. Le modèle est alors reconnu là ou la fréquence est la plus élevé.

3.2.1 Plugin ImageJ

ImageJ possède un plugin FFT qui permet d'appliquer ces méthodes, le problème est que comme c'est un plugin il est facilement utilisable via l'interface graphique et très compliquer à utiliser comme une API via le code.

3.2.2 Classe Java

Il existe des classes Java qui peuvent être téléchargées et qui contiennent toutes les méthodes nécessaire pour utiliser la FFT, il suffit juste d'appeler les méthodes voulues.[WWW0]

3.2.3 Matlab

MatLab possède les méthodes fft2 [3], fftshift, ifft2 de déjà implémenté ce qui permet de tester assez rapidement si les méthodes fonctionnent pour ensuite pouvoir les implémenter en Java.

3. Fast Fourier Transformation 2D : Applique la transformation de Fourier optimisé pour les images/matrices

4 Métadonnée image

Les métadonnées est une donnée servant à définir ou décrire une autre donnée, ce sont des informations structurés qui décrivent les documents en vue de faciliter la gestion des ressources[WWW0]. Elles sont sous la forme de mots ou de texte et contiennent certaines informations comme :

- Références du document : titre, auteur, année, sujet, éditeur, etc.
- Son statut administratif : appartenance à une collection, copyright.
- Son contenu informatif : descripteur, résumé.
- ...

Les métadonnées permettent des recherches pertinentes, précises et rapides.

Que ce soit dans un document numérique ou non les métadonnées peuvent prendre place :

- Dans le document lui-même
- En dehors du document (ce qui nécessite une référence entre le document et ses métadonnées)
- Sous une forme mixte

Pour les images il y a deux format principaux pour les métadonnés.

- IPTC : Information de catalogage enregistrées volontairement et relatives à l'auteur, au statut administratif et au contenu informatif de l'image.
- Exif : Information technique enregistrées automatiquement lors de la création du fichier image par un appareil photo numérique il est déconseillé de les modifier.

Les tags EXIF sont stockés au début de l'image (binaire), ils sont composé d'un tag, d'une clé ainsi qu'un type[WWW0].(Figure 3 (Chapitre 3)).

Tag (hex)	Tag (dec)	IFD	Key	Type	Tag description
0x000b	11	Image	Exif.Image.ProcessingSoftware	Ascii	The name and version of the software used to post-process the picture.
0x00fe	254	Image	Exif.Image.NewSubfileType	Long	A general indication of the kind of data contained in this subfile.
0x00ff	255	Image	Exif.Image.SubfileType	Short	A general indication of the kind of data contained in this subfile. This field is deprecated. The NewSubfileType field should be used instead.
0x0100	256	Image	Exif.Image.ImageWidth	Long	The number of columns of image data, equal to the number of pixels per row. In JPEG compressed data a JPEG marker is used instead of this tag.
0x0101	257	Image	Exif.Image.ImageLength	Long	The number of rows of image data. In JPEG compressed data a JPEG marker is used instead of this tag.

Figure 3 – *Tag EXIF*

De même pour les tags IPTC (Figure 4 (Chapitre 3))

4.1 Méthode d'extraction

Il existe plusieurs logicielles qui permettent d'extraire ces données mais ici ce que nous cherchons c'est une librairie qui permet de les extraire en Java.

IPTC datasets defined in Exiv2

Datasets are defined according to the specification of the IPTC Information Interchange Model (IIM).

Tag (hex)	Tag (dec)	Key	Type	M.	R.	Min. bytes	Max. bytes	Tag description
0x006e110		Iptc.Application2.Credit	String	No	No	0	32	Identifies the provider of the object data
0x0073115		Iptc.Application2.Source	String	No	No	0	32	Identifies the original owner of the intellectual content of the object data. This could be an agency
0x0074116		Iptc.Application2.Copyright	String	No	No	0	128	Contains any necessary copyright notice.

Figure 4 – *Tag IPTC*

4.1.1 Java pur

Java fournit des solutions qui permettent d'extraire les métadonnées d'une image via le package `javax.imageio.metadata`. Le problème est que nous devons récupérer l'ensemble des métadonnées, former un arbre xml et ensuite extraire les données que nous voulons c'est fastidieux et non robuste.

4.1.2 ImageJ

ImageJ permet de charger des images via la classe `ImagePlus` et lors du chargement de l'image il créer un objet `FileInfo` qui est censé contenir un certain nombre de métadonnées. Or lors des essaies il s'avérait que l'objet ne charge que quelques informations (Format, dimension, nom) et que le reste des information n'était pas complétées. La librairie étant open source après analyse du code on se rend compte que effectivement les données ne sont jamais chargées.

4.1.3 La librairie matadata-extractor

La librairie Open-source "metadata-extractor" permet d'extraire les métadonnées Exif, IPTC, XMP, ICC ... depuis des images et vidéos. La librairie est assez simple d'utilisation, de plus elle permet de lire les métadonnées sans ouvrir entièrement l'image ce qui permet un temps de traitement amélioré.[WWW0]

Exemple :

```
Metadata metadata = ImageMetadataReader.readMetadata(imagePath);
```

Avec l'instance metadata on peut itérer sur l'ensemble des métadonnées ou faire des requêtes pour récupérer l'information que nous souhaitons. Si nous souhaitons récupérer la date de création de l'image il nous suffit d'exécuter le code suivant.

```
1
2 ExifSubIFDDirectory directory
3     = metadata.getFirstDirectoryOfType(ExifSubIFDDirectory.class);
4
5 Date date = directory.getDate(ExifSubIFDDirectory.TAG_DATETIME_ORIGINAL);
```

4 Analyse et conception

On va principalement s'intéresser aux sites de guide TV. D'après plusieurs études de cas on remarque que dans les 20 premiers sites de Guide TV présent sur google il y a un grand nombre d'image similaire pour un même programme. Le but de notre sélection et de récupérer les images des programmes sur de nombreux sites pour ensuite pouvoir supprimer les doublons et garder les images de meilleur résolution. On pourra ensuite extraire les métadonnées de ces images et avec les informations récupérées sur les sites pouvoir ainsi construire une base de données conséquente avec une première sélection fiable.

1 Crawler mono-source

Rapidement dans le projet j'ai du créer un crawler mono-source, du site http://www.xmltv.fr/ qui est un XMLTV créé à partir de l'api de Télérama, le but était de construire une première base de données pour pouvoir étudier un certain nombre de points. Le principe était de lancer le crawler assez régulièrement pour construire la base petit à petit sur 4 mois.(Figure 1 (Chapitre 4))

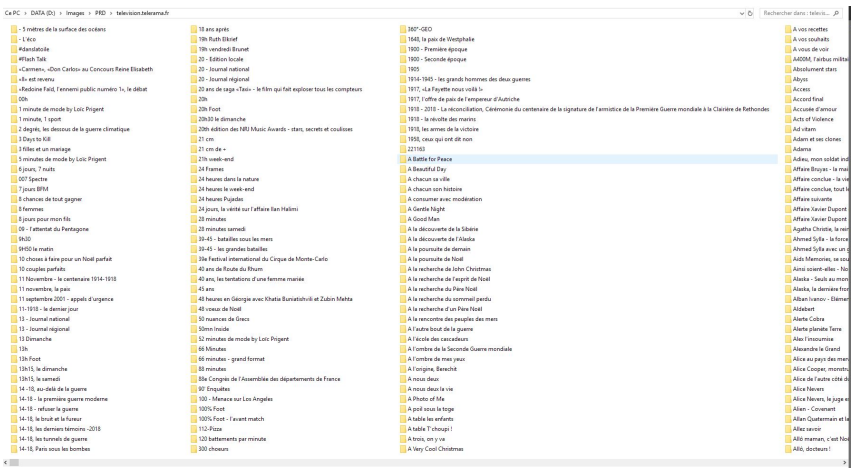


Figure 1 – *Base de données*

Au bout des 4 mois j'ai réussi à récupérer 11 000 images qui correspondent à 4000 programme TV différents pour un total de 2Go. On remarque qu'il y a environ 1000 nouvelles images chaque

semaines pour 200 nouveaux programmes. Ça implique qu'il y a un grand nombre d'images pour chaque programme ce qui permet de pouvoir ensuite les choisir. (Figure 2 (Chapitre 4))

Figure 2 – *Exemple d'images pour le programme Oggy et les cafards*

Toutes les images sont dans le format JPG et une grande partie en 720p. Avec ces informations on peut supposer que c'est surement similaire sur les autres site.

2 Crawler multi-source

Les sources peuvent être multiple (Api, XMLTV, HTML, JavaScript) mais dans le projet nous nous concentrerons sur des sources XMLTV ainsi que HTML. Chaque site possède une architecture HTML différente donc si nous voulons parser le contenu des sites il sera nécessaire d'avoir un code d'extraction des images par site tout en essayant que le code du crawler soit commun pour tout les sites. Le but sera d'extraire l'image d'un programme donnée pour pouvoir ensuite les faire correspondre aux images des autres sites. (Figure 3 (Chapitre 4), Figure 4 (Chapitre 4))

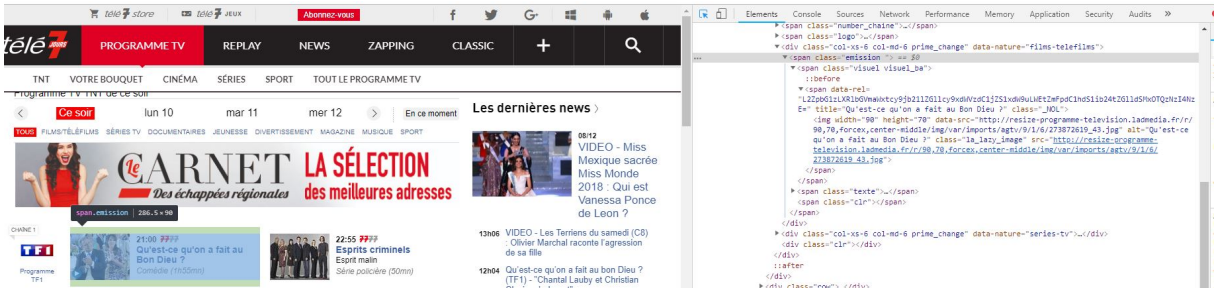


Figure 3 – *Exemple de code HTML pour un programme sur Tele 7 jours*

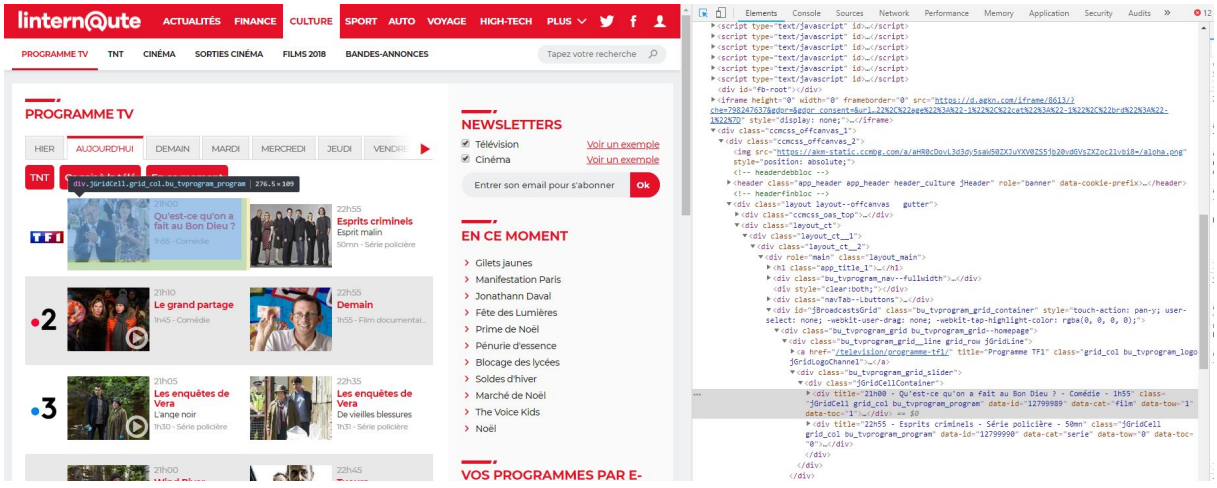


Figure 4 – *Exemple de code HTML pour un programme sur linternaute*

Pour parser le code HTML j'utiliserai la librairie JSoup qui me permettra facilement d'extraire les données que je souhaite dans la page. Il permet aussi de télécharger les images ainsi qu'a faire les requêtes HTTP. Pour pouvoir associer les programmes des différents sites chaque programme sera défini par :

— Une date de début
— Une date de fin
— Une chaine
— Un titre

Grâce à ces informations on pourra ainsi avoir une liste d'images pour un programme donné. Donc pour chaque programme on va extraire l'ensemble de ces informations. Le crawler étant un programme indépendant voici un début de modélisation (Figure 5 (Chapitre 4)).

