

# Compte Rendu INF1603

## Projet de développement

Louison RABREAU                      Alexis GOHIER  
rabreau.e2003851@etud.univ-ubs.fr      gohier.e1704272@etud.univ-ubs.fr  
Janvier-Mai 2023

## Sommaire

Abstract :	-----	Page 1
Sprint 1 à 5 :	-----	Page 1-3
Sprint 6 :	-----	Page 3-4
Explication du système :	-----	
Résultats :	-----	
Conclusion :	-----	

## Abstract

Pour pouvoir lire plus efficacement des articles scientifiques, les chercheurs de l'Irisa nous ont demandé de créer un parseur ou convertisseur d'articles au format texte, suivi du format Xml. Ce compte-rendu résume les différents avancements du projet, sprint après sprint ainsi que le fonctionnement du projet final et ses résultats.

## 1 Sprint 1 à 5 - L'avancement du projet




### Sprint 1 - Choix parseurs de format .pdf en format .txt

Consignes :

1. Evaluer les différents logiciels de codage et convertisseurs disponibles.
2. Choisir les plus adaptés.
3. Sélectionner les options qui pourraient être utiles.

Temps accordé : du 27 janvier au 3 février 2023.

Carnet de Sprint

Non fait		En continu	Terminés
Trouver un langage de programmation	Trouver un convertisseur	Artefacts	
 Obligatoire et important	 Demandé	 "Optionnel" (non demandé)	

Après avoir décidé d'utiliser Python comme logiciel, étant, d'après nous, le meilleur à utiliser ainsi que le plus simple, nous avons passé la première semaine à essayer le plus de convertisseurs possibles, ainsi que leurs options pour y déceler les plus efficaces.

Nous avons donc choisi 5 modules de conversion différents, ayant tous leurs qualités et leurs défauts ; PDF Miner, Aspose Word, PyPDF et PyPDF2 ainsi que le choix "final" PDFIUM. Ces choix restent temporaires, car nous pouvons toujours trouver mieux, plus efficace et/ou plus simple.

## Sprint 2 - Parseur de format .pdf en format .txt

Consignes :

1. Programme qui produit en sortie un fichier .txt contenant nom, titre, auteur(s) et abstract.
2. Prend en entrée un dossier.
3. (Re)Crée un sous-dossier contenant la(es) sortie(s).

Temps accordé : du 07 au 17 février 2023.

Carnet de Sprint					
Non fait		En continu		Terminés	
Programme .pdf → .txt	Extraire nom, titre, auteur(s), abstract	Artefacts	Organisation logistique	Trouver un langage de programmation	Trouver un convertisseur
Exécution par ligne de commande	Sous-dossier de sortie	Amélioration efficacité	Gestion GIT/GITHUB		
		README			
Obligatoire et important		Demandé		"Optionnel" (non demandé)	

Pour un meilleur départ de programmation, nous avons développé l'architecture suivante ; les fichiers .pdf sont convertis en un fichier brut .txt via les convertisseurs, on les passe ensuite dans un filtre pour choisir le(s) meilleur(s) résultat(s) pour ensuite pouvoir sectionner les différentes parties voulues dans la sortie .txt.

À la fin du sprint, le programme possédait une efficacité de 60% en étant peu organisé. Il nous a fallu donc le revoir par la suite, et améliorer le "filtre" pour continuer de manière plus performante.

## Sprint 3 - Parseur de format .pdf en format .xml

Consignes :

1. Programme qui produit en sortie un fichier .xml contenant nom, preambule, titre, auteur(s) et leur(s) mail(s), abstract et bibliographie.
2. Option d'exécution -t pour une sortie .txt ou -x pour une sortie .xml.

Temps accordé : du 03 au 12 mars 2023.