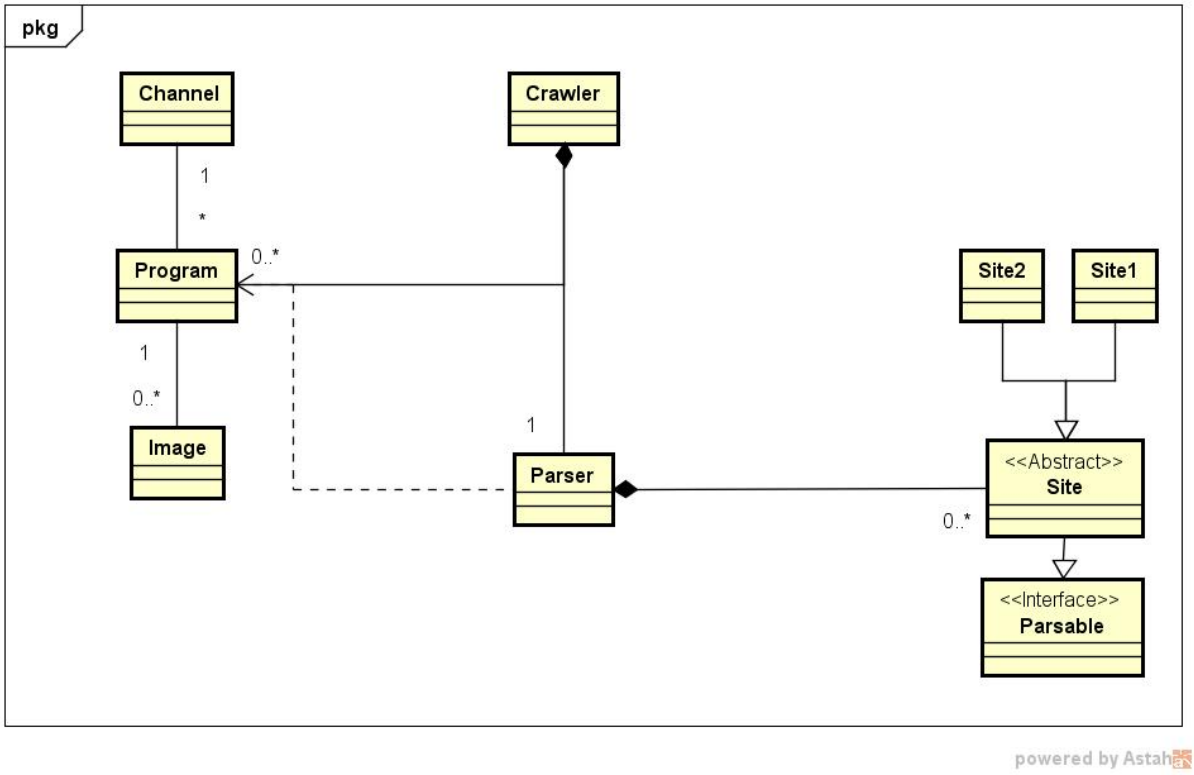


Figure 5 – *Diagramme de classe pour le crawler*

Sur le diagramme on voit que chaque programme est composé d'une chaine ainsi que d'une liste d'images. Le crawler lui pour chaque site définie accède à la page d'accueil et extrait grâce au parseur les différents programmes et les renvoi au crawler qui peut ensuite les stocker.

De plus le parseur renvoie la liste des liens vers les autres pages pour que le crawler puisse parcourir l'ensemble du site et des programmes. Le crawler possède une liste des sites visités pour ne pas les revisiter. Une fois que tout les crawlers ont fini d'extraire les programmes on va ensuite télécharger toute les images et les stocker.

Selon les propriétés des crawlers dans 1.3 (Chapitre 3)notre crawler devra respecter les règles suivantes :

— Robustesse
— Qualité
— Extensible

En effet étant automatique il doit pouvoir être fiable et ne pas crasher au moindre problème. Il

doit pouvoir extraire exactement les données nécessaires pour pouvoir extraire un programme. On doit pouvoir être capable d'ajouter de nouvelles sources au crawler à moindre coût.

3 Detection des duplicata

Pour la sélection des duplicatas on va procéder programme par programme, nous allons utiliser la méthode FFT pour la détection des duplicatas.

Nous allons parcourir tous les images du programme et on va utiliser chacune des images restante comme un modèle, une fois que toute les images ont été parcourues il sera alors facile de détecter les images redondantes.

La FFT ayant besoin d'un modèle carré d'une taille multiple de 2. Nous allons prendre le centre de l'image d'une taille 128 pixels et l'utiliser comme modèle. Les images étant le plus souvent similaire ou tronqué ce n'est pas un problème d'avoir seulement le centre pour une méthode de "Template Matching". Pour détecter si l'image est un doublons ou non nous allons fixer un seuil et si un point du "Template Matching" est supérieur à ce seuil on pourra alors dire que c'est un doublons.

Le programme devant être portable nous utiliserons le plugin ImageJ si nous réussissons à l'utiliser ou alors la classe FFT.java

4 Extraction des metadonées

Pour extraire les métadonnées nous avons soit le choix entre des méthode présente dans le SDK java qui sont bas-niveau ou alors la librairie "Drew metadata-extractor" qui est une librairie open source.

Après quelques tests nous allons utiliser la librairie car elle est facile à mettre en œuvre et permet de récupérer l'ensemble des tags EXIF ainsi que IPTC. Nous allons dans un premier temps récupérer des informations comme la résolution, l'auteur, le copyright et suivant nos besoins on pourra les étendre à moindre coût.

5 Mise en œuvre

Comme dit précédemment je vais réaliser deux programmes distincts, un crawler multi-source, ainsi qu'un autre programme sous forme d'un POC qui permettra de tester des méthodes de détection de doublons et d'extraction de métadonnées.

1 Crawler multi-sources

1.1 Les librairie

le crawler est réalisé avec Java 8 avec l'aide de Maven pour la gestion des librairie et du projet. Les librairies qui ont été utilisées sont :

- Jsoup : Pour la gestion de l'accès au sites ainsi que l'extraction des données.
- Log4J : Cette librairie permet de gérer les log, pour ainsi avoir un meilleur affichage et la sauvegarde dans des fichiers de logs.

1.2 Implémentation

L'implémentation du crawler suit le diagramme de classe suivant : Pour plus d'information sur le diagramme voir Annexe G

1.2.1 Architecture du crawler

Le crawler est multi-threadé, j'ai donc effectué l'architecture suivante Figure 2 (Chapitre 5) . En effet nous avons un thread par source, ce thread va contenir la liste des url à visiter ainsi que la liste des programmes extraits. Ce thread va lancer plusieurs sous thread qui vont piocher dans la liste des url -> parcourir la page pour en extraire tous les liens -> extraires les programmes de la pages. Figure 3 (Chapitre 5) .

Une fois que tous les threads ont finis leur travaille c'est la classe Scheduler qui va s'occuper de la fusion des programmes, en effet tous les programmes du même titre sont considéré comme identique, on va donc ajouter toutes les images de ce programme dans un seul objet.

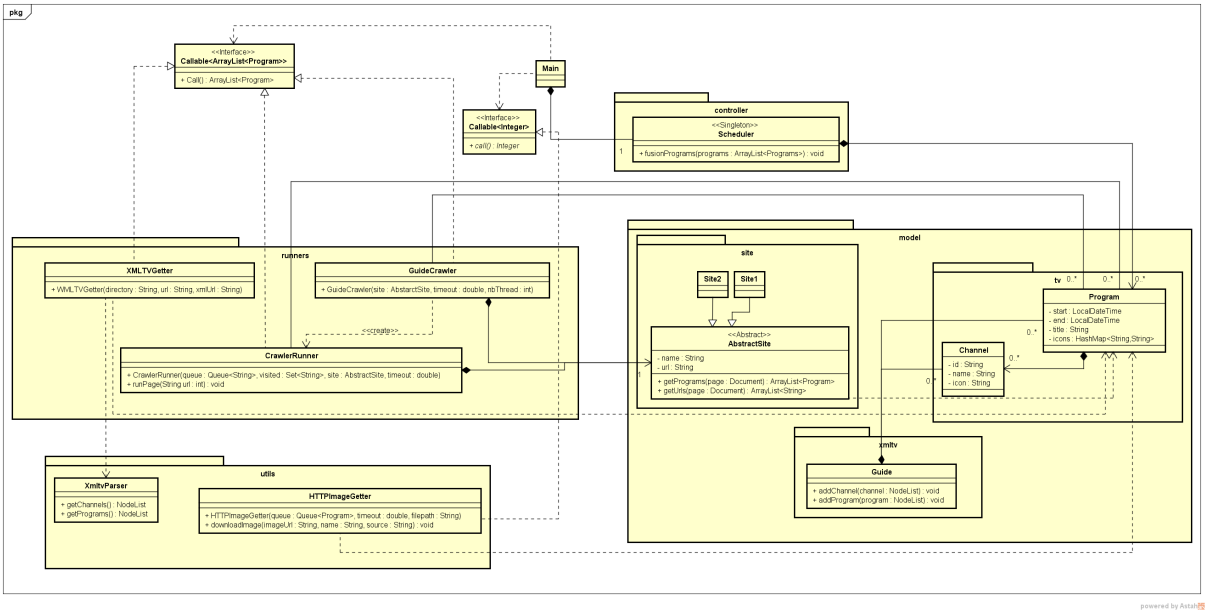


Figure 1 – *Diagramme de classe du crawler*

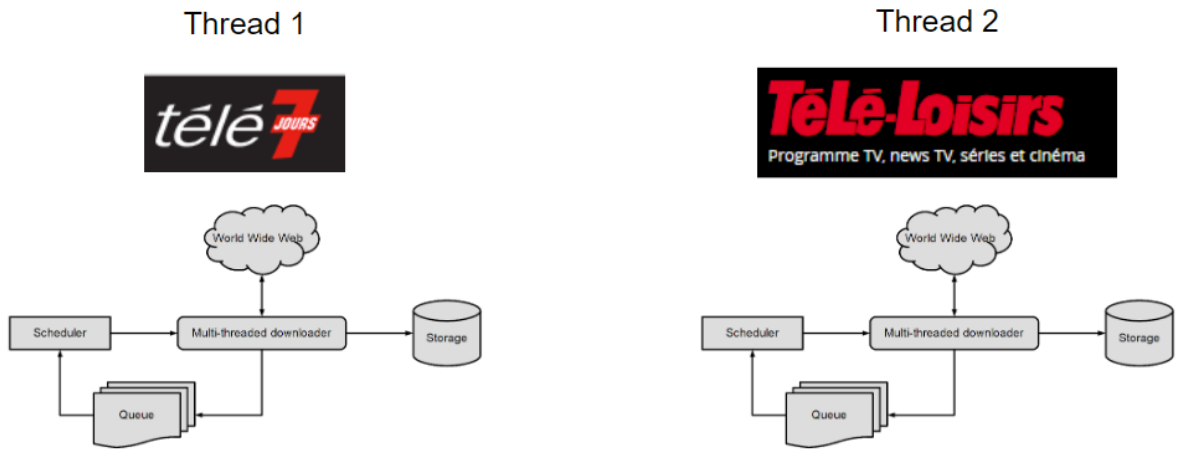


Figure 2 – *Architecture du crawler*

Une fois les programmes regroupés c'est le downloader qui rentre en jeu et qui va lui aussi de manière multi-threadé télécharger l'ensemble des images des différents programmes et les stocker sous l'architecture suivante Figure 4 (Chapitre 5).

Nous avons donc un dossier par programme et dans ce dossier la liste des images téléchargées pour ce programme sous la forme "Source _ Nom de l'image"

1.2.2 Extraction des programmes

L'extraction des programmes se fait grâce à la librairie JSOUP, elle permet d'accéder aux pages et de retourner le contenu de la page HTML et de pouvoir le parser facilement. Figure 5 (Chapitre 5)

Les différents sites de programmes TV ont souvent la même architecture. Il y a la page principale qui contient la grille des différents programmes. Et il y a aussi une page par programme, c'est

```java
@Override
public ArrayList<Program> call() throws Exception {
    double start = System.currentTimeMillis();
    double end;
    double faultCounter = 0;
    boolean stop = true;
    while(stop) {

        String url = queue.poll();
        if(url != null) {
            runPage(url);
            faultCounter = 0;
        }
        else {
            Thread.sleep( millis: 1500);
            LOGGER.info("No links site {} Thread {} size {}", site.getName(), Thread.currentThread().getName(), queue.size());
            faultCounter++;
        }

        end = System.currentTimeMillis();

        if(end - start >= timeout) {
            stop = false;
        }
        if(faultCounter >= 6)
            stop = false;
    }
    LOGGER.info("Ended : {}", Thread.currentThread().getName());
    return this.programs;
}
```

Figure 3 – *Architecture des runners*

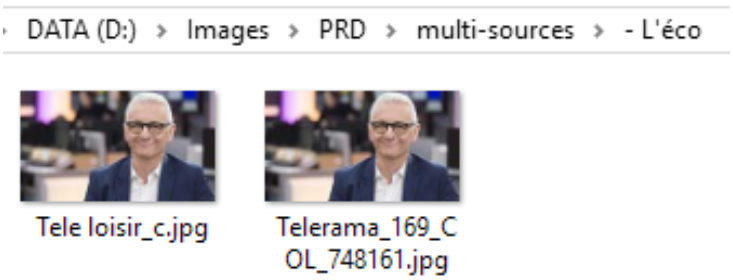


Figure 4 – *Architecture de la base de données*

cette page qui va nous intéresser car elle va nous permettre d'extraire toutes les informations du programme facilement.

De plus certain site utilisent des balise "og" ces balise permettent au site de fournir les informations essentielles de la page pour que lorsque nous partageons le lien de la page sur des réseaux sociaux il puissent afficher divers informations.

Ces balises vont énormément nous servir car dans de nombreux cas on va pouvoir extraire toutes les informations que nous cherchons dans ces balises et nous n'avons pas besoin d'analyser le site en détail pour en extraire les données.

1.2.3 Les liens

La plupart des sites utilise l'url des images pour pouvoir les dimensionner ou les cropper. Pour télécharger l'image originale il faut donc analyser les url pour trouver une astuce et ainsi télécharger l'image originale. Figure 6 (Chapitre 5)

Ici sur l'image on remarque l'ajout de "/r/193,149,forcex,center-middle". On peut en déduire que le crop se fait grâce à cet parti là de l'url. Et effectivement lorsque l'on l'enlève on accède bien à l'image originale.

Figure 5 – *Extraction des programmes*

Figure 6 – *Astuces avec le lien des images*

1.3 Résultats

Il y a un fichier de configuration pour choisir quels éléments crawler Figure 1 (Annexe E). J'ai ajouter 5 sites ainsi que l'extraction du XMLTV. Après 2 mois voici la base de données Figure 7 (Chapitre 5) .

Elle contient près de 52 000 images pour 16 000 programmes pour un total de 7 Go. Concernant la vitesse de crawling, si nous n'avons pas lancé le crawler depuis un certain temps nous pouvons avoir jusqu'à 5000 images pour 5 minutes de crawl.

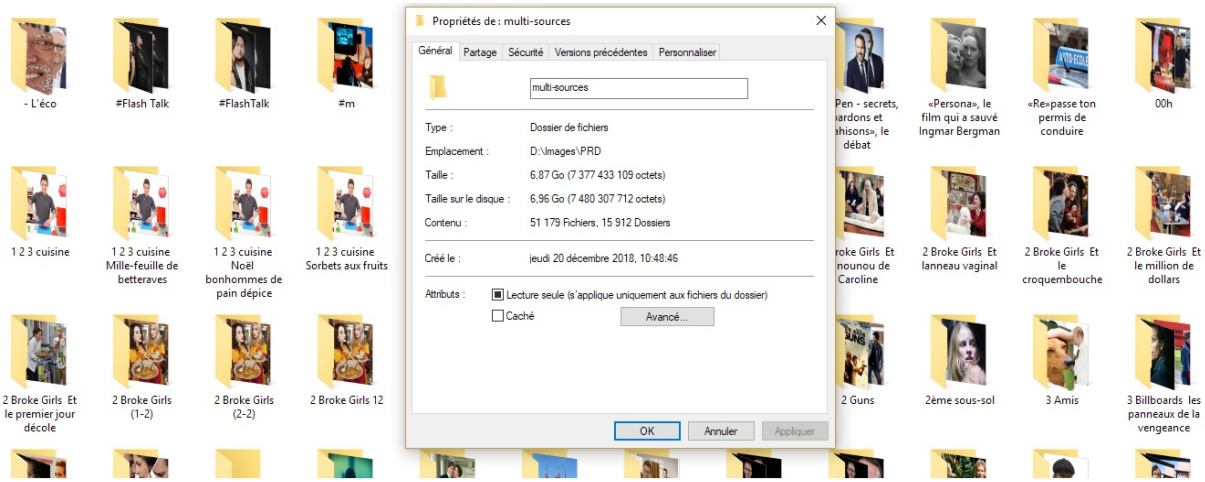


Figure 7 – *Base de données du crawler multi-source*

1.4 Risques/Limites

Les sites n'utilisent pas forcément le même nom pour les programmes ce qui fait que les programmes ne sont pas détecte comme les même alors que ça l'est. Pour remédier à ce problème nous aurions pu utiliser des méthodes de distance entre les titre mais ce n'est pas l'objectif du PRD.

Etant un crawler nous ne sommes pas a l'abri d'un site change de structure et que le crawler ne soit plus compatible, il faudra alors mettre à jour le code en fonction.

2 Détection des doublons et extraction des métadonnées

2.1 Les librairies

Le POC est réalisé avec Java 8 avec l'aide de Maven pour la gestion des librairie et du projet. Les librairies qui ont été utilisées sont :

— ImageJ : Pour la gestion des images et des opération dessus
— Log4J : Cette librairie permet de gérer les log, pour ainsi avoir un meilleur affichage et la sauvegarde dans des fichiers de logs.
— Drew-Metadata-extractor : Pour la gestion des métadonnées

2.2 Implémentation

2.2.1 Les différents type de doublons

Après étude de la base de données nous avons listés les différents type de doublons qui sont présent dans la base :

— Identique
— redimensionné
— Identique avec petit décalage de pixels

— Cropé
— Changement de teinte
— Cropé + redimensionné

De ce constat on peut les diviser en deux catégories :

— Identique, redimensionné, décallé, changement de teinte
— Cropé et cropé + redimenssioné

Nous avons décidé de nous consacrer sur la première catégorie pour pouvoir rentrer dans les délais. Pour ce faire nous avons tout d'abord crée une base de test.

2.2.2 La base de test

L'architecture de la base de test est la suivante Figure 8 (Chapitre 5). . Nous avons un dossier

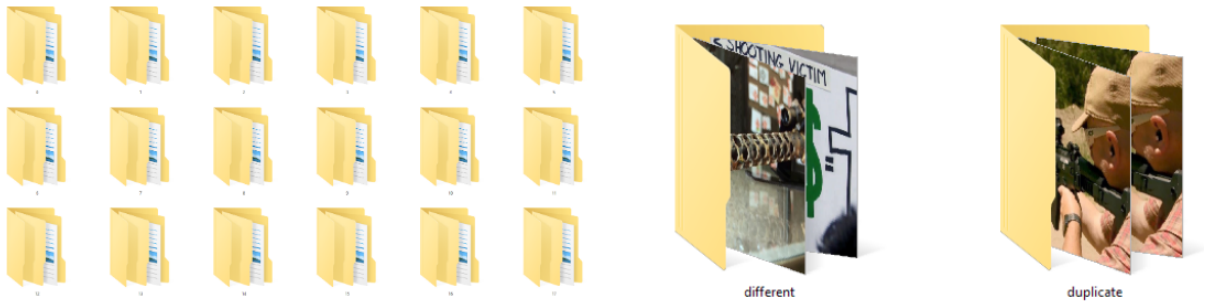


Figure 8 – *Architecture de la base de test*

par programme et dans chaque dossier 2 dossiers distinct :

— Duplicate : Une liste de doublons de catégorie 1 du programme
— Different : Une liste d'image du programme différent de l'image du dossier duplicate

Cette base étant déjà classifié nous allons pouvoir tester nos méthode de détection de doublons dessus et ainsi pouvoir connaître sa performance.

2.2.3 Les différentes distances

Nous avons décider de tester 3 méthodes différentes :

— SSD : Sum of Squared Differences, la distance est entre 0 et 1
— SAD : Sum of Absolute Distance, la distance est entre 0 et 1
— NCC : Normalized Cross correlation, c'est une corrélation entre -1 et 1 si c'est -1 les images sont anti-corrélé et donc différentes, si c'est 1 elle sont corrélé et donc des doublons.

2.2.4 La pipeline

Voici le déroulement de l'algorithme.

Pour chaque couple d'image dupliquées et chaque couple image dupliqué/image différentes :

— 1 : Conversion des images en niveau de gris
— 2 : Redimensionnement de l'image la plus grand à la taille de l'image la plus petite
— 3 : Extraction de centre de l'image jusqu'a 256x256 selon la taille de l'image

— 4 : Calcul de la distance entre les deux images

Une fois que c'est fait le programme nous génère un fichier avec l'ensemble des distances entre les images dupliqués et les images différentes. Grâce à ces fichiers nous allons pouvoir effectuer des statistiques concernant les méthodes.

2.2.5 Analyse des résultats

Grâce au fichiers voici les courbes que nous obtenons Figure 9 (Chapitre 5)

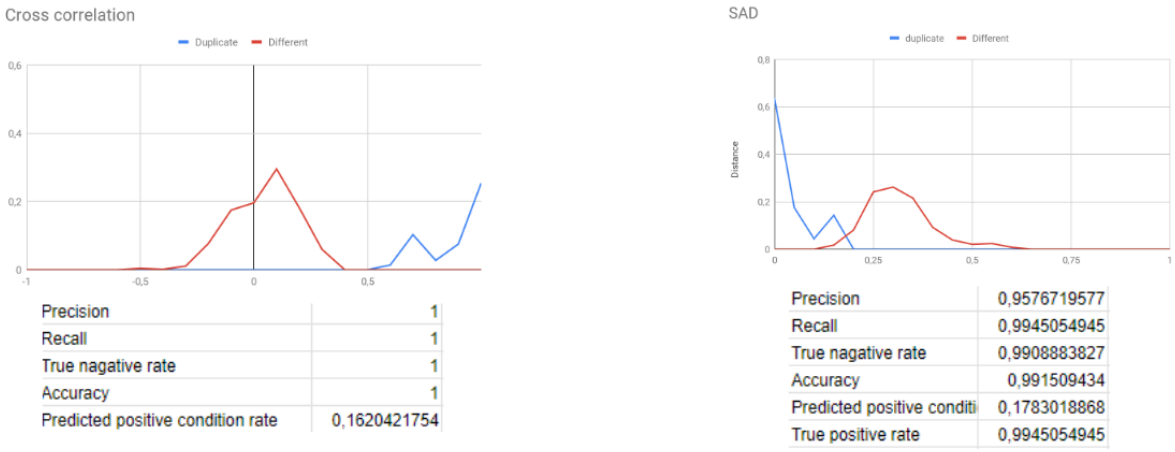


Figure 9 – *Analyse des méthodes*

En ordonnées nous avons la fréquence et en abscisse nous avons la distance/corrélation. En rouge nous avons les images différentes et en bleu les images dupliquées.

Grâce à cette courbe on peut choisir un seuil qui sera l'intersection des deux courbes. Ce seuil nous permettra de déterminer si un image est dupliqué ou non.

Avec ce seuil et les courbes on peut calculer la "Precision and recall", ça nous permet de calculer le taux de faux positif et négatif de la méthode pour savoir si elle est fiable ou non.

Dans notre cas on remarque que pour la méthode NCC les deux courbes sont séparées on a donc une fiabilité de 100%

Alors que pour la méthode SAD on remarque que les courbes se croisent, on aura donc des faux positif et négatif. La précision est de 95 %

2.2.6 L'extraction des métadonnées

L'extraction des métadonnées se fait simplement grâce à la librairie. Nous stockons dans une classe les informations suivantes :

— Longueur de l'image
— Largeur de l'image
— Copyright de l'image

6 Bilan et conclusion

1 Tâches effectuées

Les tâches qui ont été effectuées durant ce semestre sont :

- Comprendre le sujet, connaître l'objectif et commencer à réfléchir aux solutions
- Étude du contexte et des enjeux
- Étude de XMLTV et JSoup
- POC crawler mono-source
- État de l'art
- Cahier de spécifications
- Étude des métadonnées
- Analyse et prise en main de la FFT
- Rédaction de l'analyse et conception
- Préparation de la soutenance de mi-parcours
- Rédaction du rapport S9

Il n'y a pas de tâches en retard pour le S9

2 Planning S10

Comme précisé dans 1.2 (Annexe A)le S10 sera découpé en 4 sprints dont 3 principaux :

- Sprint 1 : Création d'un crawler mutli-source
- Sprint 2 : Detection de la redondance
- Sprint 3 : Extraction des métadonnées et ajout dans la base de données

Le Sprint 4 est composé essentiellement de la finition du rapport ainsi que la préparation de la soutenance.

3 Tâches effectuées S10

Les tâches qui ont été effectuées durant ce semestre sont :

- Création d'un crawler multi-sources parfaitement fonctionnel et extensible

— Tests de 3 méthodes de detection de la redondance SSD, NCC, SAD
— Extraction des méta-données

Je n'ai pas pu tester de méthodes de détection de doublons qui prenne en compte le crop + le redimensionnement. Je n'ai pas pu sauvegarder les résultats dans une base de données.