Carnet de Sprint					
Non fait		En continu		Terminés	
Programme .pdf → .xml	Options -x (.xml) -t (.txt)	Artefacts	Organisation logistique	Trouver un langage de programmation	Trouver un convertisseur
Extraire mail(s), bibliographie		Amélioration efficacité	Gestion GIT/GITHUB	Programme .pdf → .txt	Extraire nom, titre, auteur(s), abstract
		README		Exécution par ligne de commande	Sous-dossier de sortie
Obligatoire et important		Demandé		"Optionnel" (non demandé)	

Le filtre étant complexe à fournir, nous avons décidé de nous concentrer que sur un seul convertisseur (PDFIUM), pour pouvoir résoudre un à un les problèmes rencontrés. En effet, nous avons du mal à récupérer mail, lier ceux-ci aux noms, obtenir la bibliographie et gérer les options de lancement.

Tous ceci nous ammena à devoir faire un refactoring du projet, ce que l'on fit lors du sprint suivant.

## Sprint 4 - Enrichissement du parseur

Consignes :

1. Ajouter les balises d'affiliation pour le(s) auteur(s), l'introduction, le corps et la conclusion.
2. Menu textuel lors de l'exécution pour le choix des fichiers d'entrée.

Temps accordé : du 20 au 31 mars 2023.

Carnet de Sprint

Non fait		En continu		Terminés	
Menu textuel	Extraire affiliation(s), introduction, corp, conclusion	Artefacts	Organisation logistique	Trouver un langage de programmation	Trouver un convertisseur
Extraire mail(s), bibliographie		Amélioration efficacité	Gestion GIT/GITHUB	Programme .pdf → .txt	Programme .pdf → .xml
		README		Options -x (.xml) -t (.txt)	Sous-dossier de sortie
				Extraire nom, titre, auteur(s), abstract	Exécution par ligne de commande

■ Obligatoire et important     
 ■ Demandé     
 ■ "Optionnel" (non demandé)

Le refactoring se fit tout d'abord par le changement de module de conversion (PYMUPDF). L'architecture du programme dû aussi subir une mise à jour ; les fichiers d'entrée .pdf passent dans un "Carnet" qui récupère leur version brute .txt et les sépare en blocs contenant le texte et leur police d'écriture. Ces blocs sont ensuite envoyés dans un "Classeur" qui associe les différents tags demandés aux valeurs correspondantes pour la sortie .xml ou .txt.

Ces avancements restent à finir mais permettent un programme plus propre à lire tout en étant plus efficace et plus fonctionnel. Cependant certaines extractions restent à faire.

## Sprint 5 - Evaluation du parseur

Consignes :

1. Evaluer les sorties .xml via le site <http://inf1603.alwaysdata.net/ParserResultComparator.php>.

Temps accordé : du 03 au 09 avril 2023.

Après la fin du refactoring et ajout des dernières extractions, le projet nous fournit une précision moyenne d'environ 70%.

Les erreurs étaient principalement dues à la récupération des mails, qui est déterminante dans l'obtention des auteurs et affiliations. Les numéros de bas de page ainsi que les différences de police engendraient également des soucis pour la récupération de l'abstract, introduction, conclusion et de la bibliographie. Ceux-ci étaient donc des problèmes à résoudre d'ici le rendu final.

## 2 Sprint 6 - Fin du projet

### Conclusion et tests finaux

Consignes :

1. Evaluer le programme sur de nouveaux .pdf.
2. Finir le programme.

Temps accordé : du 17 avril au 13 mai 2023.