4 Bilan sur la qualité

La qualité est assuré grâce à de nombreux aspects :

— Diagramme de classe des deux programmes
— Tests unitaires des deux programmes
— Cahier de tests
— Manuel utilisateur
— Document d'installation
— Manuel du développeur
— Javadoc complète

5 Bilan auto-critique sur la gestion de projet

Il est difficile de prévoir à l'avance tout ce que l'on va effectuer pour le semestre suivant, surtout concernant le découpage des tâches et les durées. Dans mon cas il y a eu quelques tâches qui n'était pas bien estimées mais dans l'ensemble elles ont été respecté. Un autre point est qu'il y a toujours des imprévus qui retardent les tâches c'est pour cela que c'est difficile de prévoir l'avenir.

6 Conclusion

Le PRD est un exercice enrichissant, il permet d'avoir un aperçu de ce qu'est la recherche et de mettre en pratique les différentes compétences que nous avons acquises lors de notre cursus.

Mon sujet était très intéressant car il traite des problématiques actuelles sur des plateforme nouvelles. De plus il m'a permit de travailler sur les crawlers qui est un sujet qui m'intéresse énormément. La partie traitement de l'image et extraction des métadonnées était intéressante aussi car elle ma permit d'apprendre de nouvelles choses.

On peut dire que le projet a été réussi car j'ai pu fournir un crawler multi-source parfaitement fonctionnel et extensible. De plus j'ai testés 3 méthodes de détection de doublons NCC, SAD et SSD et j'ai montré que la méthode NCC était parfaitement adapté au problème avec une précision de 100%.

Annexes

A Planification

1 Découpage des tâches

1.1 Tâches semestre 9

Voici un aperçu des tâches qui ont été réalisées durant le semestre 9. (Figure 1 (Annexe A))

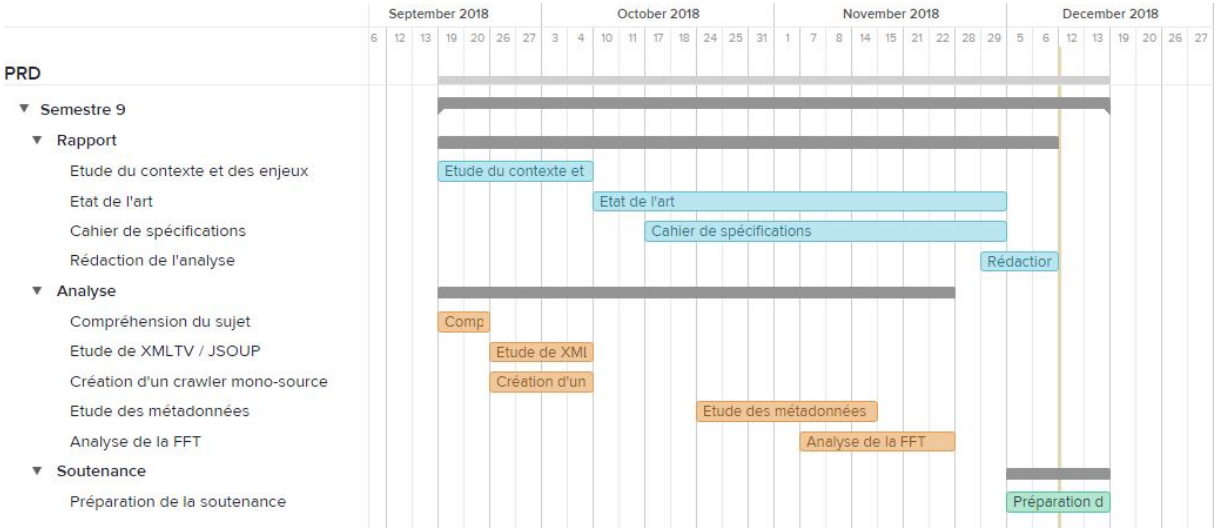


Figure 1 – *Tâches S9*

1-Compréhension du sujet

- Date de début : 19/09/18
- date de fin : 20/09/18
- Durée : 2 jours
- Description : Comprendre le sujet, connaître l'objectif et commencer à réfléchir aux solutions.

2-Etude du contexte et des enjeux

— Date de début : 19/09/18
— date de fin : 04/10/18
— Durée : 6 jours
— Description : Etude du contexte et des nombreux site de Guide TV / fournisseur de contenue. Etude des technique de selection de gros site (Netflix, Youtube). Création d'une étude de cas des images proposées par un grand nombre de sites pour un programme donné. Rédaction du rapport associés à toute ces recherches.

3-Etude de XMLTV et JSoup

— Date de début : 26/09/18
— date de fin : 09/10/18
— Durée : 4 jours
— Description : Etude du format de stockage XMLTV ainsi que de la librairie JSoup.

4-POC crawler mono-source

— Date de début : 26/09/18
— date de fin : 09/10/18
— Durée : 4 jours
— Description : Création d'un crawler mono-source XMLTV pour tester les méthode pour parser le XMLTV ainsi que de prendre en main la librairie JSOUP

5-État de l'art

— Date de début : 10/10/18
— date de fin : 29/11/18
— Durée : 14 jours
— Description : Écriture de l'état de l'art en parallèle des différentes recherches.
— Livrable : Le rapport est à rendre.

6-Cahier de spécifications

— Date de début : 17/10/18
— date de fin : 29/11/18
— Durée : 12 jours
— Description : Rédaction du cahier de spécifications.
— Livrable : Un cahier de spécification est à rendre.

7-Étude des métadonnées

— Date de début : 24/10/18
— date de fin : 14/11/18
— Durée : 6 jours
— Description : Études des différents type de métadonnées, des différents moyen de récupérer ces données. Réalisation d'un programme pour tester la solution choisie.

8-Analyse et prise en main de la FFT

— Date de début : 07/11/18
— date de fin : 22/11/18
— Durée : 4 jours
— Description : Étude du principe de la FFT et prise en main sous ImageJ et Matlab.

9-Rédaction de l'analyse et conception

— Date de début : 02/12/18
— date de fin : 10/12/18
— Durée : 2 jours
— Description : Rédaction de la partie analyse et conception dans le rapport.

10-Préparation de la soutenance de mi-parcours

— Date de début : 5/12/18
— date de fin : 13/12/18
— Durée : 2 jours
— Description : Création d'un Power Point et préparation à la soutenance.
— Livrable : Un Power Point

1.2 Planning prévisionnel semestre 10

Voici le planning prévisionnel du S10. (Figure 2 (Annexe A))

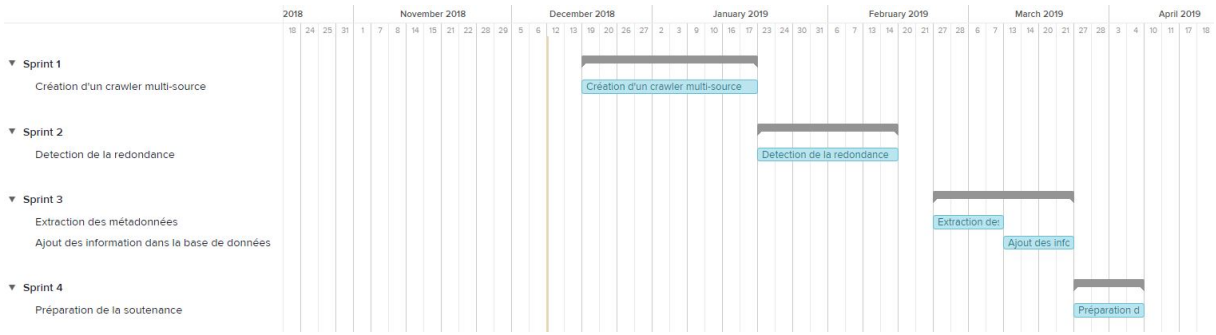


Figure 2 – *Planning prévisionnel S10*

Sprint 1

— Date début : 19/12/18
— Date fin : 17/01/19
— Durée : 6 jours
— Description : Développement du crawler multi-sources
— Livrable : Crawler multi-source

Sprint 2

— Date début : 23/01/19
— Date fin : 14/02/19
— Durée : 8 jours
— Description : Création du programme de sélection et développement de la fonctionnalité "Détection de la redondance"
— Livrable : Programme de sélection avec la fonctionnalité "Détection de la redondance".

Sprint 3

— Date début : 27/02/19
— Date fin : 21/03/19
— Durée : 8 jours
— Description : Ajout de la fonctionnalité "Extraction des métadonnées" ainsi que "Enregistrement des information dans la base de données"
— Livrable : Le programme de sélection terminé.

Sprint 4

— Date début : 27/03/19
— Date fin : 04/04/19
— Durée : 4 jours
— Description : Préparation à la soutenance et finition du rapport.
— Livrable : Rapport final et Power Point

B Description des interfaces externes du logiciel

1 Interface matériel/logiciel

Il n'y a pas d'interface matériel/logiciel, toute les communications se font avec la base de donnée qui sera en local pour le projet.

2 Interface homme/machine

Il n'y aura pas d'interface graphique pour le programme, tout se fera en ligne de commande via la console.

3 Interfaces logiciel/logiciel

Le base de donnés image étant stocké en local, l'accès à la base se fait via le disque dur. Il y a une interaction entre le crawler et le sélecteur mais elle se fait au moyen de la base d'images.

C Spécification fonctionnelles

1 Description des fonctionnalités du crawler

1.1 Définition de la fonction 1 : Création d'un crawler mono-source

Identification de la fonction 1

Création d'un crawler mono-source qui permettra de créer une première base de données, pour analyser le format des images, la quantité etc ...

Priorité : **Primordiale**

Description de la fonction fonction 1

Cette fonction permet de créer une base de données assez conséquente pour pouvoir tester nos méthodes. Il existe un format de fichier nommé XMLTV qui permet de décrire un programme Tv sous la forme d'un XML. Le site xmltv.fr fournit un programme Tv sur plusieurs semaines via L'API de télérama, grâce à ce fichier nous pouvons récupérer le nom de programmes ainsi qu'un lien vers une image du programme, certain programme étant présent régulièrement leur image associé est souvent différente.

La fonction permet donc de télécharger le XMLTv, le parser et récupérer les images des programmes et les sauvegarder. Je lancerai cet outil régulièrement pour pouvoir créer une base de données conséquente d'image et l'avantages est que j'aurais plusieurs images par programmes. L'architecture des fichiers sauvegardés est la suivante Figure 1 (Annexe C) Le programme est fait en Java et utilise la librairie Jsoup pour permettre le téléchargement du XMLTv de manière automatique.

Description précise

— Entrée : Rien
— Sortie : Une liste d'image est sauvegardée
— Exception : Le fichier XMLTV n'est pas valide

Figure 1 – *Structure de la base de données*

1.2 Définition de la fonction 2 : Crawler multi-source

Identification de la fonction 2

Amélioration du crawler précédent pour pouvoir récupérer les images des programmes présent dans le XMLTV sur plusieurs sites de guide TV.

Priorité : **Primordiale**

Description de la fonction fonction 2

Le programme extraits les différents programmes du xmltv ou d'un site puis va télécharger les images associées sur les différentes sources. Les images seront stockées de la même manière que la fonction 1.

Le crawler doit être multi-threadé et donc pouvoir récupérer les images des différents site en parallèle.

L'architecture doit être assez générique pour permettre l'ajout de nouveaux site à moindre coût.

Description précise

— Entrée : Rien
— Sortie : Une liste d'image est sauvegardée
— Exception : Si le XMLTV n'est pas valide ou un site n'est pas en ligne.

2 Description des fonctionnalités du programme de sélection

2.1 Définition de la fonction 1 : Détecter les doublons

Identification de la fonction 1

Cette fonction permet à partir de plusieurs images de détecter les doublons.

Priorité : **Primordiale**

Description de la fonction fonction 1

A partir d'un dossier de la base de donnée (liste des images pour un programme TV) on récupère toutes les images du programme et on détecte les doublons. Une méthode de "Template Matching" sera appliqué via la transformation de Fourrier et plus particulièrement la méthode FFT. On a donc en entrée une liste d'image et en sortie la base ne possède plus de redondance. Pour vérifier que nos méthodes fonctionnent bien on vérifiera "A la main", c'est à dire que nous savons quels images sont des doublons ou non et nous pourrons ainsi déterminer si l'algorithme fonctionne.

Description précise

— Entrée : La liste des images enregistrée par le crawler
— Sortie : La liste d'image sans la redondance
— Exception : Un programme ne possède qu'une image.

2.2 Définition de la fonction 2 : Extraire les méta-donnés de l'image

Identification de la fonction 2

Cette fonction permet à partir d'une image d'extraire un certain nombre de ses méta-données.

Priorité : **Primordiale**

Description de la fonction fonction 2

Cette fonction prend en entrée une image et en sorti fourni un certain nombre de métadonnées (Dimension, copyright, format ...) Si possible les métadonnées seront extraites sans lire l'intégralité de l'image.

Description précise

— Entrée : La liste des images triées
— Sortie : Les images annotées de leur métadonnées
— Exception : L'image ne contient pas de métadonnées

2.3 Définition de la fonction 3 : Ajout des information dans la base

Identification de la fonction 3

Stockage des images sélectionnées avec divers information dans une base de données No SQl / XMLTV.

Priorité : **Primordiale**

Description de la fonction fonction 3

Nous stockons l'ensemble des programmes dans une base de données avec comme données associées la liste des images sélectionnées ainsi qu'une liste de méta-données et information sur le programme.

Description précise

— Entrée : La liste des images annotées
— Sortie : Un XMLTV contenant la liste des images et leur métadonnées
— Exception : /

D Spécification non fonctionnelles

1 Contraintes de développement et conception

Le programme final doit être portable pour pouvoir être déployé sur n'importe quel machine. Le langage de programmation qui a été adopté est le Java sous l'environnement de développement Eclipse avec l'utilisation de Maven pour l'intégration des différentes librairies (ImageJ, Metadata extractor, Jsoup).

2 Contraintes de fonctionnement et d'exploitation

2.1 Performance

Il n'y a pas de contraintes de performances dans ce système car les programmes étant disponible longtemps à l'avance on peut prendre le temps nécessaire pour procéder à la sélection des images. Pour ensuite les proposer à l'utilisateur.

2.2 Capacité

Il n'y a pas de limite fixées mais il serait profitable de pouvoir appliquer nos méthodes sur un très grand nombre d'images.

Via la création de la base de données mono-sources nous avons pu extraire certaine statistiques sur la quantité des images ajouté chaque semaine et donc avoir une idée du nombre d'images à traiter. (Figure 1 (Annexe D))

On remarque qu'il y a environ 1000 nouvelles images par semaines pour une seule sources on peut estimer que le nombre d'images à traiter par semaine sera donc de 1000 * nombres de sources.

2.3 Contrôlabilité

Pour suivre l'exécution du programme les logs seront affichés dans la console. Les résultats ainsi que quelques logs seront sauvegardé dans un fichier.

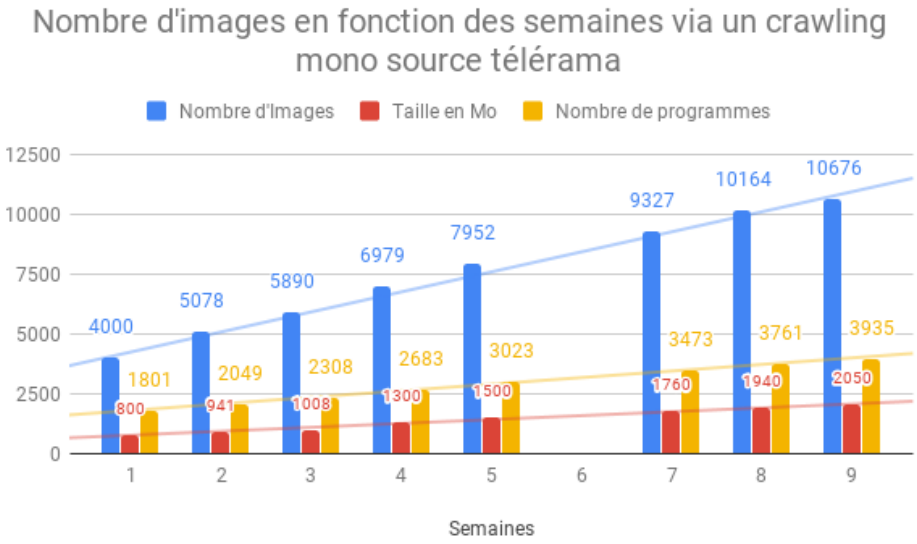


Figure 1 – *Statistiques d'ajout de programmes via la source Télérama*

2.4 Sécurité

Aucune demande particulière n'a été faite de la part du MOA en ce qui concerne la sécurité.

E Manuel utilisateur

1 Crawler multi-source

1.1 Prérequis

Il est nécessaire d'avoir le fichier xmltv.dtd trouvale ici dtd ainsi que le fichier de configuration trouvable ici config.json Voici comment se présente le fichier de configuration : Figure 1 (Annexe E) Dans le fichier il y a une partie pour les crawlers et un autre pour le downloader d'images. Il y a deux type de crawler, le crawler pour les xmltvs et les crawlers pour les sites web.

1.1.1 XMLTV

Il y a 4 paramètre pour la configuration des XMLTV

- "directory" indique où sera enregistré les fichiers xml qui seront téléchargé. Le fichier xmltv.dtd doit absolument se trouver dans ce dossier !
- "url" indique le site ou va être récupéré le xmltv
- "xml_ url" indique le lien exacte du xml
- "active" indique si l'on active ou non la recupération des programme via cette source

1.1.2 Sites

Pour chaque site il y a 3 paramètres :

- "nb_ thread" : Le nombre de thread alloué à ce site
- "timeout" : La durée de récupération des programme en ms
- "active" : Si on récupère les programmes via cette source ou non

1.1.3 Downloader

le downloader d'images et composé de 4 paramètre :

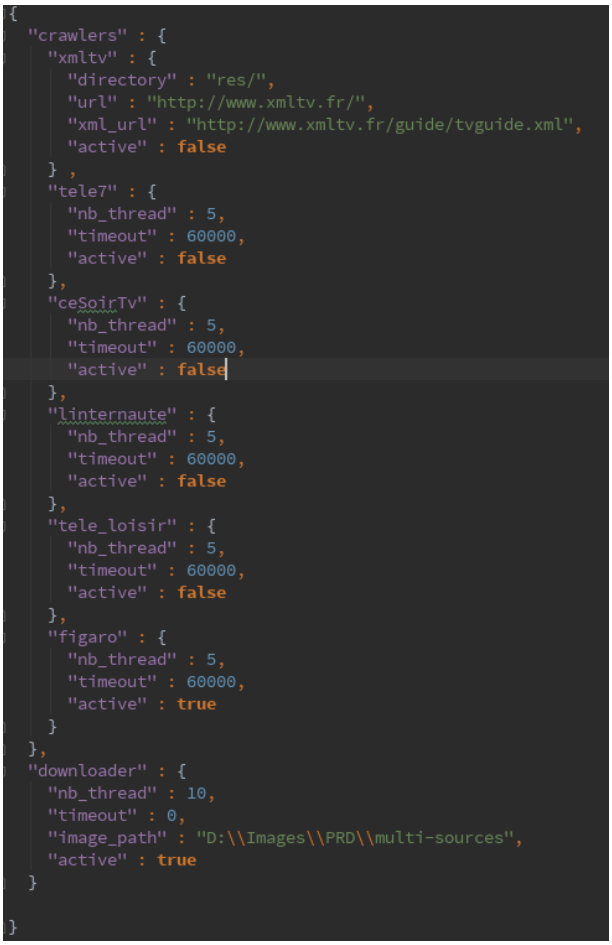


Figure 1 – *Fichier de configuration*

— "nb_ thread" : Le nombre de thread alloué au téléchargement des images
— "timeout" : La durée maximale de récupération des images, si 0 s'arrête une fois que toute les images ont été téléchargé
— "image_ path" : Le dossier où seront sauvegardé les images
— "active" : Si on télécharge les images ou non.

1.2 Lancement

Pour lancer le programme il faut ouvrir un terminal dans le dossier où se trouve le fichier crawler.jar. Ensuite nous exécutons la commande :

```
java -jar crawler.jar "fichierDeConfig.json"
```

Le programme va donc se lancer selon les instructions du fichier de configuration. Une fois le programme terminé un fichier de log va être écrit dans le dossier "res/"

F Documentation d'installation

Le projet utilise Maven, toute les dépendances sont incluent dans le fichier "pom.xml". Il est nécessaire d'avoir Java 8 et un IDE permettant l'utilisation de maven. IntelliJ est le plus recommandé car pour importer le projet il suffit d'ouvrir le fichier "pom.xml" et toutes les dépendances vont être téléchargées automatiquement et le projet est prêt à l'emploi. voir Figure 1 (Annexe F)

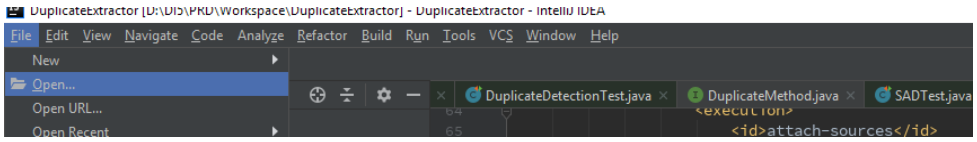


Figure 1 – *Etape 1*

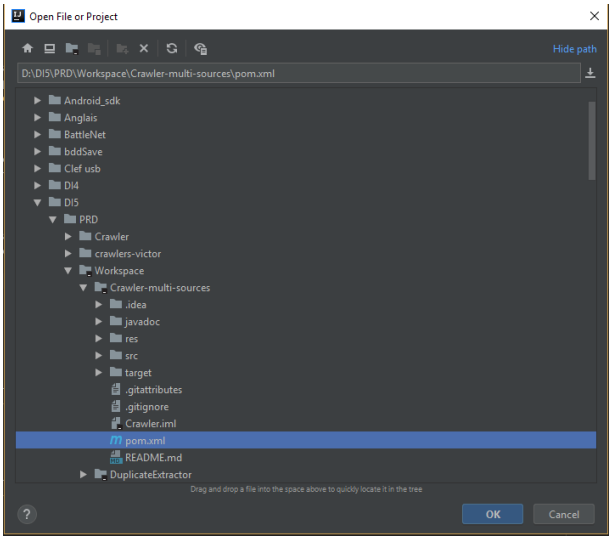


Figure 2 – *Etape 2*

Lorsque l'on est sur IntelliJ et que l'on ouvre le fichier "pom.xml" il est intéressant d'activer "l'auto import".

Si vous n'utilisez pas d'IDE vous pouvez compiler le projet grâce à la commande `mvn package` le .jar sera alors généré dans le dossier "target". Pour installer les dépendances nécessaires à la main il faudra alors lancer la commande `mvn dependency:resolve`.

G Cahier du développeur

1 Crawler multi-source

Voici le diagramme de classe du crawler (Figure 1 (Annexe G)) Le programme est divisé en 4

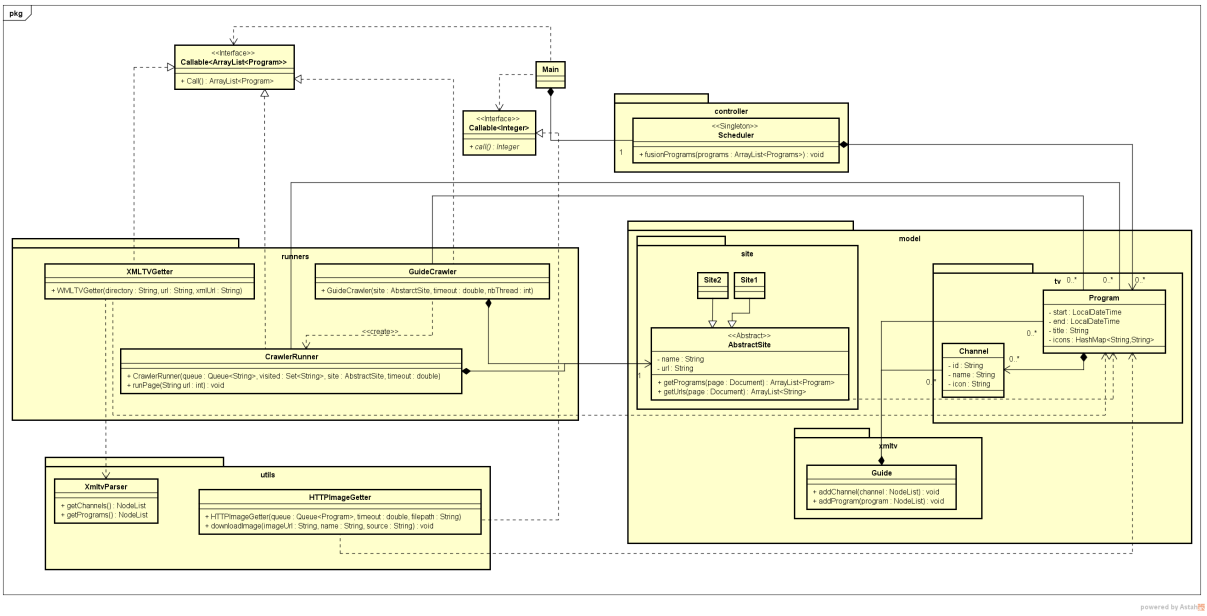


Figure 1 – *Diagramme de classe du crawler*

packages principaux :

Model

Ce sont les classes qui permettent de structurer les données. Il est lui même divisé en 3 package

site

Dans ce package il y a :

AbstractSite : C'est une classe abstraite qui permet de généraliser un site à crawler pour ensuite pouvoir ajouter de nombreux sites facilement.