## A) Explication du système

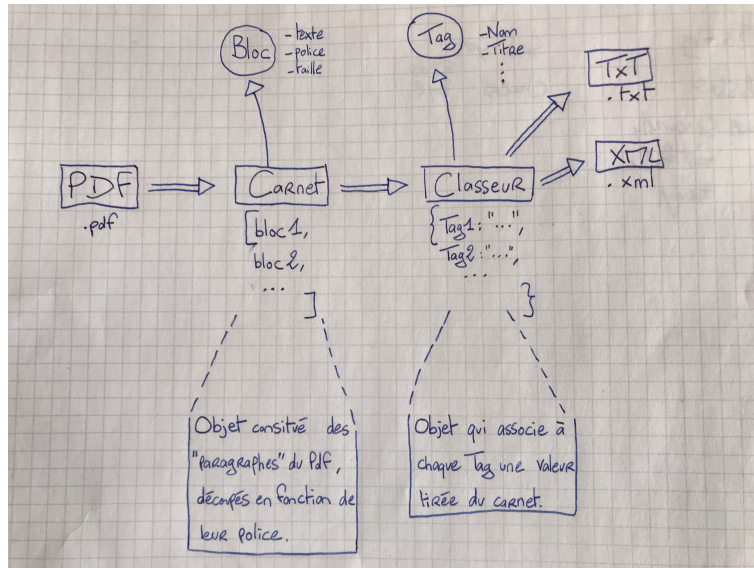


Schéma de l'architecture du projet

**Classe main :** Appelle les différentes classes dans leurs ordres d'apparition dans le programme (voir "#- Nom section-"). Après avoir **importé** tous les modules utilisés (Modules distribués) ainsi que les différentes classes spécifiques au projet (Modules du projet), les **arguments** entrés lors de l'exécution sont testés (Données statiques). Un **menu textuel** est ensuite créé pour que l'utilisateur choisisse les fichiers .pdf à convertir (Variables globales) en utilisant la **classe PDF**. Pour finir, pour chaque .pdf à convertir, le code crée un **objet Carnet** du fichier correspondant, qu'il entre en argument d'un **objet Classeur**. Le format, déterminé dans la section "Données statiques" permet d'indiquer au classeur quel type de sortie est voulu.

**Classe PDF :** Représente le fichier pdf, avec son chemin et son nom en attribut, et peut l'ouvrir avec la **méthode open** en utilisant PYMUPDF.

**Classe Carnet :** Représente une **liste** de la classe Bloc et de pages (composés elles même de Bloc), qui lors de son initialisation crée ces derniers selon les lignes contigües du fichier source ayant la même **police d'écriture**.

Cette classe possède autant de méthodes que de tags pour trouver ces derniers dans le texte. Chacun de ses **getteurs** recherche selon des supposés (ex : chaque nom est écrit avec la même police). Elle possède également une méthode qui retourne la plus petite distance entre deux éléments pour associer noms et affiliations, un testeur de similarité syntaxique pour associer noms et mails, ainsi qu'une correction de l'accentuation qui est souvent faussée ("è" avec le convertisseur nous donnera "e").

**Classe Bloc :** Héritant de la classe **string**, chaque bloc possède une police et une taille d'écriture, une position ainsi que la partie du texte correspondante. Lors de l'initialisation, une valeur est donnée en string à sa super-classe, la méthode fixe permet de remplacer cette valeur par la partie du texte voulue.

**Classe Classeur :** Comporte la classe **Tag** qui permet de déterminer toutes les sections à séparer lors du formatage (abstract, auteurs, bibliographie, ...) et leur écriture dans le fichier de sortie. L'initialisation nécessite l'objet carnet correspondant qui lui donne toutes les valeurs des tags du fichier à convertir. Cette classe possède une méthode qui permet de produire un fichier simple de sortie en format .txt ou un fichier détaillé grâce aux balises de son format .xml.

## B) Résultats

Sur le premier corpus, notre moyenne de précision était de 69.8 tandis que sur le nouveau elle fut de 70.9 ce qui prouvait la solidité et la robustesse de notre système. Après les dernières modifications, descriptions du programme et vérifications notre score dépassa toutes nos attentes ;

$$(93.81 + 94.59 + 90.34 + 81.29 + 92.9 + 77.76 + 84.42 + 94.6 + 89.37 + 80.32) \div 10 = 87.94$$

## C) Conclusion

Malgré les différents problèmes causés par le nombre plus réduit de personnes dans notre groupe, nous avons réussi à nous entendre, à se partager les problèmes et avancer convenablement. Certaines consignes nous ont demandés plus de réflexion que d'autres mais aucune ne nous a empêché de travailler.