Site... : Ce sont les classes filles de AbstactSite elles implémentent la méthode getPrograms qui à partir du page web extrait les programmes tv. Par exemple ici (Linternaute, CeSoirTv, LeFigaro ...)

xmltv

Guide : Cette classe permet d'extraire tous les programmes des xmltv et de les stocker.

tv

Channel : Cette classe permet de représenter un chaine TV surtout adapté pour les chaines inclues dans le XMLTV

Program : Cette classe permet de représenter un programme qui est l'élément principal du programme. Il contient une liste d'images stockées sous forme d'une hashMap dont la clé est l'url de l'image et la valeur est le nom du site d'où elle provient. Dans cette classe on redéfini les méthodes `hash()` et `equals()` car comme on récupère un programme depuis de nombreuses sources il est important de pouvoir savoir si se sont les même programme ou non. Ici un programme est égale si il possède le même titre.

Utils

Ce sont des classes utilitaires.

XmltvParser : Cette classe fournie des méthode utilitaire qui permettent de parser un fichier xmltv et en extraire tous les programmes.

HTTPImageGetter : Cette classe implémente `Callable<Integer>`, elle est donc lancé grâce à un `ExecutorServie`. Elle permet de télécharger les différentes images des programmes de manière multi-threadé avec le bon nom et dans le bon dossier. A la fin de la tâche elle retourne le nombre d'images qui ont été téléchargées.

Runners

Ce sont les classes qui exécuterons les différentes tâches de manière multi-threadé.

XMLTVGetter : Cette classe implemente `Callable<ArrayList<Program>>`. Elle représente donc une tâche et à la fin de cette tâches elle retourne tous les programmes qui ont été récupérés. Cette tâche a pour but de télécharger et de parser les fichier XMLTV.

GuideCrawler : Cette classe implémente `Callable<ArrayList<Program>>`. Chaque guideCrawler est associé à un seul site à crawler. Cette classe permet de créer plusieurs crawlerRunners qui seront chargées de parcourir les différentes pages du site en d'en extraire les programme. Cette classe permet donc d'agréger les résultats des différents runners.

CrawlerRunner : Cette classe implémente`Callable<ArrayList<Program>>`. Elle représente un runners, elle va parcourir un page de la queue. Et pour chaque page, en extraire tous les liens et les ajouter dans la queue globale, en extraire les programmes et passer au lien suivant. Un fois l'exécution terminée retourne tous les programmes qui ont été trouvés qui sera agrégé par le `GuideCrawler`

Controller

Ici il n'y a qu'une classe qui permettra de faire le lien entre tous les runners. Cette classe stocke tous les programmes qui ont été trouvés et en fait la fusion. Cette classe contient donc une fonction `fusionPrograms(ArrayList<Program>` qui permet de fusionner tous les programmes arrivant avec les programmes qui sont déjà présents.

La classe Main

Cette classe est la classe d'entrée du projet qui contient la méthode `main`. Voici les différentes étapes du lancement :

1. On va d'abord créer les objets JSON qui permettront d'extraire les informations du fichier de configuration données en argument du programme.

2. Ensuite nous créons un `ExecutorService` qui permettra d'exécuter nos différents crawlers

3. Nous exécutons tout les `GuideCrawler` et nous fusionnons les programmes retourné via le `Scheduler`.

4. Nous téléchargeons toues les images de manière multi-threadé

Le programme est donc doublement multi-threadé, on a 1 thread par "Site" et pour chaque site on a plusieurs Thread "runners" qui s'occuperont de parcourir les pages de manière parallèle.

1.1 Architecture du projet

Ci-dessous l'architecture du projet Figure 2 (Annexe G)

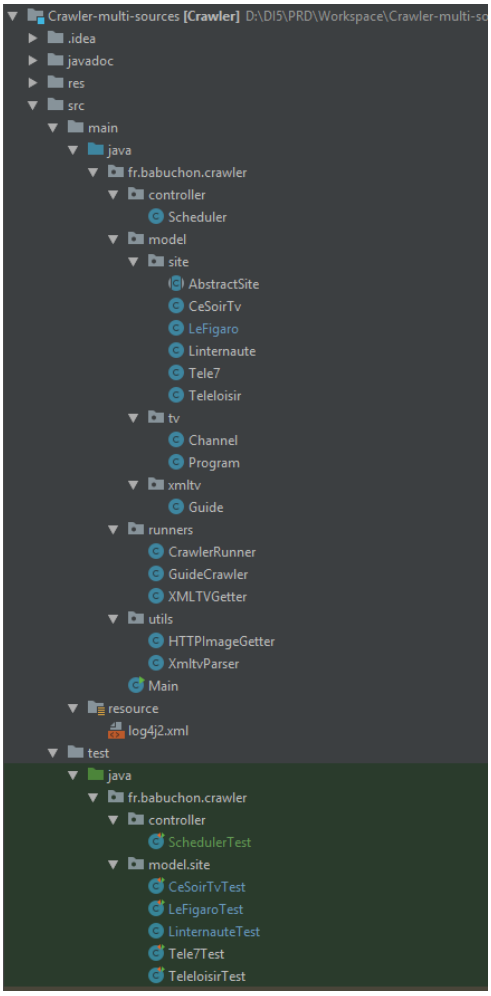


Figure 2 – *Structure du projet*

- Le dossier "javadoc" contient la javadoc généré

- Le dossier "res" contient des ressources (xmltv téléchargé, la dtd xmltx, le fichier config.json, les logs générées).

- Le dossier "src" est divisé en deux sous dossier :
  - main : Contient toutes les classes du projet. Ce dossier contient un fichier Ressource qui contient le fichier de configuration du crawler.
  - test : Contient toutes les classes test du projet.

1.2 Ajouter un nouveau site

Pour ajouter une nouvelle source au crawler il y a plusieurs étapes.

1. Créer une nouvelle classe dans le package site et la faire étendre de `AbstractSite`.

2. Ensuite il faut redefinir le constructeur :

```
public Teleloisir() {
    // On appelle le constructeur de la classe mère
        super();

    // On défini le nom du site (Evitez les espaces)
        this.name = "Tele_loisir";

        // On défini l'url de départ du crawler
        this.url = "https://www.programme-tv.net/";

        // Ajouter les url autorisées
        allowUrl.add(Pattern.compile("https://www\\.programme-tv\\.net/.*"));

      // Ainsi que les url non autorisées
      deniedUrl.add(Pattern.compile(UneRegex));

    }
```

3. Ensuite il nous faut définir le comportement du crawler lors de l'extraction d'une page. Il nous faut alors définir la fonction `getPrograms()`. C'est la méthode qui sera appelé à chaque exécution d'une page du site.

```
     /**
     * The programs url's page pattern
     */
    private static final Pattern PROGRAM_PATTERN = ↩
            Pattern.compile("https://www\\.cesoirtv\\.com/programme/.*");

@Override
    public ArrayList<Program> getPrograms(Document page) {
        ArrayList<Program> programs = new ArrayList<>();
        if(PROGRAM_PATTERN.matcher(page.location()).matches()) {
            Element titleElement = page.select(".Titre.d-ib").first();
            Element imageElement = page.select("meta[property=\"og:image\"]").first();

            String title = null;
            String imageUrl = null;

            if(titleElement != null && imageElement != null) {
                title = titleElement.text();
                imageUrl = imageElement.attr("abs:content");

            }

            if(imageUrl != null && title != null) {
                Program p = new Program(title);
```

```
                p.addIcon(trickImageUrl(imageUrl), name);
                programs.add(p);
            }
        }
        return programs;
    }
```

La méthode retourne la liste des programmes qui ont été extraits, ou une liste vide si il n'y a aucun programme.

4. Il faut ajouter le nouveau site au main ainsi qu'au fichier de configuration.

Pour le json il suffit de rajouter ces lignes dans la partie crawler.

```
"linternaute" : {
      "nb\_ thread" : 5,
      "timeout" : 60000,
      "active" : false
    }
```

Ensuite dans le main il faut ajouter :

```
// Rajouter une ligne pour le site
JSONObject tele7Object;
JSONObject teleLoisirObject;
JSONObject ceSoirTvObject;
JSONObject linternauteObject;
JSONObject figaroObject;

// Ici rajouter aussi pour récupérer l'object Json associé
// Dans le fichier de configuration
tele7Object = crawlersObject.getJSONObject("tele7");
teleLoisirObject = crawlersObject.getJSONObject("tele_loisir");
figaroObject = crawlersObject.getJSONObject("figaro");
linternauteObject = crawlersObject.getJSONObject("linternaute");
ceSoirTvObject = crawlersObject.getJSONObject("ceSoirTv");

// Derniere étape
// Rajouter le guideCrawler à l'executor service avec les bon paramètres
if(linternauteObject.getBoolean("active"))
            futures.add(service.submit(new GuideCrawler(new Linternaute(), ↩
                    linternauteObject.getDouble("timeout"), ↩
                    linternauteObject.getInt("nb_thread"))));
```

1.3 Lancer/compiler le programme via IDE

Pour lancer le programme dans l'IDE il faut penser à rajouter le fichier de configuration en argument du programme et bien avoir le fichier "xmltv.dtd" dans le dossier ou seront stockées les fichiers xmltv.

Pour compiler le programme il suffit de lancer la commande package de maven.

```
mvn package
```

2 Détection de doublons et extraction des métadonnés

Ce programme permet de tester des méthodes de sélection de doublons à partir d'une base de test, elle produit en sortie des fichiers avec toutes les distance suivant la nature de l'image.

Dupliqué ou différente. Grâce à ces fichier on peut faire des statistiques et voir quels méthodes sont les plus adaptées. Voici le diagramme de classe de la détection des doublons (Figure 3 (Annexe G))

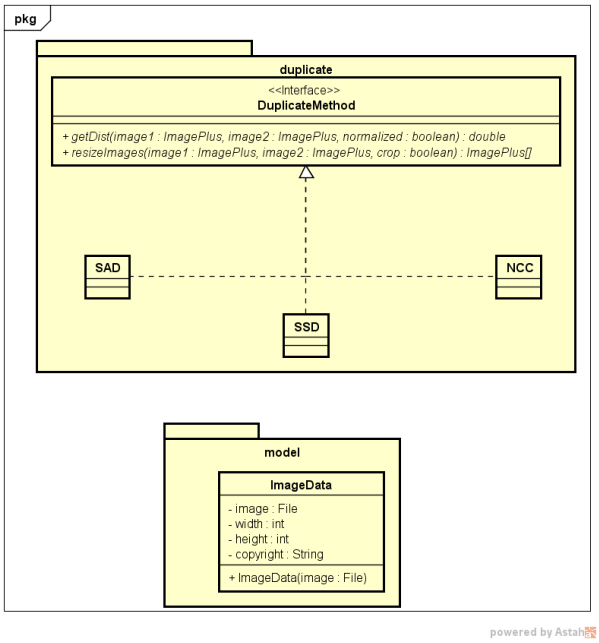


Figure 3 – *Diagramme de classe de la détection de doublons*

Dans ce programme il y a deux packages important :

Duplicate

DuplicateMethod : C'est une interface qui définie une méthode de calcul de distance entre deux images. Elle contient deux méthode, uné méthode `gestDist()` qui sera à implémenter selon la méthode. Et une méthode `resizeImage()` qui est implémenté par défaut et qui convertir les images en niveau de gris et les redimensionnement de la même taille.

SAD/NCC/SSD Sont des méthodes que j'ai implémentées.

Model

ImageData : Est une classe qui permet à partir d'une image d'extraire un certain nombre de ses métadonnées et de les stocker.

La classe Main

Elle possède un attribut "PATH" Qui correspond au dossier qui contient toutes les images à analyser sous cette forme :

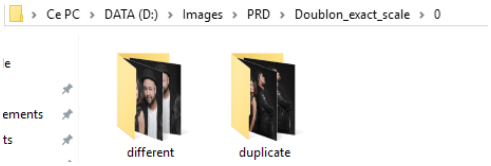


Figure 4 – *Organisation de la base de test*

On a un dossier par programme et dans chaque programme un dossier pour des images dupliqué et un dossier pour les images qui sont différentes des images dupliquées.

Ensuite le programme va calculer la distance entre toutes les images dupliquées puis entre toutes les images dupliquées et les images différentes. Ensuite on génère le fichier.

2.1 Ajouter une méthode

Pour ajouter une méthode il suffit de créer une classe qui implemente l'interface `DuplicateMethod` et de redéfinir la méthode `getDist()` Ensuite dans le main il faut rajouter la méthode dans le choix des méthodes disponibles.

```
if(METHOD.equals("ncc")) {
  method = new NCC();
}
else if(METHOD.equals("sad")) {
  method = new SAD();
}
else if(METHOD.equals("ssd")) {
  method = new SSD();
}
else {
  LOGGER.error("Invalid method");
  return;
}
```

2.2 Lancement du programme

Pour lancer le programme il faut définir tous les paramètres que l'ont souhaite au début du main.

```
private static final String PATH = "D:\\Images\\PRD\\Doublon_exact_scale";
private static final boolean CROP = true;
private static final boolean NORMALIZED = true;
private static final String METHOD = "ncc";
private static final String STATS_PATH = "res/stats_" + System.currentTimeMillis() + "_" ↩
        + METHOD + "_" + NORMALIZED + "_" + CROP + ".txt";
private static final int[] POW_2 = {16, 32, 64, 128, 256};
```

2.3 Les fichiers de sortie

Les fichiers sont écrit dans le dossier "/res" et contiennent toutes les distances qui ont été calculé avec d'abord les images dupliquées et ensuite les images différentes. Les résultats sont triés par distance croissante.

H Cahier de tests

1 Introduction

Les tests unitaires sont utilisés pour vérifier que les différentes méthodes font ce qu'elles sont censées faire. Par exemple en prenant des paramètres bien spécifique et en retournant le résultat attendu. Mon projet comportant deux programmes, le crawler, et le détecteur de doublons. Mon cahier de tests sera composée de deux parties.

2 Méthodes

J'ai fait deux type de tests, les tests unitaires ainsi que les tests fonctionnels. Les tests unitaires on été réalisée avec Junit et les tests fonctionnels "A la main", c'est à dire que l'on vérifie que le programme fait bien ce qu'on lui demande de faire.

3 Crawler multi-sources

Pour le crawler il n'y a pas beaucoup de tests unitaires car c'est des accès web, téléchargement de fichier. Mais j'ai quand même testé les méthodes qui pouvaient être testées.

3.1 Tests unitaires

L'architecture des fichiers de tests est standard, elle permet de lancer tous les tests facilement via maven (Figure 1 (Annexe H)).

Il y a une classe de test pour chaque site ainsi qu'une classe de test pour le controller, c'est lui qui va fusionner tous les programmes des différentes sources, c'est un classe critique, si ça ne fonctionne pas tout le logiciel est corrompu.

3.1.1 SchedulerTest

Cette classe permet de tester la bonne fusion des programmes et des images des différentes sources.

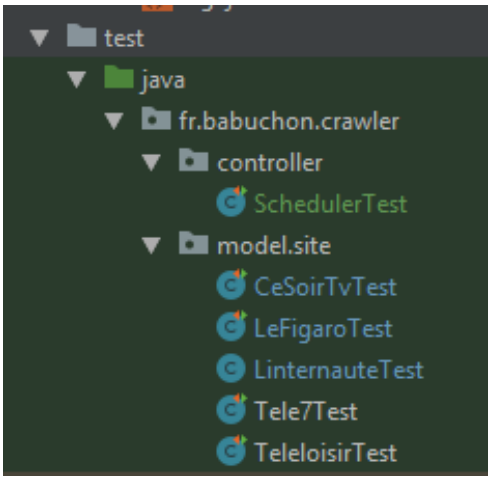


Figure 1 – *Organisation des tests pour le crawler*

— Nom de la méthode à tester : fusionPrograms()
— Description : Cette méthode permet de fusionner les programmes des différentes sources ainsi que les images associées
— Données en entrée : Deux programmes ayant le meme nom et des images différentes / deux programmes ayant des nom différents
— Données en sortie attendu : Un seul programme contenant toutes les images / Deux programmes avec chacun ses images
— Résultat réel : Ce qui était attendu

3.1.2 SitesTest

Les différents sites internet propose des images qui sont cropées automatiquement directement via l'url, il faut donc chercher un moyen à partir de l'url de récupérer l'image original. La méthode qui effectue cette modification se nomme `trickImageUrl()` elle prend l'url en entrée et retourne l'url de l'image original. J'ai donc testé que ce que la méthode retourne est bien ce que l'on attendais.

— Nom de la méthode à tester : `trickImageUrl()`
— Description : A partir de l'url d'une image sur le site retourne l'url de l'image originale
— Données en entrée : L'url de l'image cropée
— Données en sortie attendu : L'url de l'image originale
— Résultat réel : L'url de l'image originale

Exemple : Pour le site Lefigaro.

url :

"http ://i.f1g.fr/media/ext/1500x/api-tvmag.lefigaro.fr/img/000/251/25113974.jpg"

Après recherche on remarque que si l'on modifie 1500x par 5000x on récupère l'image de la taille la plus grande possible et donc l'originale.

url original :

"http ://i.f1g.fr/media/ext/5000x/api-tvmag.lefigaro.fr/img/000/251/25113974.jpg"

3.2 Test fonctionnels

Le programme prend en entré un fichier de configuration qui permet d'activer/désactiver les différentes sources. Il permet aussi de dire dans quel dossier seront stockés les images. Ainsi

que combien de threads sont allouées pour chaque sources.

ID	Contenu	Statut
1	Chargement que du XMLTV sans téléchargement	✓
2	Chargement que du XMLTV et téléchargement des images, pour vérifier que le téléchargement des images fonctionne	✓
3	Test de chaque site individuellement pour vérifier qu'il fonctionne bien	✓
4	Changement du nombre de threads des sites pour vérifier que c'est bien prit en compte	✓
5	Changement des timeout pour vérifier que ça fonctionne bien	✓

Table 1 – *Tests fonctionnels*

4 Détection de doublons et extraction des métadonnés

4.1 Tests Unitaire

Ce programme là est plus facile à tester car ce sont des méthodes basiques qui n'ont pas d'accès a internet/téléchargements. Ici toute les classes sont testées (Figure 2 (Annexe H)).

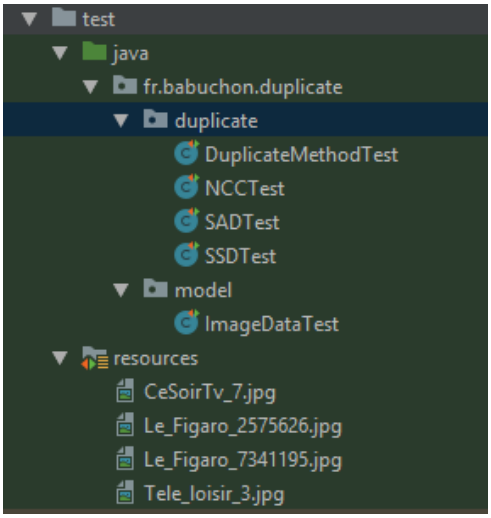


Figure 2 – *Organisation des tests pour la détection des doublons*

4.1.1 DuplicateMethodTest

Ici la méthode qui est testé est la méthode `resizeImages(ImagePlus image1, ImagePlus image2, boolean crop)`

- Nom de la méthode à tester : `resizeImages(ImagePlus image1, ImagePlus image2, boolean crop)`
- Description : A partir de deux images en entrée, converti les images en niveau de gris, les redimensionnes pour qu'elles aient la même taille, et si la variable crop est vrai, extrait uniquement le centre de l'image.
- Données en entrée : Les deux images et un booléen pour le crop

— Données en sortie attendu : Les deux images en niveaux de gris dimensionnées et cropé ou non selon les paramètres
— Résultat réel : Le résultat est bien le bon selon les différents paramètres

4.1.2 NCC/SAD/SSD Tests

Les trois classes qui sont testées représente une méthode calcul de la distance entre deux images. Pour les trois méthode on vérifie que les résultats retournées sont bien ceux qui sont attendues.

— Nom de la méthode à tester : `getDist()`
— Description : Retourne la distance entre deux images.
— Données en entrée : Les deux images et une variable booléenne qui indique si on normalise la distance ou non.
— Données en sortie attendu : La distance entre les deux images
— Résultat réel : La distance est bonne

4.1.3 ImageDataTest

Cette classe permet d'extraire et stocker les métadonnées d'une image.

— Nom de la méthode à tester : `ImageDate()`
— Description : Extrait les metadonnées et les stocke dans les attribut de la classe
— Données en entrée : Le chemin vers l'image
— Données en sortie attendu : Les valeurs extraite sont les bonnes
— Résultat réel : Les données extraites sont les bonnes

Comptes rendus hebdomadaires

Compte rendu n°1 du 17/09/2018

- Recherche et documentation sur le scrapping
- Recherche de nombreux sites de Guide médias/TV et de fournisseurs de contenus
- Analyse des différents sites trouvés (Contenu, mise à disposition des inforamtions, code html pour voir si il est facile à crawler ...)
- Pour une série donnée j'ai regardé l'ensemble des images mises à disposition sur ces sites pour analyser les différences de contenues.
- J'ai installé le modèle latex pour les PRD

Compte rendu n°2 du 24/09/2018

- Début de la rédaction du rapport et recherches associées.
- Renseignement sur Jsoup, XMLTV ainsi que comment parser le XML en Java
- Début de la création du crawler mono-source XMLTV

Compte rendu n°3 du 01/10/2018

- Finition du crawler mono-source et téléchargement des images associées (1801 programmes TV, 4000 images pour 80 Mo)
- Première version du rapport avec les parties (Acteur, enjeux, contexte et objectif)
- Lectures d'articles concernant l'importance des choix des images par Netflix

Compte rendu n°4 du 08/10/2018

- Début de l'état de l'art
- Renseignements sur l'IQA

Compte rendu n°5 du 15/10/2018

- Avancement de l'état de l'art
- Amélioration de la partie contexte et objectifs du rapport
- Début du cahier de spécification

Compte rendu n°6 du 22/10/2018

- Avancement du cahier de spécification

— Prise en main d'imageJ
— Début de l'analyse des métadonnées sur la base.

Compte rendu n°7 du 05/11/2018

— Formation au différents type de metadonnées
— Test de différentes manières d'extraire les metadonnées (Java pur, ImageJ, drew metadata-extractor) et extraction d'un certain nombre d'information sur la base.
— Test d'utilisation de la FFT sous imageJ non concluante

Compte rendu n°8 du 12/11/2018

— Avancement du rapport
— Avancement de l'état de l'art

Compte rendu n°9 du 19/11/2018

— Changement d'objectif on enlève la partie IQA du projet et on se concentre sur la partie crawler, detection des doublons, extraction de métadonnées
— Ajustement du rapport / cahier de spécification en fonction

Compte rendu n°10 du 26/11/2018

— Etat de l'art Crawler / metadonnées
— Amélioration spec fonctionnelle / non fonctionnelle
— Création d'un diagramme UML "Use Case"
— Création d'un diagramme de composants

Compte rendu n°11 du 03/12/2018

— Fin du cahier de specification
— Etat de l'art sur la detection des doublons
— Partie analyse et conception du rapport
— Planification et diagramme de gantt
— Finition du rapport

Compte rendu n°12 du 17/12/2018

Début du développement du crawler multi-sources. Il contient les fonctionnalités suivantes

— Fichier de configuration en JSON
— Extraction des programmes XMLTV/Tele 7 jour / Télé loisir de manière multi-threadé
— Architecture permettant l'ajout d'un nouveau site en quelque ligne de code
— Téléchargement des images de manière multi-threadé selon l'architecture du crawler mono-source (Un dossier par programme, nom de l'image : "Source_Nom.jpg")

Compte rendu n°13 du 07/01/2019

— Ajout de robustesse et fiabilité au crawler
— Ajout d'une nouvelle source LeFigaro, cette source est très intéressante car pour un programme il y a plusieurs images associées (Parfois plus de 10)

Compte rendu n°14 du 14/01/2019

— Ajout de javadoc

— Correction de plusieurs bug
— Amélioration du logger
— Téléchargement de nombreuses images

Compte rendu n°15 du 21/01/2019

— Nombreux tests et ajustement pour la méthode FFT sous MATLAB car je n'arrive pas à avoir des résultats qui me conviennent.
— Implémentation de la méthode du SAD pour voir si ça fonctionne mieux sous MATLAB, en effet ça fonctionne mieux mais c'est beaucoup plus lent.

Compte rendu n°16 du 28/01/2019

— Tests avec la FFT et implémentation de la validation croisé dans le domaine des fréquences
— Construction d'une base de tests constituées de doublons
— Après étude visuel sur cette base on peut remarquer les cas suivants :
  — Image identique (Dans la plupart des cas)
  — Image redimensionnée (assez fréquent)
  — Image cropée (Peu courant)
  — Image redimensionnée + cropée (Peu courant)

Compte rendu n°17 du 04/02/2019

— Construction d'une nouvelle base de test avec pour les doublons que des images identiques ou redimensionné. Ajout d'un autre dossier avec que des images différentes aux doublons
— Début du programme java avec la pipeline (pour chaque couple d'images dupliquées et différentes : niveaux de gris-> redimensionnent -> SAD).

Compte rendu n°18 du 11/02/2019

— Implémentation De la méthode SAD, NCC, SSD, les méthodes sont implémentés de la manière "exact match" c'est à dire qu'on parcours une seul fois toute l'image. Après plusieurs tests il s'avère que c'est la méthode NCC qui est la plus précise. Sur la base de test (54 images dupliquées et 109 différentes) on retrouve que le plus corrélé des différent est 0,39 et le moins corrélé des similaires est 0,78. Suite à ces résultats et observations sur la base il s'avère que quelque fois les images sont presque identiques mais sont décalées de quelque pixels c'est pour cela que la corrélation pour des images "identiques" ou redimensionné la corrélation est un peu plus basse. Ce problème de décalage devrait être fixé lors d'une implémentation du type template matching.
— Implémentation de l'extraction des métadonnées et de tests unitaires associés, les données extraites pour le moment sont : Copyright, longueur, largeur.

Compte rendu n°19 du 25/02/2019

— Début des tests unitaires
— Modélisation uml pour le crawler
— Corrections de bugs

Compte rendu n°20 du 04/03/2019

— Ajout de 2 nouveaux sites (cesoirtv, linternaute)
— Augmentation de la base de test (112 dupliqués / 234 différents)

— Normalisation de SAD et SSD
— Génération d'un graphe à partir des statistiques fournis en sortie de la détection des doublons.

Compte rendu n°21 du 11/03/2019

— Diagramme de classe des deux programmes
— Tests unitaires des deux programmes
— Cahier de tests
— Manuel utilisateur
— Document d'installation
— Manuel du développeur

Compte rendu n°22 du 18/03/2019

— Agrandissement de la base de données de test : 200 images dupliquées/ 350 images différentes.
— Changement des histogrammes en courbe de fréquences.
— Calcul de la precision and Recall pour les 3 méthodes.
— Préparation de l'oral

Compte rendu n°23 du 25/03/2019

— Finition du rapport
— Création des posters

Webographie

[WWW0] DREWNOAKES. *metadata-extractor*. 9 déc. 2018. URL : https://github.com/drewnoakes/metadata-extractor.

[WWW0] EXIV2, éd. *EXIF tag*. 9 déc. 2018. URL : http://www.exiv2.org/tags.html.

[WWW0] XMLTV FR, éd. *XMLTV*. 9 déc. 2018. URL : http://www.xmltv.fr/.

[WWW0] JSOUP. *JSoup*. Sous la dir. de JSOUP. 9 déc. 2018. URL : https://jsoup.org/.

[WWW0] NETFLIX. *Extracting image metadata at scale*. 3 oct. 2018. URL : https://medium.com/netflix-techblog/extracting-image-metadata-at-scale-c89c60a2b9d2.

[WWW0] NETFLIX. *Netflix explains video thumbnail testing*. 3 oct. 2018. URL : https://www.engadget.com/2016/05/03/netflix-explains-video-thumbnail-testing/?guccounter=1.

[WWW0] NETFLIX. *Selecting the best video artwork*. 3 oct. 2018. URL : https://medium.com/netflix-techblog/selecting-the-best-artwork-for-videos-through-a-b-testing-f6155c4595f6.

[WWW0] NETFLIX. *The power of a picture*. 4 oct. 2018. URL : https://media.netflix.com/en/company-blog/the-power-of-a-picture.

[WWW0] OUTSOURCEIT. *Comparison of Open Source Web Crawlers for Data mining and web scraping*. 9 déc. 2018. URL : https://outsourceit.today/comparison-open-source-web-crawlers/.

[WWW0] PRINCETON. *FFT java*. 9 déc. 2018. URL : https://introcs.cs.princeton.edu/java/97data/FFT.java.html.

[WWW0] TELERAMA, éd. *Telerama Guide TV*. 9 déc. 2018. URL : https://television.telerama.fr/tele/grille.

[WWW0] WIKIPEDIA, éd. *Exif*. 9 déc. 2018. URL : https://en.wikipedia.org/wiki/Exif.

[WWW0] WIKIPEDIA, éd. *Template Matching*. 9 déc. 2018. URL : https://en.wikipedia.org/wiki/Template_matching.

[WWW0] WIKIPEDIA. *Web scraping*. 9 déc. 2018. URL : https://fr.wikipedia.org/wiki/Web_scraping.

[WWW0] ZENITH. *Mobile internet to reach 28 percent of media use in 2020*. 17 oct. 2018. URL : https://www.zenithmedia.com/mobile-internet-to-reach-28-of-media-use-in-2020/.

Bibliographie

[0] Christopher D.Mannig, Prabharkar Raghavan et Hinrich Scutze. *Introduction to Information Retrieval*. Sous la dir. de Cambridge University Press. 2008. Chap. 20.

[0] Ryan Mitchell. *Web scraping with python*. Sous la dir. de Simon St.Laurent et Allyson MacDonald. 2015. isbn : 978-1-491-91027-6.

[0] Mark S.Nixon et Alberto S.Aguado. *Feature Extraction & Image Processing for Computer Vision*. 2012, p. 223-235. isbn : 978-0-12-39654-3.

Projet Recherche & Développement

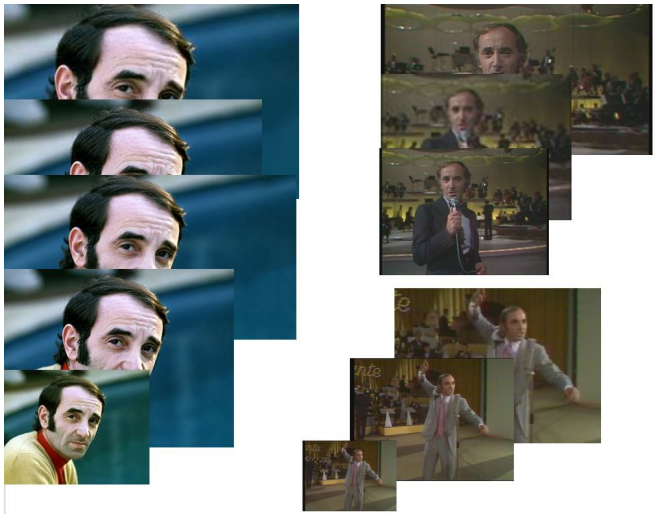
Scraping d'image smart pour portail guide media TV/vidéo

Louis Babuchon

Encadrement : Mathieu Delalandre

Contexte

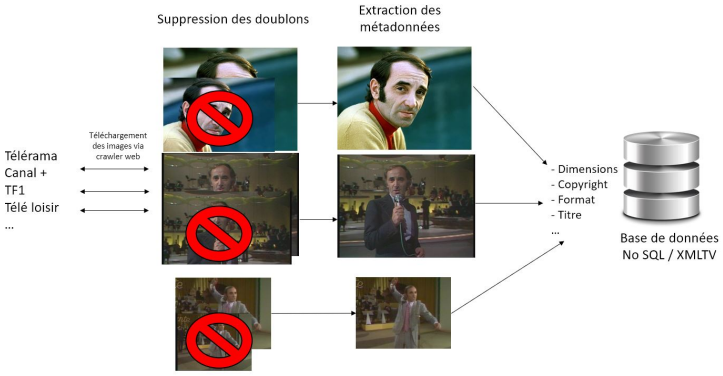
De nombreux sites proposent des Guide Médias TV. Et chaque site propose des images pour chaque programme TV. Le problème est la redondance des données. En effet sur les 20 premiers programme TV sur google, pour un programme donnée il y a seulement 3 images différentes qui peuvent être dimensionné, croppé, changé de teinte.



Objectif

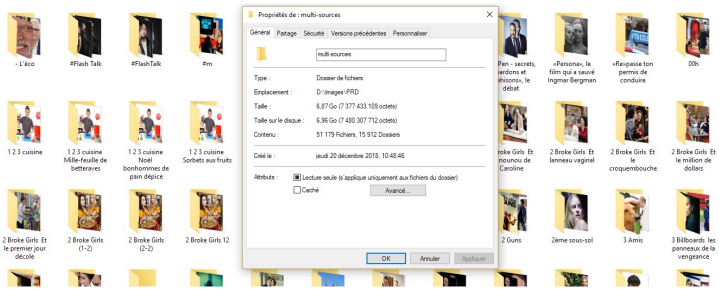
Le projet est composé de 3 objectifs principaux :

- Création d'un crawler multi-sources qui permet de télécharger les images de manière automatique
- Détection des doublons avec des méthodes d'analyses d'images
- Extraction des métadonnées.



Application

Ce sujet permet de créer une base de données enrichie pour des programmes TV. Mais il pourrait aussi être appliqué à de nombreux autres domaines.



Base de données (52 000 images pour 16 000 programmes)

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Projet Recherche & Développement

Scraping d'image smart pour portail guide media TV/vidéo

Louis Babuchon

Encadrement : Mathieu Delalandre

Contexte

De nombreux sites proposent des Guide Médias TV. Et chaque site propose des images pour chaque programme TV. Le problème est la redondance des données. En effet sur les 20 premiers programme TV sur google, pour un programme donnée il y a seulement 3 images différentes qui peuvent être dimensionné, croppé, changé de teinte.

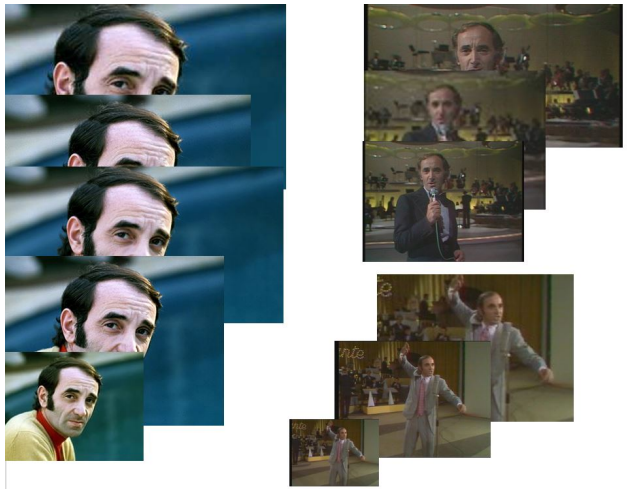
Objectif

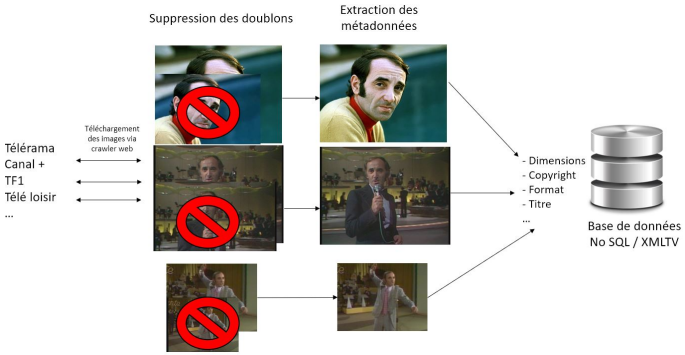
Le projet est composé de 3 objectifs principaux :

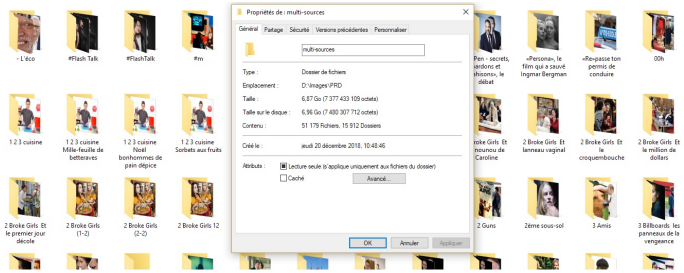
- Création d'un crawler multi-sources qui permet de télécharger les images de manière automatique
- Détection des doublons avec des méthodes d'analyses d'images
- Extraction des métadonnées.

Application

Ce sujet permet de créer une base de données enrichie pour des programmes TV. Mais il pourrait aussi être appliqué à de nombreux autres domaines.







Base de données (52 000 images pour 16 000 programmes)

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Scraping d'image smart pour portail guide media TV/vidéo

Résumé

Ce document est le rapport du projet de recherche et développement "Scraping d'image smart pour portail guide média TV/Video" Il permet de comprendre le contexte et les enjeux de ce projet et ainsi les objectifs attendus. Le rapport contient l'état de l'art, le cahier de spécification ainsi qu'une analyse permettant de mettre au point la solution au problème.

Mots-clés

Crawler, FFT, Analyse d'image, Média, Guide TV, PRD, LaTeX

Abstract

This report is about the research and development project "Smart image scraping for media guide TV/Video portal" It allows to understand the context and issues about this topic as well as the expected objectives. This report contains the state of the art, the spécification book as well as an analysis to develop the solution to the problem.

Keywords

Crawler, FFT, Image processing, Media, Tv guide, PRD, LaTeX

Tuteur académique
Mathieu Delalandre

Étudiant
Louis Babuchon (DI5)